SLCD5 Reference Manual
Version 1.34

Firmware Version 1.2.9 and BMPload 2.2.0 and later

May 10, 2010
Hardware Revisions A thru D

© Copyright Reach Technology Inc. 2003-2010
All Rights Reserved

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

www.reachtech.com
(503) 675-6464
sales@reachtech.com

Table of Contents

Figures

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, ROFIT, 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.

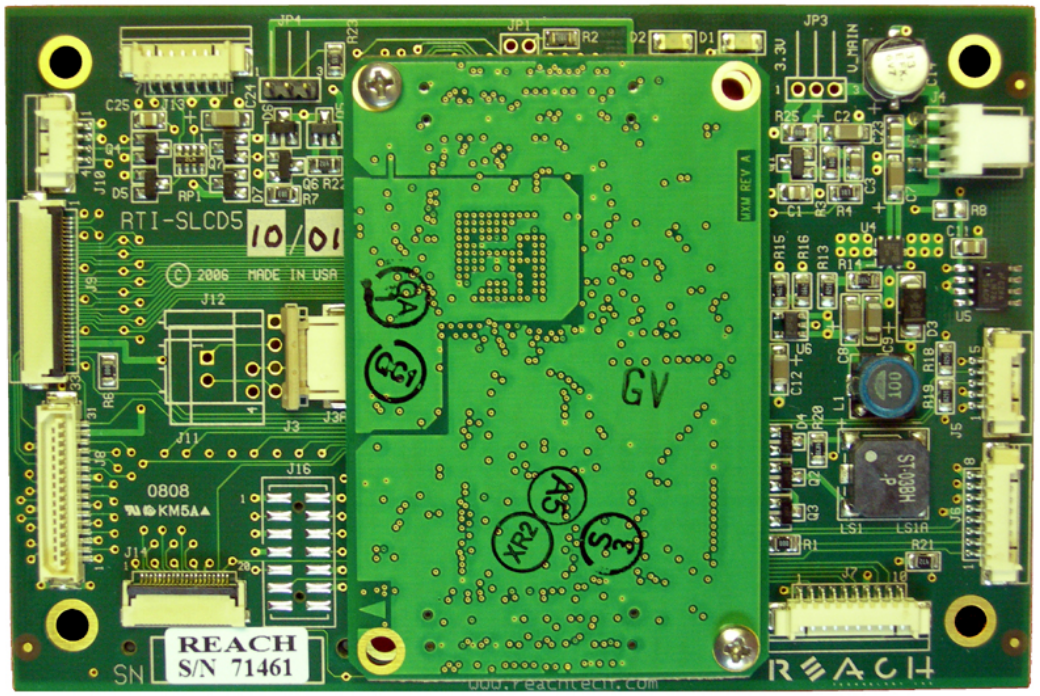
Software, including technical data, is subject to U.S. export control laws, including the U.S. Export Administration Act and its associated regulations, and may be subject to export or import regulations in other countries. Customer agrees to comply strictly with all such regulations and acknowledges 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 provides complete Graphical User Interface for embedded systems using QVGA or VGA 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.



1.2. *Features*

- Drives digital TFT displays at QVGA and VGA 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 200MHz
- On-board RS232 and TTL level interfaces up to 230,400 baud
- 4MB data flash for user downloadable bitmaps with 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. *Dimensions*

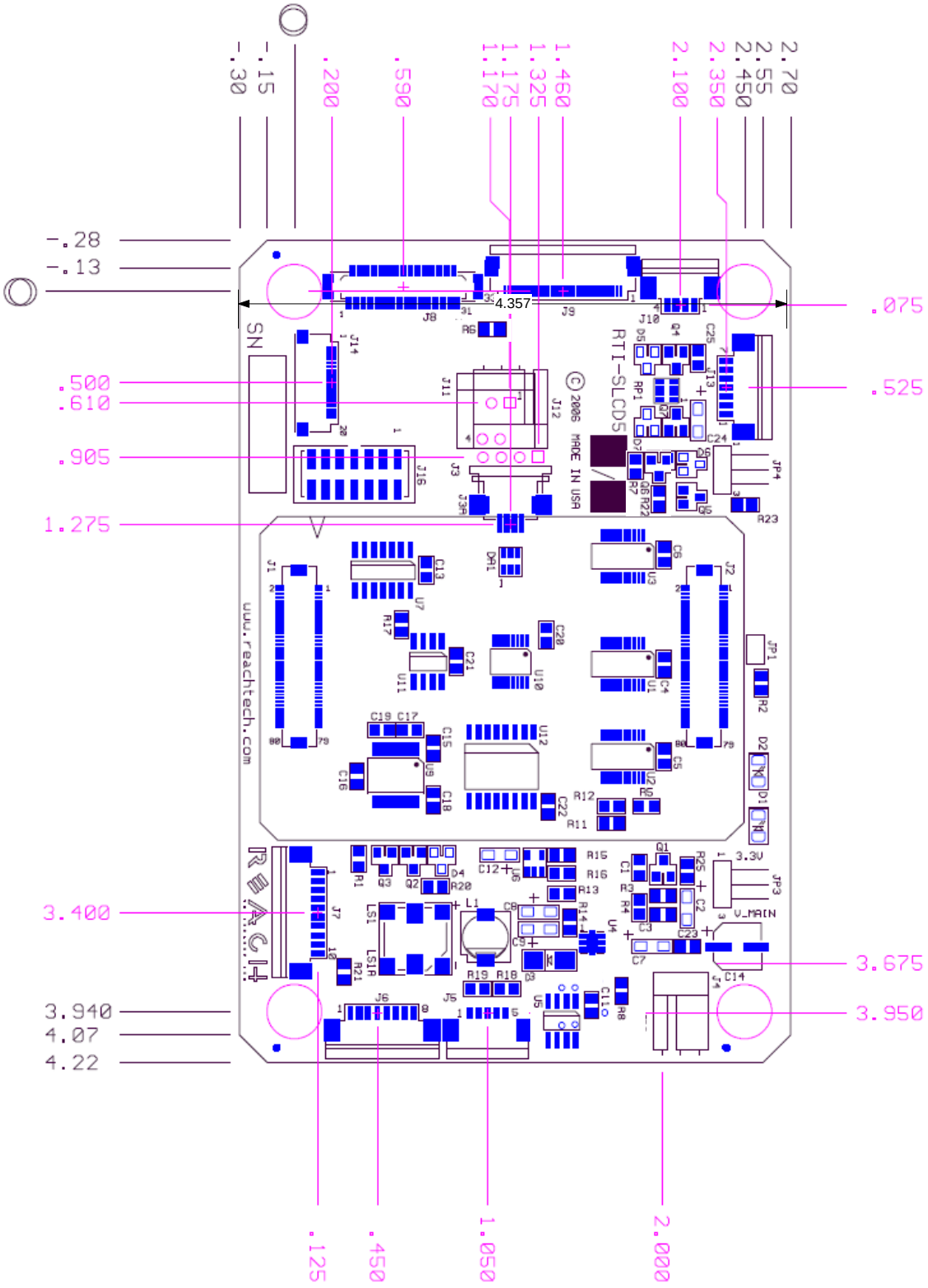


Figure 1: SLCD5 (Rev B) dimensions and connector locations (inches)

1.4. *Electrical Characteristics*

The SLCD5 is powered from 5-12V DC +/- 10%. It has an on-board switching regulator that generates the 3.3V for the panel and the SLCD5 circuitry.

It drives 3.3V LCD panels that have a 3.3V "CMOS" level compatible data interface. A list of compatible panels in provided in Appendix A.

SLCD5 Board Only Power Supply requirement: (SLCD5 ONLY; you must add input-referenced panel power plus inverter for total power requirement)

Voltage:	**Typical Current (beeper off):**	**Typical Current (beeper on full):**
5V	230mA (typ.)	300mA (typ.)
12V	150mA (typ.)	220mA (typ.)

1.5. *Panel support*

See Appendix A.

1.6. *SD card support*

The SD card slot supports cards that are FAT 16 formatted. We have tested and recommend SanDisk brand cards. You may need to format them on the PC before use with FAT format selected (***DO NOT USE "FAT32"***).

1.7. *Rev A vs. Rev B board Fab*

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. Fom the back side, the board revision is etched in the trace layer.

Rev A	**Rev B**	**Explanation**
JP2 Present R8 not present	JP2 not present R8 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 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 sych 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

2. Connectors and Jumpers

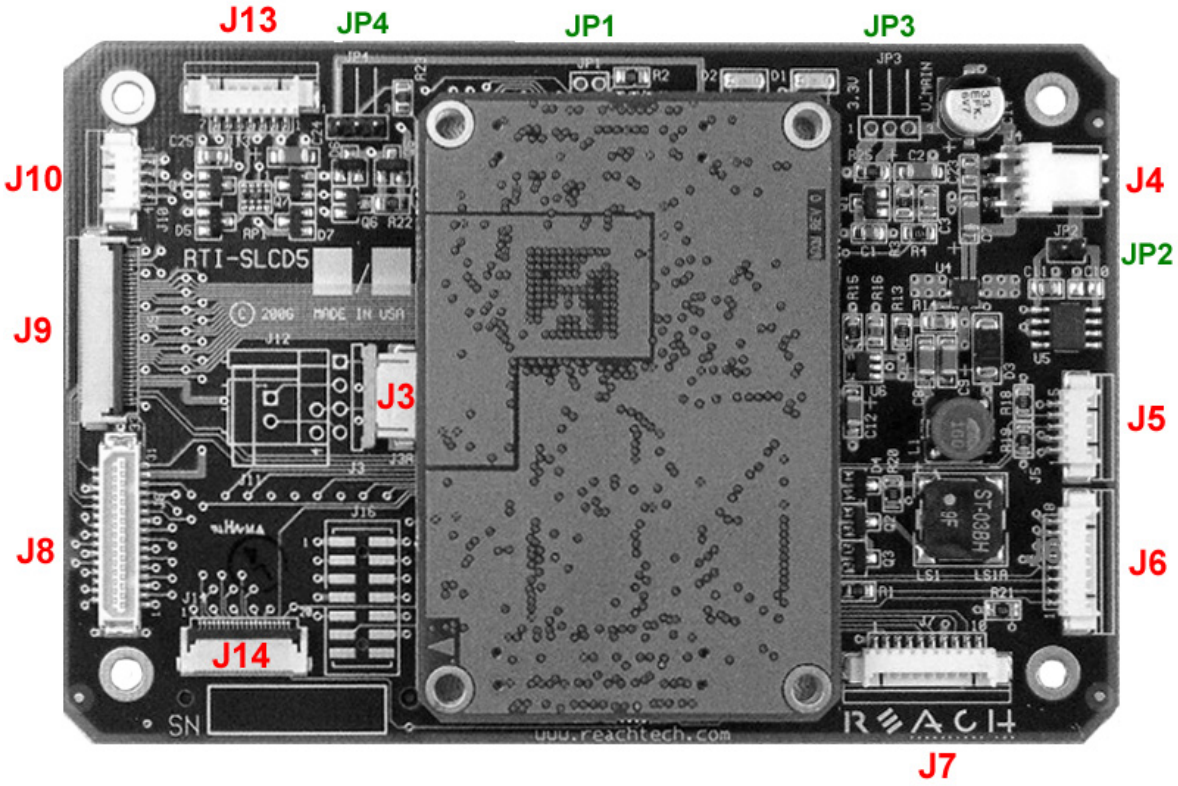


Figure 2: Connectors and Jumpers (Rev A shown; Rev B does not have JP2)

2.1. *Overview*

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

Purpose	Use	Notes
POWER	J6	One cable for both communications and power.
	- or -	
	J4	Power cable is to separate from communications cable OR if inverter takes more than 1A current
COMMUNICATIONS		
	J6	COM0 port (typically RS-232)
	J7	COM1 port (RS-232 or CMOS 3.3V)
PANEL	J8	Most VGA panels
	J9	Most QVGA panels
INVERTER	J13	High power for VGA panel inverters
	J10	Compatible with SLCD5 inverter cabling for QVGA panels

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(Typically. 12V) input, max. 1A *(Note 1, 2)*
6	Ground
7	5 - 12V main power Input *(Note 1, 3)*
8	Ground

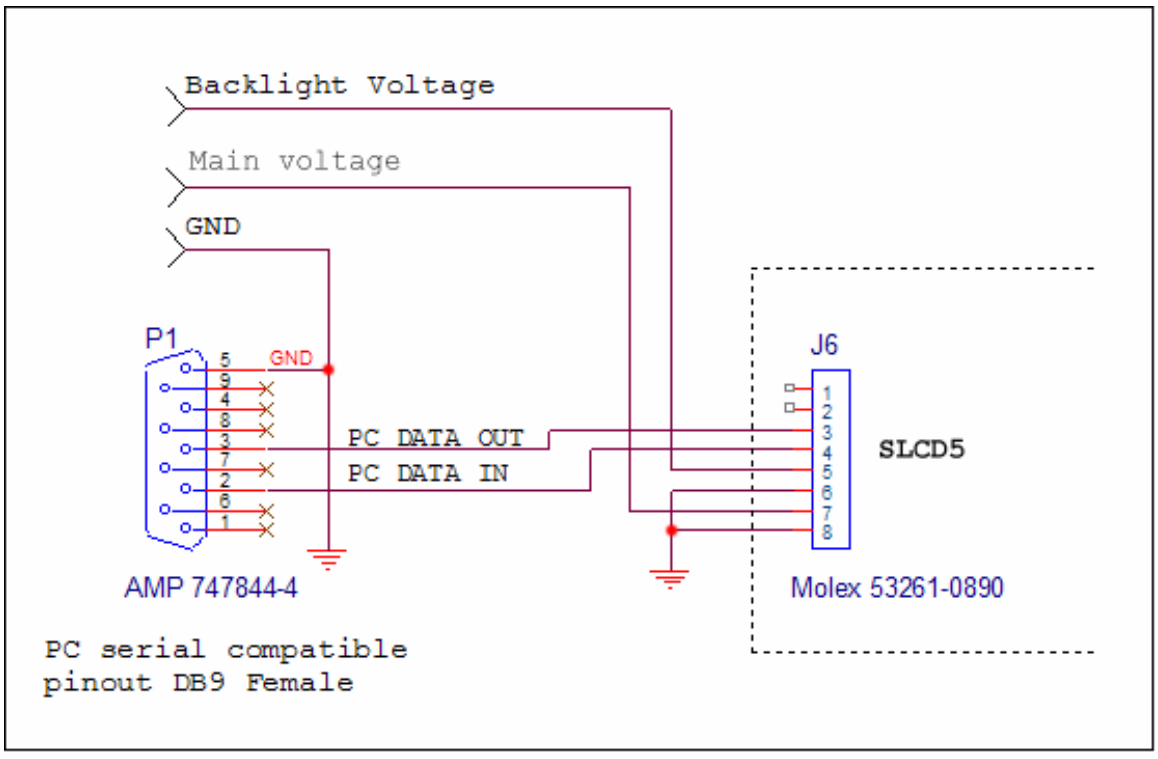
Note 1: The board can be powered either through J6 OR J4. If it is powered through J4, pins J6-5 and J6-7 do not need to be connected.

Note 2: Rev B board: if pin 5 is powered with 5V for a 5V backlight driver / inverter, resistor R8(zero ohm) 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.

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

Typical connection from a PC is as follows:



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.

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)*
6	Ground
7	5 - 12V main power Input *(Note 1, 3)*
8	Ground

2.4. *J7 - Communication COM1 and Reset*

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.

J7 10 Pin Molex 53261-1090 for RS232 COM1 Communications

Pin	*Connections for RS232 Mode*
1	3.4V output (for powering external transciever)
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)
9	Connect to pin 4
10	not used

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

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 transciever)
2	*RTS (pulled low, high is tristate)
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 (active low)
9	do not use
10	*TxD- (standard UART transmit data output, pulled low, high is tristate)

** These signals require external pull-up resistors to the external logic 3.3V rail.*

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 **9**	n/a n/a	Connect J7-4, J7-9 together at J7

2.5. *J4 - Power (if separate power input is desired)*

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)
2	Main power, connects directly to J6-7. Typically 5V or 12V (Note 1)
3	Ground (common to main and inverter power)

Note 1: Notes for J6 power levels should be applied

2.6. *J3 - 4 Wire Touch (presence is hardware version dependent)*

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

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

All signals are ESD protected via California Micro Devices PACDN044Y5.
For Hitachi QVGA touch, Fujitsu touch

2.7. *J11 - 4 Wire Touch (presence is 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 (presence is 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

For 3M (new) touch

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 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

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
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 in the firmware. The sense and range of the brightness voltage output is also set in the firmware.

2.13. *J5 - External I2C (not typically used)*

J4 5 Pin Molex 53261-0571 or equivalent

Pin	*Signal*
1	V3.15
2	Ground
3	SCL (serial clock)
4	SDA (serial data)
5	I2C IRQ input (pull down - pulled up to pin 1 on board via 4.7K)

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 SCDAA1A0900

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	SD_DETECT
11	GND
12	SD_WP
13	not connected
14	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 10mA at 3V (must provide external series resistor)

2.16. *JP2 - special power*

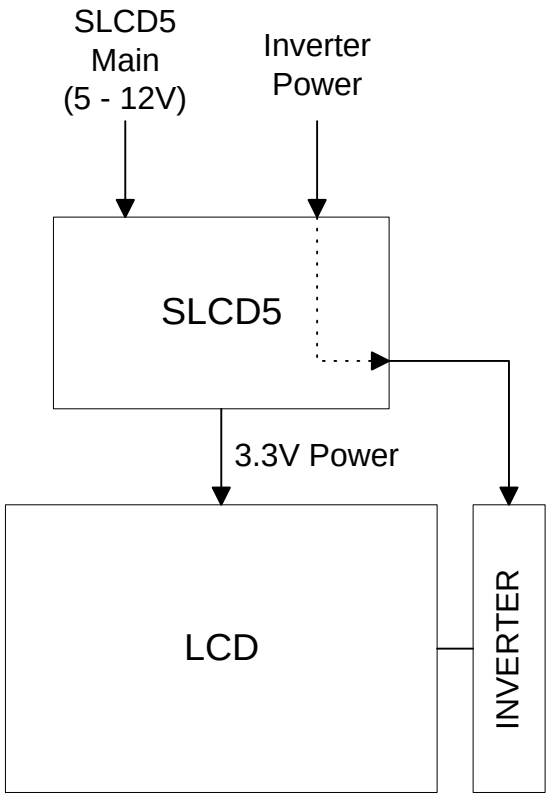
The SLCD5 is typically powered by 12VDC, and generates 5V for its on-board speaker and inverter interface via a linear regulator. If the board is powered with 5V (JP6-7 or J4-2) the linear regulator needs to be bypassed for maximum beeper volume and inverter control. Insert JP2 in this case.

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.



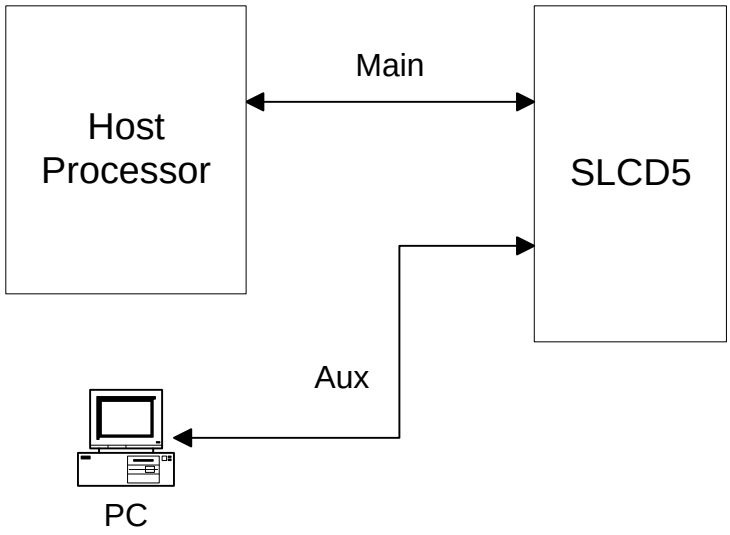
3.2. *Serial*

The SLCD5can use either RS-232 levels or CMOS logic levels for serial communication. As shipped, 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.

The COM0 and COM1 ports can be mapped to "Main" and "Aux" ports as needed. The Main port is connected to the embedded processor and controls the display. The Aux port is typically used to update the display bitmaps and macros.

By default, serial communications 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).

The following diagram illustrates the communications setup. By default, the Main port is COM0 and the Aux is COM2. The Aux port typically is connected to a PC as needed for bitmap loading. In production, either port can be used for loading.



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.

3.4. *System Configuration*

The SLCD5 maintains 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 more convenient 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. If the bitmaps and macros fit in the on-board serial flash, then the SD card is only needed for initial power-on load and configuration setting.

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.

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.

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	Equivalent to *orient command.
invEnaHiTrue	0 or 1	Same as *invEnaHiTrue command.
pwmIsEnable	0 or 1	Same as *pwmIsEnable command
maxBrite	0..255	Same as *maxBrite command
minBrite	0..255	Same as *minBrite command
brite	0..255	Same as xbb command
touchSwap	0 or 1	Same as *touchSwap command
tsamp	0-20	Default value is panel-dependent See *touchParm command.
tspan	0-20	Default value is panel-dependent See *touchParm command.
touchMode	0*..7	bit 1 => lockout bit 2 => validate touch pressure bit 3 => touch wander acts as release See *touchMode command.
tplo	0-65535	Default value is panel-dependent See *touchMode command.
tphi	0-65535	Default value is panel-dependent See *touchMode command.
tcTimeout	0-99	same as *tcTimeout command
auxEsc		Same as *auxEsc command.
beepFreq	0..3000	Same as bf command
beepVol	0..255	Same as bv command
debounce	50 - 200	Same as *debounce command
typematicDelay		Same as first argument of typematic command
typematicRepeat		Same as second argument of typematic command

CONFIG.INI options: (* indicates default)

PONMAC		Same as *PONMAC command
SPL		Same as *SPL command
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 clocks
vpw		Set LCD vertical sync width timing in clocks
vbp		Set LCD vertical "back porch" timing in clocks
downloadToSD	0 or 1	If 1, serial download of the bitmap/macro file will be stored on the SD card. Note that the SD must be specially prepared for this option.
config	"text string"	Sets response for *config command
LCDshiftclock	0 or 1	0 = Active on positive edge of LCD Shift Clock. 1 = Active on negative edge of LCD Shift Clock.
baud0 baud1	1 - 8	1 = 1200 baud 2 = 4800 baud, 3 = 9600 baud 4 = 19200 baud 5 = 38400 baud 6 = 57600 baud 7 = 115200 baud 8 = 230400 baud
externalSpeaker	0* or 1	0 = External Speaker OFF, Beeper ON 1 = External Speaker ON, Beeper OFF

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.

A firmware version is available with an embedded interpreter for standalone type applications where the SLCD5 acts as both interface and host. Contact Reach sales for more information.

4.2. *Compatibility with SLCD controller*

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

- The bitmaps must be in 24 bit format, not 8 bit palletized. If an application is moved from the SLCD to the SLCD5, all bitmaps must be converted to the new format.

- The fonts have 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:

  OLD: `s<index><state>`
  e.g. `s1281`

  NEW: `s<index> <state>`
  e.g. `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 developed on the PC as bitmaps (.bmp files) in 24 bit color. These are then either downloaded into the SLCD5 data flash memory, or stored on an SD card that is then inserted into the SLCD5. A macro file is a set of commands that can be invoked with a single command, and are detailed in Appendix E.

In a development environment, the SD card is a convenient way to change images and stored macros. Once development is complete, the SD card can also be used to update the data flash memory. Once updated, an SD card is no longer needed.

Note: Appendix D describes bitmaps and the BMPload program used to store these either into the SLCD5 data flash or to an SD card. Macros are a sequence of SLCD5 commands and are described in Appendix E.

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 the communications port looped back transmit to receive. This loopback is via a jumper on the "PowerCom4" board in the case of the unenclosed kit. The demo macro is #1. **Section E.6 contains a listing of the pre-loaded macro file.**

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. Just plug the supplied 12VDC power supply into the barrel connector on the PowerCom 4 board. The display should light up and lead you through various touch-activated screens.

Note that the demo is preloaded on the kit, 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 of the SLCD5 manual provides a short 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. ***In order to communicate over the serial port, the Demo jumper JP1 must be removed.*** This jumper loops back transmit to receive on the serial port and this is what tells the SLCD5 to run the demo.

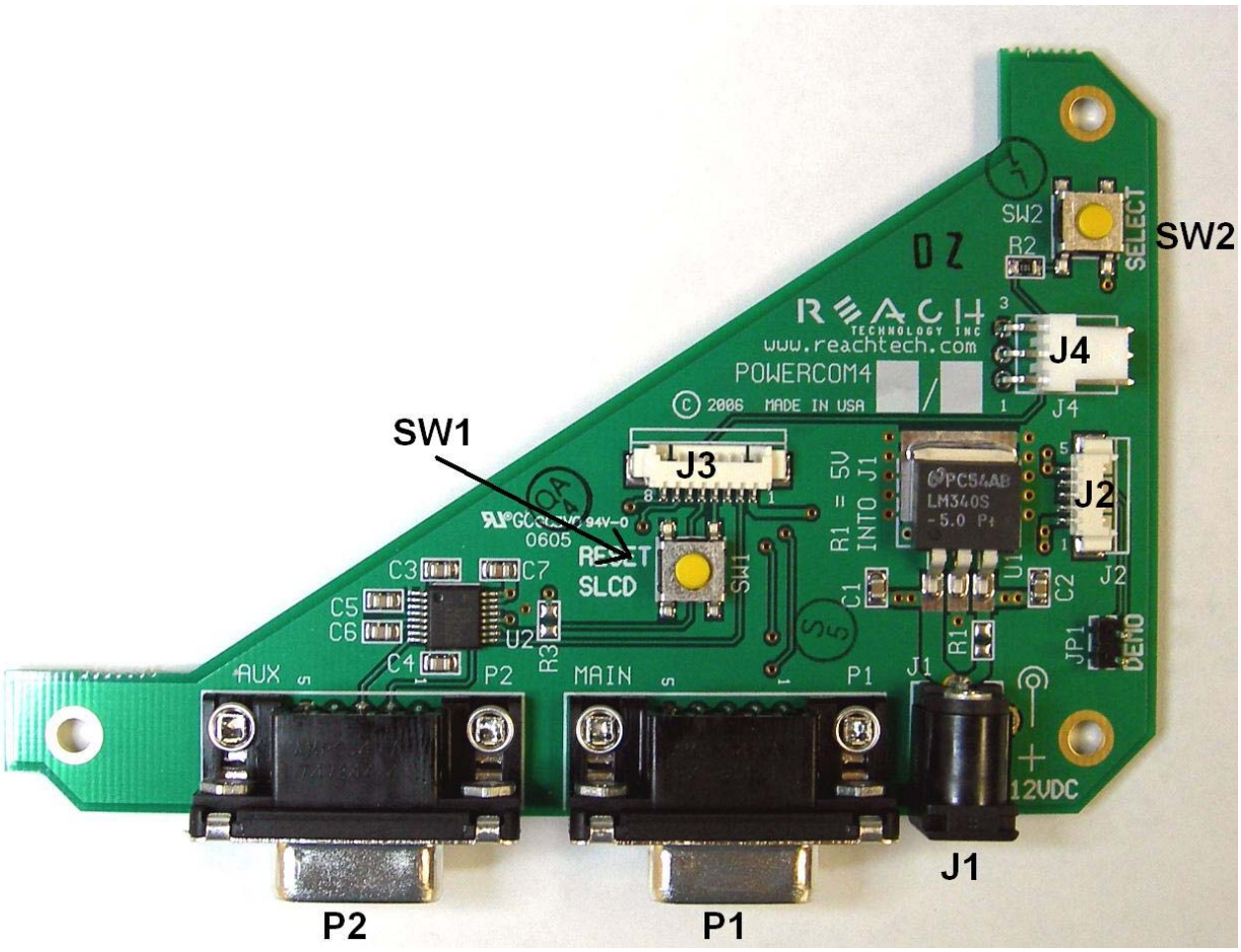
As shipped, the serial port is set to 115,200 baud, 8 bit, 1 start, 1 stop, no parity. There are two DB9 connectors on the "PowerCom 4" board. Connect the PC using a straight through cable to the DB9 marked "MAIN" (P1). A USB-to-serial adapter cable can also be used and plugged directly into this connector. ***Note: the Belkin USB-serial adapter has software compatibility issues and is not recommended.***

Once connected, use HyperTerminal or similar terminal emulator to send and receive commands from the kit. **Appendix G of the SLCD5 manual provides a short tutorial.**

HyperTerminal has limitations that can cause problems; specifically it cannot send the "escape" character. We recommend ProComm Plus from Symantec, or RealTerm (shareware). ProComm has the advantage of being able to run scripts which can simulate the user interface on the SLCD5 using a PC.

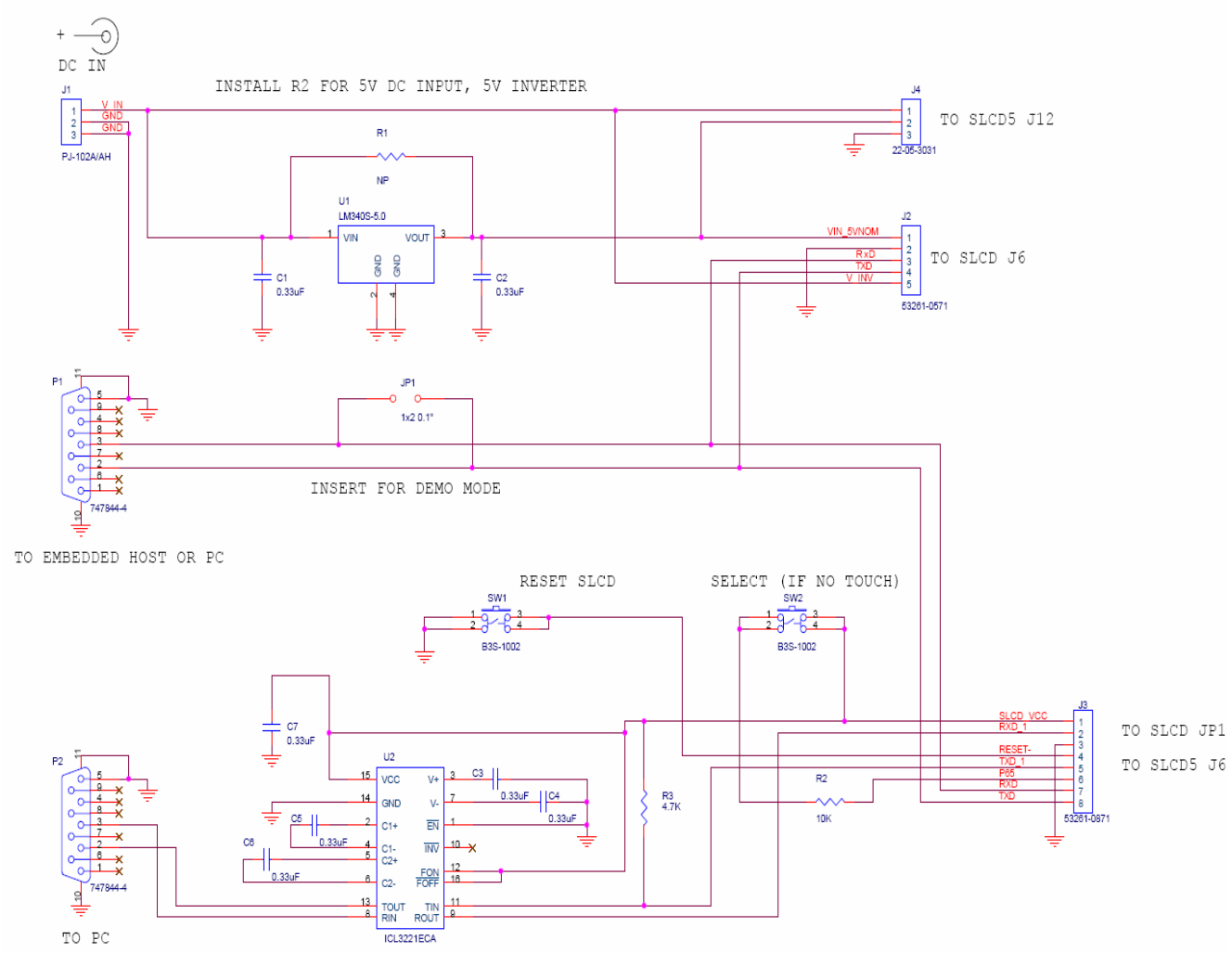
4.7. *Connecting the kit to an embedded controller*

The main purpose of the kit is to provide an embedded controller with a Graphical User Interface (GUI). The controller can be connected to the kit using the “MAIN” (P1) DB9 port. The second "AUX” (P2) DB9 port is provided so that a PC can download new bitmaps and macros without having to disconnect the embedded system. How it works is this: when the aux port receives three <return> characters in a row, it switches to become the main port. This way, the BMPload program can take control and download new bitmaps. Use the reset button to simulate a power-on event which restores the main port as the default control port, or use the *prevCons command.



PowerCom4 Board

4.8. *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 "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. These are loaded on the SD card that comes with the kit. If the 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 is the “DEMO” serial loopback jumper that is installed at the factory in order to automatically run the demo at power up. Remove the jumper prior to attempting serial communications with SLCD5 controller.

7. The J1 barrel connector is the 12VDC external power supply connector for the development kit. It is 2.1mm, center pin positive.

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 POWER-ON MACRO feature.
- 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=639, y=479 (VGA landscape mode).
- 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 ¾ full, an XOFF is

issued to the host. An XON is then issued when the buffer is approximately ¼ 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 the 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 Software 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:

```
Boot Loader begins execution
Hardware Component Initialization
Display ":SLCD5 Bootloader V#/#"...
Read SD Card file contents
Read Last saved Firmware File name in EEPROM
IF < SD Card Firmware File name != EEPROM Firmware File name> THEN
   BEGIN Programming Flash
            Display "Found" New Firmware File
                  Display "Erasing" Flash Memory
                  Display "Programming" Flash Memory
   END Programming Flash memory
ENDIF
Jump to Firmware code
Boot Loader completed
Firmware begins execution
Display “Starting firmware”
Read EEPROM
IF < EEPROM Valid > THEN
   Set Default Option Values
ENDIF
IF <SD Card Present> THEN
         IF < CONFIG.INI present> THEN          //System Configuration
            DO
                  Parse Option Keyword
                  Set Option system variable
                  IF < Verbose Mode == ON>
                        Display (Option Keyword and Value)
```

```
                    ENDIF
            UNTIL <End of File>
   ENDIF
   IF <SPLASH.BMP Present> THEN                    //Splash Screen
            Display (SPLASH.BMP)
   END
   IF <SLCD.BIN present> THEN                      //BMPs and Macros
            IF <FLASH1.INI present> THEN
                    Copy SLCD.BIN to Data Flash
                    Use Data Flash Memory for BMPs/Macros
            ELSE
                    Use SLCD.BIN for BMPs/Macros
            ENDIF
   ELSE
            IF < Data Flash Memory Valid> THEN
                    Use Data Flash Memory for BMPs/Macros
            ELSE
                    Display (Error:  Data Flash Memory unusable)
            ENDIF
   ENDIF
   IF <RUNDEMO.INI present> THEN                   //Demo macro (used at power-on)
            IF <"demomac = <macro number>"> THEN
                    Set Demo Macro Number to <macro number>
                    Run <macro number>
                ENDIF
   ENDIF
ELSE //SD Card NOT Present
   Use Data Flash Memory for BMPs/Macros
ENDIF

While <TRUE>          //Application-User Processing
   Process Touch Panel, Serial Port, Macros
END
Firmware completed
```

Figure 3: System Software Boot Algorithm

5. Software Command Reference

Notes:

COMPRESSED SYNTAX

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

```
p <pixels><return>
```

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.

COMMAND SYMANTICS

All commands end with a <return> character, which is 0x0D, decimal 13. All return information ends 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 requred.

Square brackets with a vertical spacer '|' indicates optional arguments.

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.

TOUCH PRIORITY

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

SET PEN WIDTH

Description	Sets the pen width for line drawing commands including line, rectangle *but not circle*. Default is width of 2.
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.
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.
Command:	sc `x y`
Example:	sc 10 20<return> t "hello"
	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 test display command. *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 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. Note that the SET CURSOR command location is relative to this global origin. Note: 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
0	Black		10*	Light Grey
1	White		11*	Light Blue
2*	Blue		12*	Light Green
3*	Green		13*	Light Cyan
4*	Cyan		14*	Light Red
5*	Red		15*	Light Magenta
6*	Magenta		16*	Yellow
7*	Dark Brown			
8*	Dark Grey			
9*	Grey			

* Only valid for color display

Doing:

Example: `s 0 1`

Form 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 RBG values.
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.

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

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.

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 previously 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 SLCD5 data flash memory. See Appendix D and I for details.

Command: `xi <number> x y`

Arguments: <number> is bitmap number as listed in the "ls" command.

Example `xi 4 10 20`

This displays the 4th memory record 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 line 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.***

The mode (N, R, T, X, TR) most recently specified is used until another mode is specified. Initially, Normal mode is used, and will remain in effect until changed. To prevent confusion about which mode is in effect, always explicitly declare mode.

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 WW
```

displays the text

```
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.

TEXT FLASHING DISPLAY

Description: Displays a flashing text string starting at the current or specified point using the currently set font. The string is alternately drawn in the selected foreground color, then erased with 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|T|X|TR]`

All parameters except index and "text string" are optional. If unspecified, <t> will default to 500 mS. A null text string "" is acceptable.

Arguments: <index> is an identifier for this text; accepted identifiers are 0 through 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: foreground / background colors are reversed.

T – Transparent: text written on top of current display with no "background box".

X – XOR

TR – Transparent reversed

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

The only required parameters are the <index> and the "text string". The timing defaults to 500 milliseconds, the text is written to the current cursor position, and the mode will be set to transparent.

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

This puts the text

```
FLASHING TEXT
```

With the top left corner of the 'P' 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.

TEXT FLASHING DISABLE

Description: Disables flashing text instances 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 text; accepted identifiers are 0 through 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 or animation index, use the “tfd <index> <state>” command to stop the animation at the selected state, and then use the “anix <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 text; accepted identifiers are 0 through 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 currently stopped text flash animation
Command:	`tfe <index>`
Arguments:	<index> is the identifier for the text; accepted identifiers are 0 through 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 has its own memory for state save/restore.
Command:	`ss`
Arguments:	None
Examples:	`ss`
	Save the drawing state.

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.
	Note: each Macro has its own memory for state save/restore.
Command:	sr
Arguments:	None
Examples:	sr
	Restore the drawing state.

PIXEL READ / WRITE

Description: The "pw" commands write a pixel and the "pr" commands read a pixel at location (x,y).

Commands:
```
pw x y [565 color value]
pwx x y [RRGGBB 6 character color value]
```

Arguments: if no color value is given, the pixel is written in the foreground color.
565 color = RRRRRGGGGGGBBBBB (binary)

RRGGBB = 24 bit color value as 6 character ASCII string where each color is 00 thru FF

Example: `pw 10 20 F801`

This sets the pixel at (10,20) to R=11111XXX (8 bit), G = 0, B = 00001XX.

Command: `pr x y`

Returns: 565 color value as 4 ASCII hex digits, e.g. `F801`

Command: `prx x y`

Returns: 24 bit color value as 6 ASCII hex digits, e.g. `FF0008`

Example: `prx 10 20 (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: `l x0 y0 x1 y1`

Example: `l 0 0 639 479`

This will draw a line from the upper left-hand corner of the screen to the lower right hand corner

DRAW RECTANGLE

Description: Draws a rectangle using the foreground color or an arbitrary color

Command: `r x0 y0 x1 y1` [style] [color]

Arguments: Upper left corner is (x0,y0) and lower right corner at (x1,y1).

style: omitted=regular line, 1=filled, 2= one pixel wide dotted line, 3= filled color is RRGGBB or RGB format.

color: fill color in RGB or RRGGBB format (see SET COLOR detailed)

Example: `r 100 100 180 120`

Draws a rectangle positioned at 100,100 with a width of 80 and a height of 20.

`r 100 100 180 120 1`

Draws a rectangle filled with the foreground color positioned at 100,100 with a width of 80 and a height of 20.

`r 50 100 180 120 1 C03`

Draws a rectangle filled with the color R=C,G=0,B=3 positioned at 50,100 with a width of 80 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 a 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

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.
- 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 bitmap6 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 is 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.

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 height of 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 height of 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 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: `xxy <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: `xxynb <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 TOCUH "xc all" command

DISABLE TOUCH

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

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.

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 zero. 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 touch action is continuous and the slider values range from 0 at the bottom to 100 at the top.

Example notification: `l128:50`

CIRCULAR SLIDER DEFINE

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

Command: `hotspot, background, x, y, slider, type, top, bottom, value, scope, maxAngle, minAngle, startDeg, drawRadius, maxRange`

Arguments:

`hotspot` - 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.

`background` – background bitmap index

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

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

`type` – 1 =

`top` – maximum slider value

`bottom` – minimum slider value

`value` – value between top/bottom to place slider knob

`scope` – number of value points to change per revolution. This parameter is only valid for type 2 circular sliders. Any value given for type 1 circular sliders will be ignored. The scope must be at least 1 and no larger than the difference between top and bottom.

`maxAngle` – maximum angle (0-360) from positive x-axis in degrees to place slider knob for type 1 sliders. The maximum value will be tied to this angle. Any value given for type 2 circular sliders will be ignored

`minAngle` – minimum angle (0-360) from positive x-axis in degrees to place slider knob for type 1 sliders. The minimum value will be tied to this angle. Any value given for type 2 circular sliders will be ignored

`startDeg` – angle (0-360) from positive x-axis in degrees to place slider knob/ to calculate out of bounds. This parameter is only valid for type 2 circular sliders. Any value given for type 1 circular sliders will be ignored

`drawRadius` – radius from center of the background image to draw the rotator/knob image. The draw radius must be small enough so the rotator/knob will never be drawn outside of the background image.

`maxRange` – maximum range in pixels from the center of the rotator/knob slider that a touch will take effect.

NOTE: For type 1 sliders there will be a dead zone between the min/max angle. Type 2 sliders have accessible dead zones when out of range of the min/max values but the value will not go beyond the min/max.

NOTE: All circular sliders are clockwise increasing. For example, if you have a type 1 slider with min at 0 and the max at 360 the rotator/knob will not move because there are 0 degrees for the rotator to move. Simply switch the min/max values to get the full 360 degrees.

Host notification when slider value is changed:

`l<idx>:<value>`

Example: `slc 128 46 100 30 47 1 500 0 250 0 240 300 0 25 120`

This example assumes that the demo bitmaps are loaded with 46 and 47 being the slider background and control respectively. A type 1 slider is created in the middle of the screen (left corner = 100, 30) with possible values between 0 and 500 with its initial value at 250. The minimum value will be tied to 240 degrees and the max at 300 degrees from the positive x-axis. This will make the initial value of 250 (half of the max) to be placed midway between min/max (90

degrees). The rotator/knob will be drawn at 25 pixels from the center of the image with a valid touch being at most 120 pixels away.
Example notification: `l128:50`

`slc 128 46 100 30 47 2 4000 1 2000 1000 0 0 90 25 120`
This example assumes that the demo bitmaps are loaded with 46 and 47 being the slider background and control respectively. A type 2 slider is created in the middle of the screen (left corner = 100, 30) with possible values between 1 and 4000 with its initial value at 1000. This will make the initial value of 2000 tied to 90 degrees form the positive x-axis. The value will change by 1000 every rotation until the min/max has been reached. The rotator/knob will be drawn at 25 pixels from the center of the image with a valid touch being at most 120 pixels away.
Example notification: `l128:50`

SLIDER VALUE

Description:	Sets the value of a previously defined slider object.
Command:	s`v idx val`
Arguments:	`idx` - slider index `val` - value for the slider.
Example:	`sv 128 50` Sets slider index 128 to value 50.

CIRCULAR SLIDER VALUE

Description:	Sets the value of a previously defined circular slider object.
Command:	s`vc idx val`
Arguments:	`idx` – circular slider index `val` - value for the slider.
Example:	`svc 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

`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 DEFINE

Description: Defines 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, the data starts again on the left in "Oscilloscope" style.

Command: `cd n x0 y0 x1 y1 t dw bv tv bc <pens>`

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

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

`t` - chart type; must be 1

`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)

`bv` - bottom data value (lowest y value)

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

`<pens>` - one or more sets of two values: pen width and pen color. Width is 1 or 2, color is same format as "bc" parameter.

Example: `cd 0 10 20 110 120 1 4 0 99 333 2 0FF 1 F00`

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).

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.

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
```

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. Next two more data points are added.

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 at 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, and maximum size of each argument is version-dependent. See Appendix E.*
Command:	`m <n> [macro parameters . . ]`
Arguments:	<n> is the macro number between 1 and 255. 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 3 "" 2` This causes macro #3 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>:<label> [macro parameters . . ]`
Arguments:	Same as above, with macro label. See Appendix E for a full description.
Example:	`m 3:subfunction` arg1 arg2
Note:	The maximum number of arguments is 10, and maximum size of each argument is 8 characters.

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> 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.

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"

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.

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).
- `1` - 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>` arguments for the macro. These are delimited by spaces. Double quotes can be used to surround the 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 (cont'd)

Example 2: bd `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.

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

Arguments:

`<n>`	The index of the animation, 0 through 9
<text string>	Any valid command *Except* animation or flashing text.

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.

ani? 0

Returns:

```
ani 0 f 13B
ani 0 S 000 fff
ani 0 ta LT
ani 0 o 0 0
ani 0 sc 0 0
ani 0 t "Hello world"
ani 0 y 500
ani 0 f 13B
ani 0 S fff 000
ani 0 ta LT
ani 0 o 0 0
ani 0 sc 0 0
ani 0 tm T
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>] [<line2>]`

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.

`<line2>` Optional value added to first argument for total line offset. Useful when displaying a bitmap that starts at line and is line2 high.

Example: `wvr 100 35`

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: Similar to WAIT VERTICAL RETRACE, however works in any screen orientation. Returns when a vertical (landscape orientation) or horizontal (portrait orientation) display retrace occurs. This command is used to avoid "tearing" or “flashing” in animations.

Command: `wrf <x> <y>`

Arguments:	<x>	Utilized in portrait orientation mode. This is the x coordinate to wait for refresh. The number of vertical scan lines to wait is the X screen resolution minus x-line value. Note that the vertical scan lines start from the right to left of screen when viewing in portrait orientation.
	<y>	Utilized in landscape mode. This is the y coordinate. This is also the number of horizontal scan lines to wait after retrace. Note that the horizontal scan lines start from the top to bottom of screen when viewing in landscape orientation.

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

Example: `wrf 100 35`

This example is in portrait orientation mode. This waits until vertical scan lines refresh the screen until reaching x coordinate 100. For a QVGA LCD screen (320x240 pixels), this would wait 220 vertical scan lines (320-100 = 220). Note this is the case since scan lines are updated from right to left when viewing in portrait orientation.

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 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>`

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

Description: Turns the external speaker on/off and stores the setting in non-volatile memory 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.

First beep after power on seems to be ignored.

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

Description: Sets the volume level of the beeper. The value is stored in non-volatile memory and restored on power-on.

Command: `bv [+|-]`<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.

BEEP FREQUENCY

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

Description: Sets the frequency of the beeper. The value is stored in non-volatile memory and restored on power-on. 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 [<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 `bf 2500`

Sets the beep frequency to 2500 Hertz

`bf`

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. The value is stored in non-volatile memory and restored on power-on. The "xbbs" version of the command does NOT store in EEPROM for fast operation, e.g. when slowly dimming.
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

Description	Sets a new baud rate of single port PowerCom board. 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.
Command:	`baud [230400|115200|57600|38400|19200|9600]`
Argument:	baud rate
Example:	`baud 57600`
Note:	The baud rate is non-volatile (recalled) upon reboot.

SET BAUD RATE OF COM0 AND COM1 PORT

Description	Sets a new baud rate for port specified. 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.

Command:	`baud0 [230400|115200|57600|38400|19200|9600]`
	`Or`
	`baud1 [230400|115200|57600|38400|19200|9600]`

Argument:	baud rate
Example:	`baud0 57600`
Example:	`baud1 19200`
Note:	The baud rate is non-volatile (recalled) upon reboot.

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.
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, QVGA). Panel string - Contact REACH Technology with Panel String to determine display panel information
Example:	`SLCD5 V1.1.20`

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 320 or 640 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 239 (QVGA) or 0 to 479 (VGA).

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<n><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<n><return> where H is a single hex digit.

LIST BITMAPS

Description	Returns a summary of the bitmaps 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.
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 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 FILES FROM SD CARD

Description: Causes the bitmap file "SLCD????.BIN", “SPLASH.BIN” to be loaded from the SD card into memory. For a complete description refer to the Firmware portion of “Figure 3: System Software Boot Algorithm”. Note that file "SLCD????.BIN" is generated by the BMPload.exe program and contains both bitmaps and macros.

Command: `*LOAD`

LOAD BITMAP FILE FROM SD CARD 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, so the directory name is limited to 8 characters.

Example: `sdload \Images`

The file "SLCD????.BIN" (if present) is loaded into memory from the directory “Images” from the root (‘\’) of the SD Card.

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 up to 1,000.

Command: `*getScreenAsBMPFile`

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

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". This takes the place of the initial display version text string.

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 5th memory record at location (0, 0) on power-on reset.

POWER-ON MACRO

Description: Used to define a macro that is executed when the unit is first powered on. This can be used to set the desired baud rate if the default of 115,200 is too fast.

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`

DEMO MACRO

Description: Used to define a macro that is executed during power-on, when the parameter "demomac = 1" appears in the rundemo.ini file.

Command: `*DEMOMAC <index | name>`

Arguments: (none) = display the current demo macro index (or name), or 0 for none.

`<index>` = 0 or 255 disables the demo macro feature

`<index>` = 1 through 254 sets the demo macro to the specified macro.

Example: `*DEMOMAC 1`

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:

Standard (ASCII) notification	Binary notification
`x<index><return>`	`X<binary index>`
`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`
	`<binary state> 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 Autoswitch

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 EEPROM and will be used on power-up. Note that this switches between the MAIN and AUX ports of the PowerCom4 board.

Command: `*com0main (Sets and switches to MAIN Com Port)`

`*com1main (Sets and switches to AUX Com Port)`

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. These are stored in non-volatile memory.

Command: `typematic <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: `typematic 200 50`

This sets the delay to 2 seconds and the repeat rate at 500ms = 2 per second.

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

SET TOUCH DEBOUNCE

Description: Sets the delay between touch button responses. This is stored in non-volatile memory. 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 commands allows these signals to be set.

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 ploarity.

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.

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.

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.

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 * maximum value.

DEFINE TOUCH ACTION

Description: The touch screen can have different operating characteristics defined using this command.

Command: `*touchMode [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: `*touchMode 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.

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.
	-	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. If no argument is used, the current value is displayed.

Command: `*tcTimeout [<timeout in seconds>]`

Example: `*tcTimeout 5`

This sets the timout to 5 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.

Command: `*auxEsc <hex value of ASCII character>`

Arguments: (none) - display current escape character
`0x01 though 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.

EEPROM READ, WRITE

Description:	The EEPROM can be read and written to. ***CAUTION: DO NOT OVERWRITE THE CONFIGURATION DATA.***
Commands:	`*eer <hex address byte>` `*eew <hex address byte> <hex value byte>`
Arguments:	all arguments are two ASCII characters from 00 to FF. `<hex address byte>` must be greater than 0xC0 to avoid configuration data area. This gives the user 64 data bytes from C0 to FF.
Examples:	`*eew c1 a5` `*eer c1`
Returns:	`c1 = a5`

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: "<font_alias> <filename>.sif <CR>"
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 polynomical 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, 0x01,
0xC0, 0x80, 0x41, 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, 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, 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, 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, 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, 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, 0x01, 0xC0, 0x80, 0x41,
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: "~HHHHSLCD 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]
```

Appendix A - LCD and Touch Panels compatible with the SLCD5 controller

The SLCD5 controller has been tested with the following LCD and touch panels:

A.1 *8.4" LCD NEC NL6448BC26-01, NEC NL6448BC26-09*

Color TFT 8.4", 3.3V operation

Use Axon or Quadrangle DF9-31 to DF9-31 data cable

Use ERG K2677 inverter

A.2 *8.4" Touch Panel*

Fujitsu N010-0554-T511 (RoHS, 1.1mm glass), 4 wire resistive

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

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	e.g.	P/N FDC31/254AFF03
		DF9-31 to DF9-31 cable 10" female - female same side 1-1 pinout
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

Appendix C - Ordering information

C.1 General information

The Reach part number for the SLCD5 is 42-0087. When ordering, you must 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 software that comes pre-loaded on the board.

Example: 42-0087 Rev 01/01
01 = Hardware configured for Fujitsu touch panel
01 = Software version for NEC NL6448BC26-01 panel, Fujitsu touch, K3034 inverter.

C.2 Contact Reach directly for specific ordering information.

Reach Technology Inc
4575 Cushing Parkway
Fremont, California 94538
(510) 770-1417
or
(503) 675-6464

Appendix D - BMPload program

D.1 *Overview*

NOTE that for Version 1.1.29 and higher, the bitmap storage format has changed. BMPload Version 2.1.7 and above must be used or the R and B colors will appear swapped.

The SLCD5 contains 4 Megabytes of on-board data flash memory for storing bitmaps and macros. This is normally loaded via the SD card during production setup. Once loaded, the data is non-volatile; that is, it is permanently stored even if power is turned off. The SD card can also be used; ***see Appendix I for more details about using the SD card.***

Stored bitmaps are displayed on the screen using the "xi" command (See **DISPLAY BITMAP IMAGE** Command) and by the define button and other commands. The BMPload.exe program is used to create a compressed image containing the bitmaps and macro file. This image can then be written to the SD card or downloaded into the SLCD5 via the serial port.

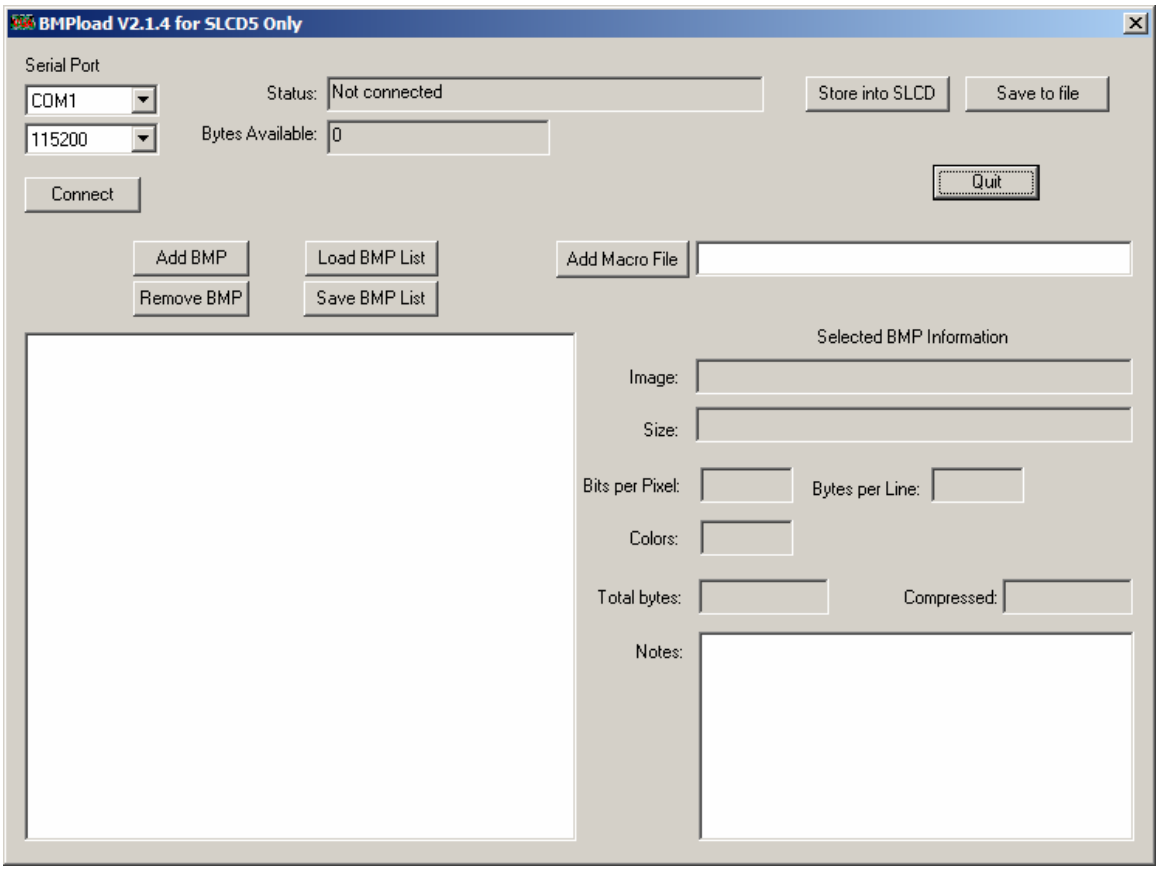
D.2 *Bitmap Format*

The BMPload program requires bitmaps to be in 24 bit RGB color. The SLCD5 uses 16 bit color in 565 format; the BMPload program does the required translation.

BMPload also compresses images using RLE16 (run length encoding). This is very efficient for control surfaces that have repeated pixels of the same color.

D.3 *Program Operation*

BMPload runs under Windows 98 through XP. 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:

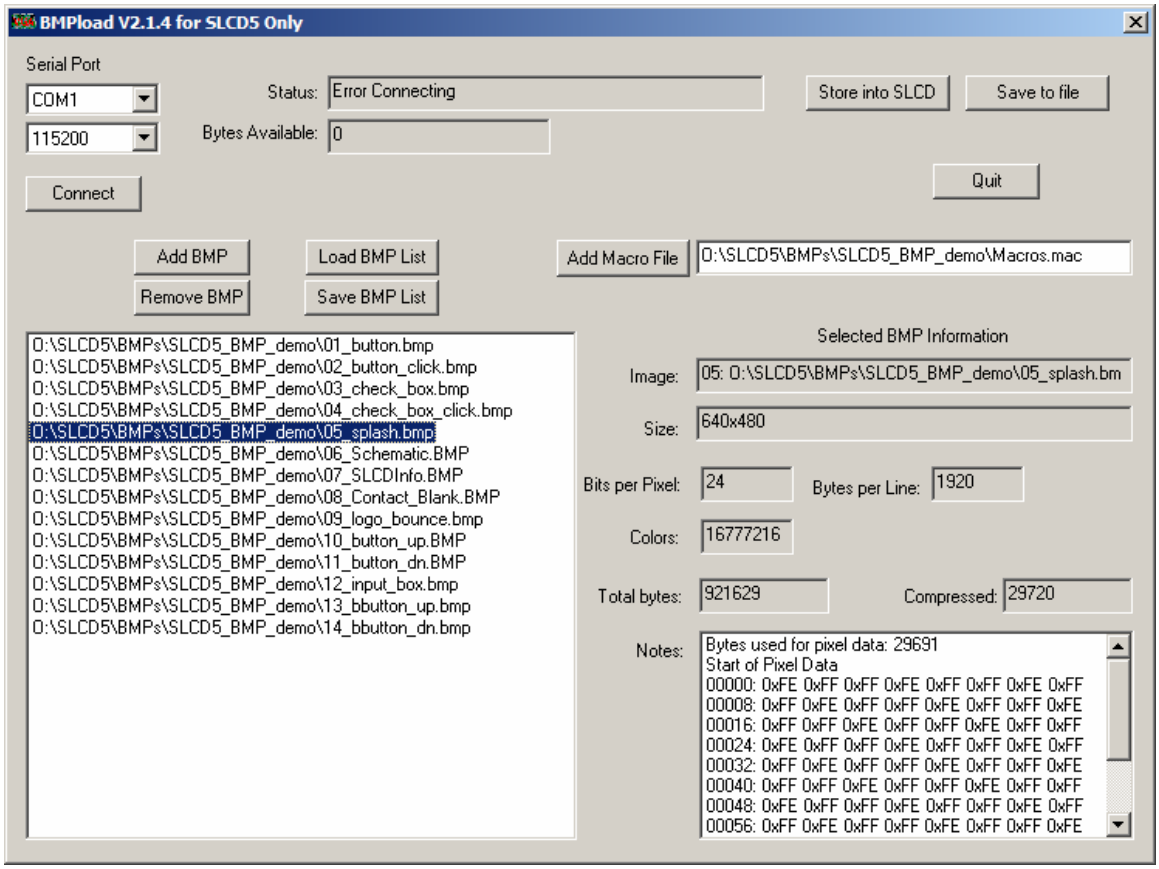


You can now use the "Add BMP" button to add BMP files to the list. Note that the order is important because you use the index number in the DISPLAY BITMAP IMAGE command. The best way to keep this clear is to start all bmp file names with their index number, for example "01_first_bitmap.bmp".

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_Schematic.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 memory image as well as the bitmaps.

Once you have added the bitmaps, use the "Save to FIle" button to save them to the SD card. The file name must be 8 characters, start with "slcd" and have the extension ".bin".

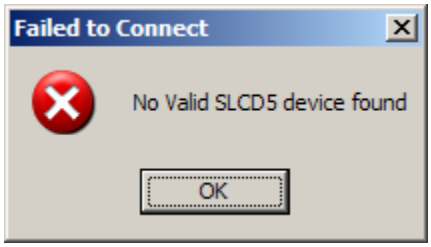
Alternatively, you can use the "Store into SLCD" to write the image into the SLCD5 flash memory via the serial port. This can be slower than using the SD card.

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 HyperTerminal 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.4 *BMPload speed issues*

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 File method.

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 so that by pushing a button, a macro can generate 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 could create a numeric keypad and the parameters would specify where to draw the keypad on the screen. This reduces hard coding of graphical elements and promotes reuse between screens and products.

There are version-dependent limits on the macro commands and their arguments. For firmware version 2.3.0 and above, those limits are:

MAXIMUM CALL DEPTH = 4

A macro can call another macro, but only to a depth of 4.

MAXIMUM ARGUMENTS PER MACRO = 4

MAXIMUM CHARACTERS PER ARGUMENT = 8

MAXIMUM TOTAL STORED ARGUMENTS = 50 (stored via the TOUCH MACRO ASSIGN WITH ARGUMENTS command)

E.2 *Macro File Format*

The macro file is an ASCII text file and can be generated by Windows applications such as Notepad. The file format is designed so that the same macro definition file can be used a) to load the macros into the SLCD5 flash memory, and b) as a 'C' include file in the user's microcontroller program. This way there is only one file to avoid confusion with macro index numbers.

The format for each macro is as follows:

```
#define <text_name> <number>
(one or more command lines)
.
.
#end
```

The <text_name> is an identifier that follows 'C' language conventions, and is included for reference if the macro file is included in a C program. It has no other use.

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 lines starting with the '/' forward slash symbol. All lines outside of a "#define…#end pair are treated as comments. By using 'C' style comments in a creative way, only the #define lines are seen by the C program.

E.3 *Macro Parameters (Arguments)*

Macros can be parameterized by using the special escape sequences `` `0`, `1`, `2`, `` and `` `3` `` in the command lines. These are replaced at execution time by the arguments supplied by the command that invoked the macro. Macro arguments are only available within programmed macros, and are not available in the command line interface.

Parameterized macro example:

```
#define example 1
t "`0`" `1` `2`
#end
```

The following command uses this macro to display the text "Hello" at location x=10, y=20:

```
>m 1 Hello 10 20
```

E.4 *Special macro arguments and commands*

Memory commands

Memory commands were added to implement the keyboard in the demo macro that comes installed with the SLCD5 kits. These allow a character string to be saved and manipulated. See Section E.6 for examples. The character string is accessed as a special macro parameter.

The commands are::

`mpush "<string>"`

This appends the string argument to the memory variable. The maximum stored string length is 80 characters

Example: `mpush "0"`

`mpop <number>`

This removes the <number> of characters specified from the end of the memory variable.

Example: `mpop 1`

Internal Arguments

The macro system recognizes other symbols in the parameter esacpe format – enclosed in back tick marks. These are as follows.

Memory variable

`M`

This is replaced by the string stored by the mpush command. See the demo macro in Section E.6 for examples.

Random number

`R<lo>:<hi>`

This is replaced by a random number in the range <lo> to <hi>; see the demo macro in Section E.6 for examples.

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 (colon in column 1). Here is the format of a label:

:<label name>

Example:

:attach

Labels are invoked by calling the macro in the normal fashion, but including the colon and label name directly after the macro 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

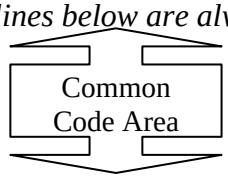
The format of a macro label invocation is: "m <macro number>:<label name>".

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

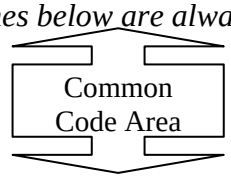
:define *(execution stops here)*
t "Defining button" 200 50
bd 1 0 32 1 "INCREASE" 9 5 10 11

#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 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<return>
```

Now cycle power to the SLCD5, and the initial baud rate will be 9600 baud.

E.6 *Macro Example – (factory loaded into SLCD5 flash)*

The following file is factory loaded into the SLCD5 data flash. It is used to implement the self-running demo that starts if a loopback is detected on the SLCD5 serial port on power-on.

The demo macro is number 1 which runs through a series of screens and shows how to use parameterized macros and buttons that invoke macros with parameters.

Macro 6 is a simpler example that displays a keypad and a single digit when any of the keys are pushed.

```
//
// Reach SLCD5 macro demo file
// 2-1-2005
//
//---------------------------------------------------------------------
//  This macro file assumes the following bitmaps are loaded (in order):
//
// 01_button.bmp
// 02_button_click.bmp
// 03_check_box.bmp
// 04_check_box_click.bmp
// 05_Hitachi_Logos.BMP
// 06_Hitachi_PanelInfo.BMP
// 07_SLCD5Info.BMP
// 08_Contact_Blank.BMP
// 09_logo_bounce.bmp
// 10_button_up.BMP
// 11_button_dn.BMP
// 12_input_box.bmp
//
//---------------------------------------------------------------------

/* comments can be used in either 'C' style or C++ */
// blank lines are allowed
// We comment out the contents of the macro so this complete file
// can be used as a C include file

//---------------------------------------------------------------------

//---------------------------------------------------------------------
// MACRO #1
// This is the macro that invokes the self-running demo.
// It is the default value for the *DEMOMAC command, so it starts
// if the SLCD5 is powered on with its serial port looped back to itself.
//---------------------------------------------------------------------
#define power_on_loopback_demo 1 /*
m29  // check for touch calibration
m10
m12
#end */


//---------------------------------------------------------------------
// MACROS #2 - #6
// These macros implement a number pad at relative (0,0)
// It assumes button BMP 19, 20 are loaded
//---------------------------------------------------------------------
#define number_pad 2 /*
```

```
// bitmap that holds displayed number
xi 12 0 0
// set font 24 for buttons
f 24
// define buttons
bd 1 0 32 1 "1" 9 5 10 11
bd 2 32 32 1 "2" 9 5 10 11
bd 3 64 32 1 "3" 9 5 10 11
bd 4 0 64 1 "4" 9 5 10 11
bd 5 32 64 1 "5" 9 5 10 11
bd 6 64 64 1 "6" 9 5 10 11
bd 7 0 96 1 "7" 9 5 10 11
bd 8 32 96 1 "8" 9 5 10 11
bd 9 64 96 1 "9" 9 5 10 11
bd 10 0 128 1 "*" 9 5 10 11
bd 0 32 128 1 "0" 9 5 10 11
bd 11 64 128 1 "#" 9 5 10 11
// tell user we're done
beep 10
/* a comment here only works if C compiler handles nested comments */
#end
   ...(end comment out contents) */

//---------------------------------------------------------------------
// Macro to set font and color for input box display. Foregorund color
// is XOR of background color to match the buttons
//---------------------------------------------------------------------
#define grey24 3 /*
S 333 CCC
f 24
#end  */

//---------------------------------------------------------------------
// These macros write text to the screen
//---------------------------------------------------------------------
#define button_text 4 /*
m2
t "`0`" 10 4
#end  */


//---------------------------------------------------------------------
// This macro enables the buttons specified in number_pad macro
// to write text to the screen inside a bitmap
//---------------------------------------------------------------------
#define attach_buttons 5 /*
xa 0 p 3 0
xa 1 p 3 1
xa 2 p 3 2
xa 3 p 3 3
xa 4 p 3 4
xa 5 p 3 5
xa 6 p 3 6
xa 7 p 3 7
xa 8 p 3 8
xa 9 p 3 9
#end  */

//---------------------------------------------------------------------
// This macro creates a keypad and displays the key
// using previous macros
//---------------------------------------------------------------------
#define keypad_demo 6 /*
```

```
s 0 1
m1
m4
#end  */

#define rand_draw 7 /*
p `R1:5`
m8
tr `R0:319` `R0:189` `R0:319` `R0:189` `R0:319` `R0:189`
m8
c `R0:319` `R0:159` `R10:40`
:repeat
#end */

#define rand_color 8 /*
s `R0:16` 1
#end */

#define demo_end 9 /*
m10
#end */

#define clear 10 /*
s 0 1
z
#end */

#define next_button 11 /*
o 0 0
f 13B
bd 1 257 202 1 "Next" 16 11 1 2
xm 1 `0`
#end */

#define splash_logos 12 /*
xi 5 0 0
m11 13
#end */

#define splash_panel 13 /*
xi 6 0 0
m11 14
#end */

#define splash_SLCD5 14 /*
xi 7 0 0
m11 15
#end */

#define splash_fonts 15 /*
z
f 24BC
t "On-board Proportional Fonts" 10 5
sc 0 0
o 20 40
f 8
t "8 point font: quick brown fox 0123"
f 16B
t "\n"
f 10
t "10 point: quick brown fox 0123"
f 16B
t "\n"
```

```
f 13
t "13 point: quick brown fox 0123"
f 16B
t "\n"
f 13B
t "13 point bold: quick brown fox 0123"
f 16B
t "\n"
f 16
t "16 point: quick brown fox 0123"
f 16B
t "\n"
f 16B
t "16 point bold: quick brown fox 0123"
f 18BC
t "\n18 point bold comic: quick fox 0123"
t "\n"
f 24
t "24 point: quick fox 0123"
f 24B
t "\n24 point bold: quick 01\n"
f 32
t "32 point & "
f 32B
t "32 bold"
m11 31
#end */

#define splash_keyboard 16 /*
z
// clear string memory
mpop -1
f 24BC
t "Easy to use buttons" 54 0
f 13B
// set up stateful button with macro callback
m17 "Off"
bd 2  50 24 2 "" "" 0 0 0 0 3 4
xa 2 p 17 "On  "
xa 2 r 17 "Off"
// setup demo instant button
bd 3 200 20 1 "Hold" 16 11 1 2
// adjust origin for keyboard
o 0 72
// keyboard row one
m18  10   0   0 1
m18  11  32   0 2
m18  12  64   0 3
m18  13  96   0 4
m18  14 128   0 5
m18  15 160   0 6
m18  16 192   0 7
m18  17 224   0 8
m18  18 256   0 9
m18  19 288   0 0
// keyboard row two
m18  20   0  32 Q
m18  21  32  32 W
m18  22  64  32 E
m18  23  96  32 R
m18  24 128  32 T
m18  25 160  32 Y
m18  26 192  32 U
```

```
m18  27 224  32 I
m18  28 256  32 O
m18  29 288  32 P
// keyboard row three
m18  30   0  64 A
m18  31  32  64 S
m18  32  64  64 D
m18  33  96  64 F
m18  34 128  64 G
m18  35 160  64 H
m18  36 192  64 J
m18  37 224  64 K
m18  38 256  64 L
m18  39 288  64 " "
// keyboard row four
m18  40   0  96 Z
m18  41  32  96 X
m18  42  64  96 C
m18  43  96  96 V
m18  44 128  96 B
m18  45 160  96 N
m18  46 192  96 M
m18  47 224  96 ,
m18  48 256  96 -
// special erase key
bd   49 288  96 1 "rub" 5 9 10 11
xm   49 20
// reset origin
o 0 0
// draw cursor
m19
// link to next screen
m11 21
#end */

#define keyboard_button_display 17 /*
f 13B
t "`0`" 85 34
#end */

#define keyboard_key 18 /*  index x y letter
bd `0` `1` `2` 1 "`3`" 11 9 10 11
xa `0` p 19 "`3`"
#end */

#define keyboard_press 19 /* letter
mpush "`0`"
t "`M`_    " 0 57
#end */

#define keyboard_erase 20 /*
mpop 1
m19
#end */

#define splash_charts 21 /*
z
f 24BC
t "Data visualization charts" 28 0
// 5 levelbars
o  43 40
m22 0
o  96 40
```

```
m22 1
o 149 40
m22 2
o 202 40
m22 3
o 255 40
m22 4
// 1 long chart
o 10 140
cd 0 0 0 301 49 1 3 1 100 008 2 F00 2 0F0 2 FFF
p 2
l 302 0 302 50
l 0 50 302 50
o 0 0
// link to next screen
m11 24
m23
#end */

#define levelbar_init 22 /*
ld `0` 0 0 20 80 0 0 1 888 100 F00 65 FF0 50 0F0
p 2
l 21 0 21 81
l 0 81 21 81
lv `0` 0
#end */

#define master_flopper 23 /*
// cycle through a prime (to slide on chart) number of relative randoms.
// this thing updates REALLY fast so we need to slow it down a little.
// 1
w 100
lv 0 `R10:50`
lv 1 `R20:60`
lv 2 `R20:60`
lv 3 `R20:60`
lv 4 `R10:50`
cv 0 `R10:20` `R40:60` `R80:100`
// 2
w 100
lv 0 `R10:50`
lv 1 `R40:80`
lv 2 `R40:90`
lv 3 `R20:80`
lv 4 `R10:50`
cv 0 `R10:30` `R30:50` `R60:90`
// 3
w 100
lv 0 `R20:70`
lv 1 `R40:60`
lv 2 `R60:100`
lv 3 `R20:60`
lv 4 `R20:50`
cv 0 `R20:40` `R40:60` `R60:80`
// 4
w 100
lv 0 `R20:50`
lv 1 `R20:60`
lv 2 `R60:80`
lv 3 `R20:60`
lv 4 `R20:50`
cv 0 `R20:30` `R30:60` `R70:80`
// 5
```

```
w 100
lv 0 `R30:50`
lv 1 `R20:100`
lv 2 `R40:100`
lv 3 `R20:100`
lv 4 `R30:50`
cv 0 `R10:30` `R50:80` `R70:100`
// 6
w 100
lv 0 `R40:80`
lv 1 `R20:100`
lv 2 `R40:100`
lv 3 `R20:100`
lv 4 `R40:80`
cv 0 `R10:30` `R60:90` `R60:100`
// 7
w 100
lv 0 `R20:50`
lv 1 `R30:60`
lv 2 `R60:80`
lv 3 `R30:60`
lv 4 `R20:50`
cv 0 `R10:30` `R30:70` `R60:100`
:repeat
#end */

#define splash_drawing 24 /*
z
f 18BC
t "Fast colorful drawing primitives!" 10 210
m11 25
m7
#end */

#define splash_info 25 /*
z
xi 8 0 0
s 0 1
// change this to "m11 9" to exit the demo instead of restarting it
m11 1
m26
#end */

#define logo_bounce 26 /*
// to the right and down
xi 9 0 0
xi 9 1 2
xi 9 2 4
xi 9 3 6
xi 9 4 8
xi 9 5 10
xi 9 6 12
xi 9 7 14
xi 9 8 16
xi 9 9 18
xi 9 10 20
xi 9 11 22
xi 9 12 24
xi 9 13 26
xi 9 14 28
xi 9 15 30
xi 9 16 32
xi 9 17 34
```

```
xi 9 18 36
xi 9 19 38
xi 9 20 40
xi 9 21 42
xi 9 22 44
xi 9 23 46
xi 9 24 48
xi 9 25 50
xi 9 26 52
xi 9 27 54
xi 9 28 56
xi 9 29 58
xi 9 30 60
xi 9 31 62
xi 9 32 64
xi 9 33 66
// to the right and up
xi 9 34 64
xi 9 35 62
xi 9 36 60
xi 9 37 58
xi 9 38 56
xi 9 39 54
xi 9 40 52
xi 9 41 50
xi 9 42 48
xi 9 43 46
xi 9 44 44
xi 9 45 42
xi 9 46 40
xi 9 47 38
xi 9 48 36
xi 9 49 34
xi 9 50 32
xi 9 51 30
xi 9 52 28
xi 9 53 26
xi 9 54 24
xi 9 55 22
xi 9 56 20
xi 9 57 18
xi 9 58 16
xi 9 59 14
xi 9 60 12
xi 9 61 10
xi 9 62 8
xi 9 63 6
xi 9 64 4
xi 9 65 2
xi 9 66 0
// to the left and down
xi 9 65 2
xi 9 64 4
xi 9 63 6
xi 9 62 8
xi 9 61 10
xi 9 60 12
xi 9 59 14
xi 9 58 16
xi 9 57 18
xi 9 56 20
xi 9 55 22
xi 9 54 24
```

```
xi 9 53 26
xi 9 52 28
xi 9 51 30
xi 9 50 32
xi 9 49 34
xi 9 48 36
xi 9 47 38
xi 9 46 40
xi 9 45 42
xi 9 44 44
xi 9 43 46
xi 9 42 48
xi 9 41 50
xi 9 40 52
xi 9 39 54
xi 9 38 56
xi 9 37 58
xi 9 36 60
xi 9 35 62
xi 9 34 64
xi 9 33 66
// to the left and up
xi 9 32 64
xi 9 31 62
xi 9 30 60
xi 9 29 58
xi 9 28 56
xi 9 27 54
xi 9 26 52
xi 9 25 50
xi 9 24 48
xi 9 23 46
xi 9 22 44
xi 9 21 42
xi 9 20 40
xi 9 19 38
xi 9 18 36
xi 9 17 34
xi 9 16 32
xi 9 15 30
xi 9 14 28
xi 9 13 26
xi 9 12 24
xi 9 11 22
xi 9 10 20
xi 9 9 18
xi 9 8 16
xi 9 7 14
xi 9 6 12
xi 9 5 10
xi 9 4 8
xi 9 3 6
xi 9 2 4
xi 9 1 2
:repeat
#end */

// to debug the mpush/mpop buffer
#define display_memory 27 /*
t "`M`"
#end */

// to debug poweron macros
```

```
#define pontest 28
beep 100
w 500
:repeat
#end

// startup calibration option
#define optional_calibration 29
s 0 1
z
f24B
// whole screen touch area
xs 128 0 0 319 239
// if touched, execute macro 30
xm 128 30
t "Touch screen to calibrate." 20 100
w 1000
t "."
w 1000
t "."
w 1000"
t "."
w 1000
xc 128
m8
#end

// calibrate then run demo
#define tc 30
tc
m10
m12
#end

#define splash_fixed_fonts 31 /*
z
f 24BC
t "Fixed Width Fonts Include:" 20 5
sc 0 0
o 25 40
f 4x6
t "4x6 font: quick brown fox 0123"
f 8x10
t "\n"
f 6x8
t "6x8 font: quick brown fox 0123"
f 8x12
t "\n"
f 8x8
t "8x8 font: quick brown fox 0123"
f 8x12
t "\n"
f 8x10
t "8x10 font: quick brown fox 0123"
f 8x12
t "\n"
f 8x13
t "8x13 font: quick brown fox 0123"
f 8x16
t "\n"
f 8x16
t "8x16 font: quick brown fox 0123"
f 8x16
```

```
t "\n"
f 12x24
t "12x24 font: quick 0123"
f 12x24
t "\n"
f 16x32i
t "16x32 font: 0123"
f 8x16
t "\n\n"
f 32x64
t "32X64\xB0C"
m11 16
#end */
```

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. Many 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<return>

This puts an "X" on the screen whenever a valid touch is recognized. To turn off, use:

*debug 0<return>

Appendix G - Tutorial

G.1 *Self-running demonstration*

Once the kit has been removed from the packaging and setup per the instructions provided on the CD, the built-in demo routine can be used to verify operation. The kit should have Jumper JP1 installed on the PowerCom4 board for the demo to run and verify operation. Also make sure to remove any cable from the DB9 connector while running the demo. The display should show "Touch screen to calibrate…" which indicates the demo is operating. Follow the prompts to run the demo.

Note that the Jumper JP1 / loopback plug loops back the SLCD5 serial data transmit and receive signals to the SLCD5. When the SLCD5 first powers on, it issues a ">" prompt. If it sees that prompt come back, it starts the demo routine.

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.

This tutorial assumes only a basic PC installation is available and therefore uses Hyperterminal to communicate. The program Procomm Plus from Symantec is recommended for advanced work as it supports scripting and other options. See http://www.symantec.com/procomm/

Once the SLCD5 has been connected to an available serial port, open Hyperterminal (Programs->Accessories->Communications->Hyperterminal) and enter SLCD5 for the name of the connection. Then enter the serial port connected to the SLCD5 in the "Connect using" field. Finally, set the Bits per second to 115200, and Flow control to Xon / Xoff. Hit OK and the program main screen appears. Hit the enter (return) key and you should see a '>' prompt character. This indicates that you are communicating with the SLCD5 board.

Now, go to menu File->Properties->Settings. Set Emulation to TTY. Press the "ASCII Setup", and set "Send line ends with line feeds", "Echo typed characters locally" (i.e. half duplex mode), and "Append line feeds to incoming line ends". Hit OK, OK to return to the main screen.

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.

When you want to run the Reach supplied BMPload program, you will need to logically disconnect Hyperterminal from the serial line. To do so, click on the icon showing a telephone with the handset and a small red arrow pointing down, or through the menu Call -> Disconnect. Reconnect again when BMPload is terminated.

G.3 *Simple commands*

This section presents some simple commands that illustrate some of the SLCD5 capabilities. It assumes that the bitmaps and macro files that were loaded from the factory are still present.

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 right click in Hyperterminal and choose "Transmit to Host"]

Clear the screen:

```
z
```

Type Hello World in a 24 point bold font starting at pixel x=100, y=110:

```
f 24B
t "Hello World" 100 110
```

Same as above, but with yellow text on a blue background:

```
z
f 24B
s 16 2
t "Hello World" 100 110
```

Create a vertical blue rectangle at x=40, y=100 to x = 60, y = 150.

```
z
s 2 1
r 40 100 60 150 1
```

Restore fore / back color to black on white:

```
s 0 1
z
```

Alternative way to do the blue rectangle without changing the foreground color:

```
z
r 40 100 60 150 1 00F
```

Display stored full screen bitmap:

```
xi 7 0 0
```

Define momentary button #1 named "Test" in the middle of the screen that sends a return string when pressed and released:

```
z
f 16B
bd 1 150 110 5 "Test" 2 8 10 11
```

G.4 *Macros*

The SLCD5 comes with pre-loaded macros to demonstrate this capability. Refer to Section E.6 above, or the file "Macros.mac" on the distribution CD.

Enter the following command to invoke the top level macro to display a keypad and display the last number pushed in an entry box:

```
m6
```

Macros have a repeat capability allowing them to loop while waiting for a button to be pressed that will jump to another macro. This is how the demo is implemented. To break out of repeating macros, hit the Escape key followed by Enter.

G.5 *Developing your Application*

Developing your application involves creating as many different screen pages as you need. For each page:

1. Design the bitmaps you want to use using a graphics editor. You can use Adobe Photoshop®, Photoshop Lite, GIMP (Open Source), or Windows Paint to create the bitmaps. See Appendix H.
2. Create a 320x240 pixel canvas using the above, and place the bitmaps where you want them to go. The graphics editor can be used to determine the top right point of the bitmap in terms of X, Y pixels. This is used in the SLCD5 command to locate the image and text.
3. Download the bitmaps using BMPload.
4. Write a series of SLCD5 commands to build the display screen and process the defined buttons.

Application note AN-100 describes an example program written for the Rabbit / Zworld RCN3720 core module. It is a useful starting point for developing SLCD5 control programs.

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)

Users can read the stored Highcolor value by using the PIXEL READ / WRITE command.

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 *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 I – SD card support features

I.1 *Overview*

The SLCD5 board contains an SD card slot. This slot can be replicated in a more convenient location using connector J14.

The SD card can perform the following functions:

- load new firmware into the SLCD5 code flash
- store bitmaps and macros file
- update on-board data flash with bitmaps and macros file
- save screen shots

NOTE: The SD card must be 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.

I.2 *Firmware Upgrade*

The SLCD5 contains an EEPROM that maintains the name of the file last used to load the current firmware. On power-on, the SLCD5 boot loader program checks if an SD card is present, and if so, reads the SD card looking for a file with the name:

SLCD????.ELF

where '?' is any character. If it finds this file, it compares the full file name with the stored name. If they are different, it initiates a firmware update using the SLCD????.ELF file on the SD card. 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.

I.3 *Application Programming (Bitmaps and macros)*

The BMPload program can store bitmaps and macros directly into the 4MB data flash on the SLCD5. It can also save the data to a file to be loaded from the SD card. This functionality is part of the “System Software Boot Process”. Refer section “System Software Boot Process” to for details.

I.4 *Screen Shot Saving*

Users can save the current contents of the entire screen on a file on an inserted SD Card. Refer to “SAVE SCREEN SHOT TO SD CARD” command for details.

I.5 *Summary*

Use the BMPload program to create the SLCD.BIN file containing the bitmaps and macros for the application. Then follow the scenarios below:

Development:

Store SLCD.BIN on SD card

Production, SLCD.BIN less than 4MB:

Store SLCD.BIN and FLASH1.INI on SD card. Install card and power-on SLCD5. Once SLCD5 generates user prompt, remove SD card

Production, SLCD.BIN more than 4MB:

Store SLCD.BIN and on SD card. Include installed SD card with SLCD5.

Appendix J – RS485 Multipoint Communications

J.1 *Overview*

The SLCD5 board Revision G does not have RS485 as a physical interface option. However an external RS232 to RS485 converter can be used. This type of adapter automatically enables the RS485 transmitter when the RS232 transmit data is active. Either half duplex or full duplex RS485 can be supported.

In order to support multipoint communications, the Version 2.3.0 and above software has an option to support addressed polling. This forces all SLCD5 responses including button pushes to be queued and reported only with a poll command. This appendix describes how to use this protocol.

The protocol supports a maximum of 255 SLCD5 controllers on a shared line; the actual limit may be less than this due to physical bus loading limitations.

J.2 *Setup*

A setup command is used to place the unit into RS485 mode. This mode is saved in non-volatile memory and will remain enabled unless explicitly disabled. Once enabled, the SLCD5 will not respond to commands unless they are preceded by the RS485 address header.

Setup command:

```
*rs485 <SOF><AD1><AD2><return>
```

`<SOF>` single ASCII character to be used as the "Start Of Frame" character for the shared communication bus. This should not be the '>' character, and must be unique so that it is not used for anything except the start of frame.

`<AD1>` single ASCII character from '0' to '9' and 'A' to 'F' which is the most significant address character.

`<AD1>` single ASCII character from '0' to '9' and 'A' to 'F' which is the least significant address character.

NOTE: address FF is reserved for the host address.

Example:

```
*rs485 /12<return>
```

This sets the 485 mode and specifies '/' as the SOF character and address hex 12 (equivalent to decimal 18) as the unit address. Note that if the character '/' will be used in a text command to the SLCD5, then another character such as '`' (backtick) should be used as the SOF.

For the example above, the SLCD5 responds as follows:

```
RS485 Mode SOF 0x2F (/) ADR 12<return>
```

This response verifies the setup since from this point onwards the SLCD5 will use these selections for addressing.

J.3 *Command Operation*

Once in rs485 mode, all commands to the SLCD5 must start with the three character address prefix specified in the setup command, and the selected SOF character should not be used within the command itself. Otherwise, the command syntax is the same as non rs485 mode. The unit responds to commands exactly the same as normal mode except that all responses start with the three character prefix <SOF>FF. The FF address is reserved for the address of the host on the rs485 bus.

Examples:

```
SEND:      /12z<return>
RECEIVE:   /FF>
```

J.4 *Button responses and polling*

All messages from the SLCD5 that are caused by button presses (for example button notification and macro execution messages) are queued in the order they occurred and are sent when the host next initiates communication with the unit. This includes the poll command which is a null command - the three character prefix followed by a <return>. If the host happens to issue a command (for example to change a value on the display) and a button is simultaneously pushed, the host will receive the button notification message before the command completed response.

Polling example: (button 1 pushed)

```
SEND:      /12<return>
RECEIVE:   /FFx1<return>
```

Button response during display command example: (button 1 pushed)

```
SEND:      /12t "12:15pm"<return>
RECEIVE:   /FFx1<return>
RECEIVE:   /FF><return>
```

Appendix K – SD CARD REMOTE

K.1 *Overview*

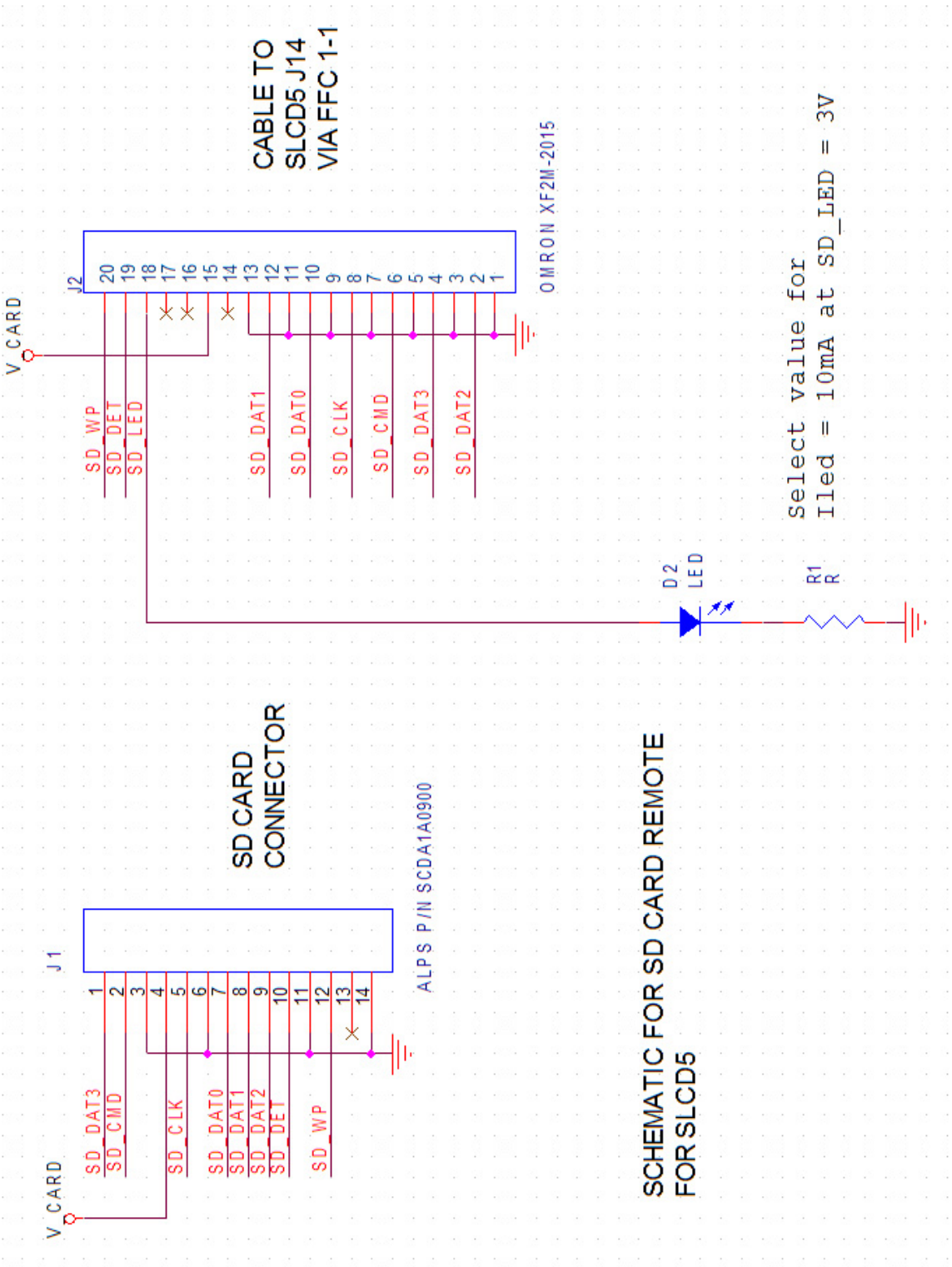
The SLCD5 has an on-board SD card slot. It may be more convenient to place this SD card connector elsewhere; in this case, connector J14 can be used to cable to a small application-specific board containing a remote SD card slot.

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 ohms.

K.2 *Schematic*



8. Software Manual Change History

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 Appendix E..
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 Image 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 Feature 87 (Bitmap translation needed (24bit to High Color))
Feature 142 (UTF8 support for UNICODE fonts)
Feature 133 (Macros support up to ten parameters)
Feature 132 (Macro call by name). Macros can also use a name.
Feature 151 (Incoming CRC). Commands sent to the SLCD5 can use a CRC.
Feature 128 (Add index parameter to listing commands).
Feature 127 (Macro Save State and Restore State commands macro depth aware). These commands were added to the manual.
Feature 109 (Version string needs additional build information).
Feature 55 (Ability to remember serial baud rates changes).
Defect 115 (SLIDER DEFINE continuous touch has no impact)
Defect 187 (Transparent bitmaps not documented and look iccky....)
Defect 219 (DISABLE/ENABLE TOUCH does not document range optional parameters)
Defect 231 (Section missing on how to use external fonts).
Defect 241 (Internal Arguments description valid only within macro)
Defect 242 (DRAW RECTANGLE doesn't do what manual says)
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