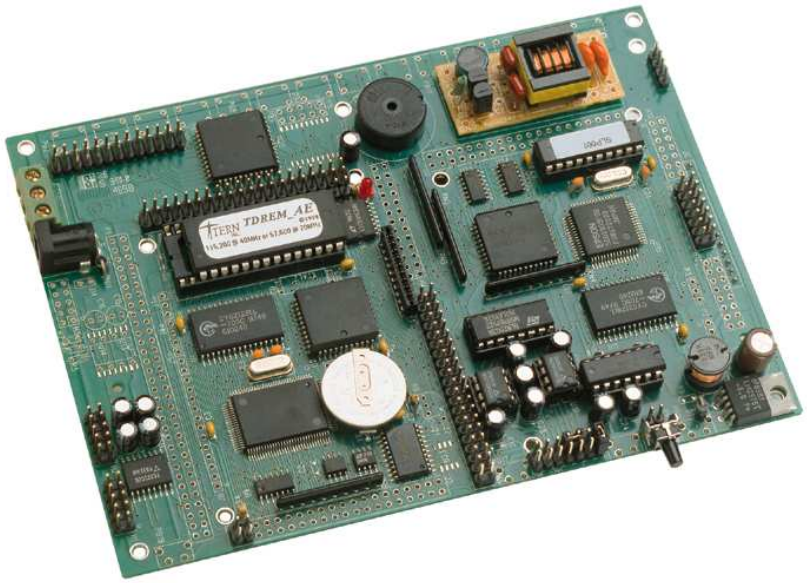
SmartLCD™

1/4 VGA Graphic LCD Interface Controller with 12-bit ADC and 12-bit DAC



Technical Manual

TERN INC.
1950 5th Street, Davis, CA 95616, USA
Tel: 530-758-0180 Fax: 530-758-0181
Email: sales@tern.com http://www.tern.com

COPYRIGHT

SmartLCD, NT-Kit, MemCard, A-Engine, A104, and ACTF are trademarks of TERN, Inc.
Am188ES and Am186ES are trademarks of Advanced Micro Devices, Inc.
Paradigm C/C++ is a trademark of Paradigm Corporation
Microsoft, MS-DOS, Windows95/98/2000/XP are trademarks of Microsoft Corporation.
IBM is a trademark of International Business Machines Corporation.

Version 4.0

October 25, 2010

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of TERN, Inc.

© 1999-2010 TERN INC.
1950 5th Street, Davis, CA 95616, USA
Tel: 530-758-0180 Fax: 530-758-0181
Email: ***sales@tern.com*** ***http://www.tern.com***

Important Notice

TERN is developing complex, high technology integration systems. These systems are integrated with software and hardware that are not 100% defect free. ***TERN products are not designed, intended, authorized, or warranted to be suitable for use in life-support applications, devices, or systems, or in other critical applications. TERN*** and the Buyer agree that ***TERN*** will not be liable for incidental or consequential damages arising from the use of ***TERN*** products. It is the Buyer's responsibility to protect life and property against incidental failure.
TERN reserves the right to make changes and improvements to its products without providing notice.
Temperature readings for controllers are based on the results of limited sample tests; they are provided for design reference use only.

Chapter 1: Introduction

1.1 Functional Description

The ***SmartLCD (SL)*** is a complete C/C++ programmable user interface that includes a 320 x 240 graphic LCD (1/4 VGA), touch screen, CCFL backlighting, LCD controller (SED1335), and an embedded controller (188 CPU). The available on-board power supplies include a 5V switching regulator, -24V voltage source, CCFL backlighting power, and software-programmable LCD contrast voltage.

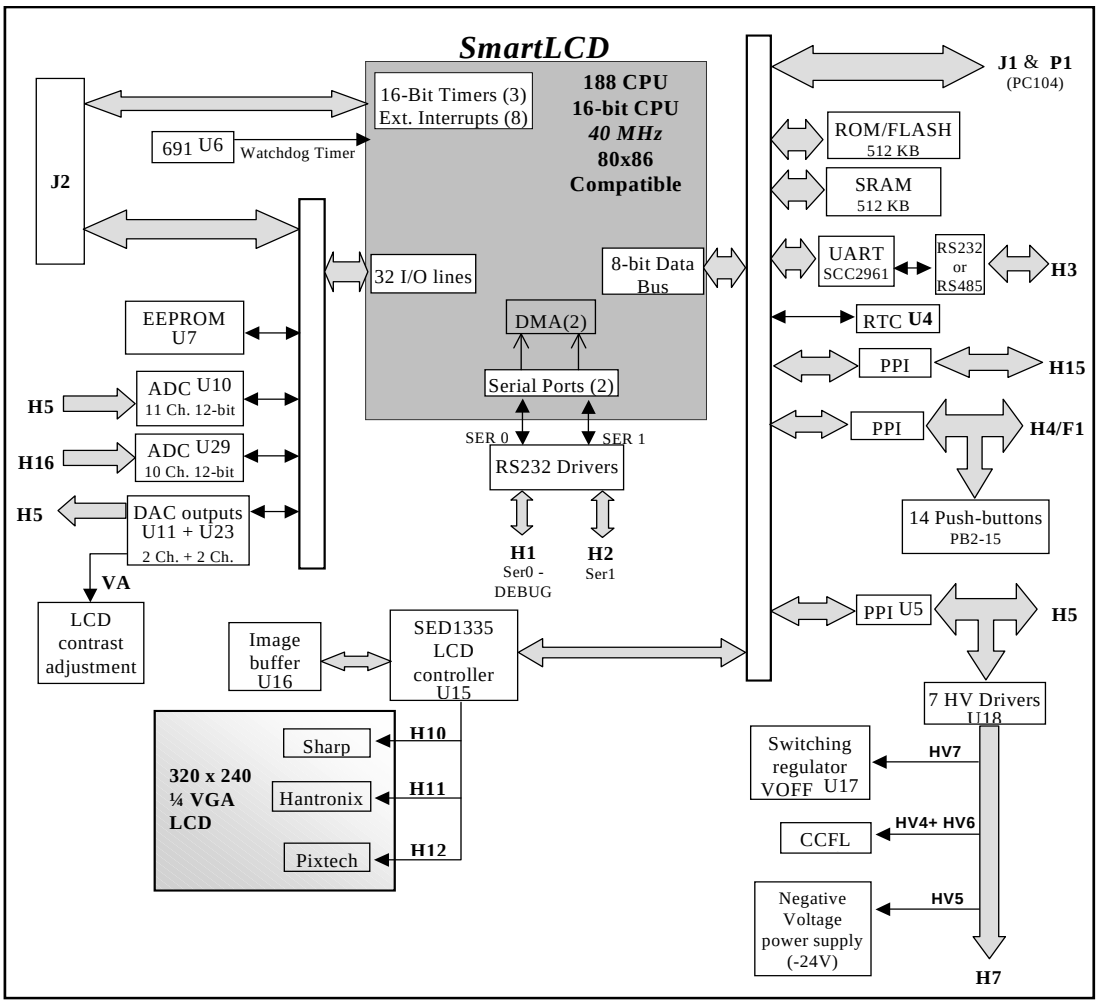


Figure 1.1 Functional block diagram of the SmartLCD

Measuring 6.5 by 4.3 inches, the ***SmartLCD*** supports 512 KB ROM/Flash, 512 KB battery-backed SRAM, a 512-byte serial EEPROM, real-time clock (RTC72423), lithium coin battery, three timer/counters, watchdog timer, PWM, and up to three serial ports (RS-232/485). It also supports up to 20 channels 12-bit ADC, up to four channels of 12-bit DAC, and seven high-voltage drivers. A total of three PPIs (82C55) provide 24x3 TTL I/Os in addition to the 32 multifunctional I/O pins from the CPU. Additional memory

can be added via the FlashCore-0 (FC-0) interface. Using the FC-0, 50-pin Compact Flash memory cards may be added, with a capacity of up to 1GB.

Two DMA-driven serial ports from the CPU support high-speed, reliable serial communication at a rate of up to 115,200 baud. An optional UART SCC2691 may be added for a third UART on board and can be configured as RS-232 or RS-485, supporting either normal 8-bit or 9-bit multi-drop RS485/422 network with twisted-pair wiring.

There are three 16-bit programmable timers/counters and a watchdog timer. Two timers can be used to count or time external events, at a rate of up to 10 MHz, or to generate non-repetitive or variable-duty-cycle waveforms as PWM outputs. Pulse Width Demodulation (PWD), a distinctive feature of the 188 CPU, can be used to measure the width of a signal in both its high and low phases. It can be used in many applications, such as bar-code reading.

The 32 I/O pins from the CPU are multifunctional and user-programmable. Some of the I/O pins are used for serial ports, timer I/Os, or clock, You may have 15 or more lines free to use, depending on your application.

Three 82C55 PPI chips (U5, U20, U27) provide 72 bi-directional I/O lines, of which 3 lines (U5 I20, I21, I22) are used for the ADC (U10) and 22 lines (U27 T00-07 T10-15, T20-27) are used to interface to 70 key matrix touch screen and 14 push bottoms. The 24 I/O lines at H15 from U20 and 21 I/O pins at H5 from (U5) are free for application.

Figure 1.2 Front view of the SmartLCD: 320x240 graphic LCD displaying bird

In order to interface to a ¼ VGA, 320x240-pixel graphic LCD, an LCD controller (SED1335, SMOS) and an image buffer are on-board. The 188 CPU can communicate with the SED1335 via high-speed 8-bit data bus. Software drivers and sample programs are available for applications that require both graphics and text display. Power supplies for the CCFL backlighting, -24V negative voltage, and the regulated 5V can be controlled by software for low power consumption in portable battery applications. The battery voltage can be measured using one channel of 12-bit ADC. The LCD display contrast voltage can be adjusted using one channel of 12-bit DAC.

A transparent 10x7-key matrix touchscreen is adhered to the front of the 320x240 graphic LCD. A sample program “sl_grid.c” will highlight the 70-key grid and respond to touch.

There are eight external interrupt inputs. Schmitt-trigger inverters are provided for six interrupt inputs, to increase noise immunity and transform slowly-changing input signals into fast-changing and jitter-free signals.

A supervisor chip with power failure detection, a watchdog timer, and an LED are on-board.

An optional real-time clock provides information on the year, month, date, hour, minute, second, and 1/64 second, and an interrupt signal.

Up to two 12-bit ADC chips (TLC2543) can be installed. The ADC chips each have 11 channels of analog inputs with sample-and-hold and a high-impedance reference input (5V) that facilitate ratiometric conversion, scaling, and isolation of analog circuitry from logic and supply noise, supporting conversion up

to a sample rate of approximately 10 KHz based on the 40 MHz CPU. The ***SL*** may use ADC to monitor the power input.

Up to two, 2-channel 12-bit DAC chips can be installed on-board supporting a total of four channels 12-bit, 0-4.095V analog voltage outputs capable of sinking or sourcing 5 mA. By default, one DAC channel is used to control the LCD contrast.

There are seven solenoid drivers on-board, each capable of sinking 350 mA at 50V. Two solenoid drivers are used to control the CCFL backlighting, and one solenoid driver is used to control the negative power source.

1.2 Features

Standard Features

- Dimensions: 6.5 x 4.3 inches
- Program in C/C++
- Power consumption:
  100/170 mA at 9V for 40/20 MHz without CCFL and negative power-on.
  90/130 mA at 12V for 40/20 MHz without CCFL and negative power-on.
  In standby mode, less than 1 mA with Switching Regulator VOFF=high.
  280 mA at 12V with the CCFL back lighting and negative power on.
- Compact Flash card support, up to 1GB (via FC-0)
- Power input: +8.5V to +12 V unregulated DC. On-board +5V switching regulator.

Although the Switching Regulator is rated at +24V, no more than +12V input is allowed. Higher input voltage will damage the CCFL power supply.

- 320 x 240 graphic LCD, SED 1335, 10x7 touch screen, CCFL backlighting
  (module size 167x109 mm; viewing area 121x91 mm)
- 16-bit CPU (188 CPU), Intel 80x86 compatible, 40 MHz or 20 MHz
- 512KB SRAM, and support for up to 512KB Flash/ROM
- 2 high-speed PWM outputs and Pulse Width Demodulation
- 24x3 bi-directional I/O lines from three PPIs (82C55).
- 512-byte serial EEPROM, external interrupt inputs, 3 16-bit timer/counters
- 2 CPU serial ports (RS-232)
- 2 channels of 12-bit DAC, 0-4.095V output. One DAC is used for the LCD contrast adjustment.
- 11 channels of 12-bit ADC in U10 and H5, sample rate up to 10 KHz (TLC2543)
- Supervisor chip (691) for power failure, reset, and watchdog
- 7 solenoid drivers: 2 for CCFL, one for negative power supply.

Optional Features:

- 10 additional channels of 12-bit ADC, sample rate up to 10 KHz (TLC2543)
- 2 additional channels of 12-bit DAC, 0-4.095V output
- 10 additional channels of 12-bit ADC in U29 and H16, sample rate up to 10 KHz (TLC2543)
- SCC2691 UART (on-board) supports 8-bit or 9-bit networking
  UART comes with RS232 (default) or 485 drivers
- Real-time clock RTC72423, lithium coin battery

1.3 Physical Description

The physical layout of the SmartLCD is shown in Figure 1.3.

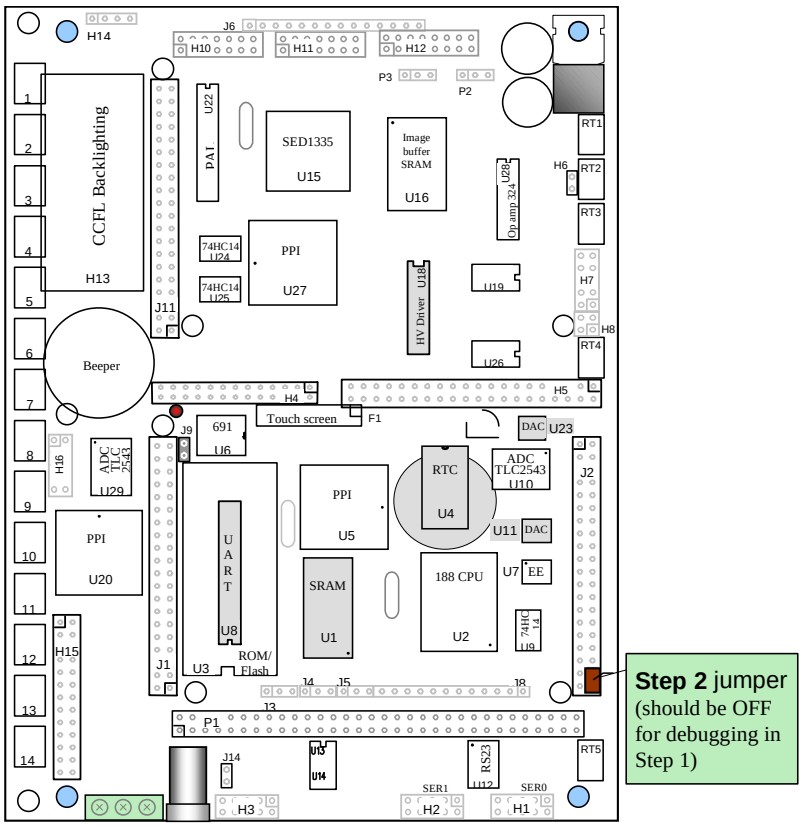
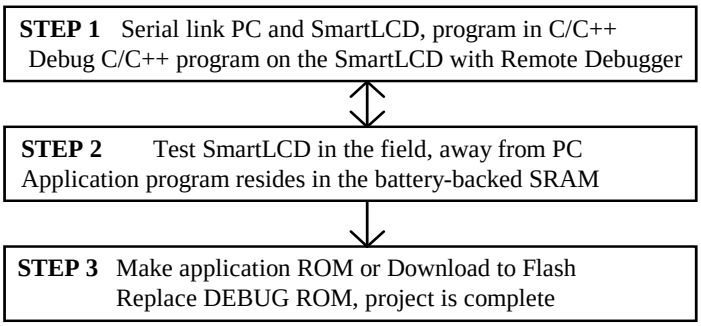


Figure 1.3 Physical layout of the SmartLCD

1.4 SmartLCD Programming Overview

Development of application software for the SmartLCD consists of three easy steps, as shown in the block diagram below.



You can program the SmartLCD from your PC via serial link with an RS232 interface. Your C/C++ program can be remotely debugged over the serial link at a rate of 115,000 baud. The C/C++ Evaluation Kit

(EV-P) or Development Kit (DV-P) from TERN provides the Paradigm C/C++ environment, complete with a text editor, compiler, locater, I/O driver libraries, sample programs, and documentation. These kits also include a DEBUG ROM (*AE_0_115)* to communicate with the Paradigm C/C++ environment, a PC-V25 (DB9-IDE10) cable to the connect the controller to the PC, and a 9-volt wall transformer. *See your Evaluation/Development Kit Technical Manual for more information on these kits.*

After you debug your program, you can test run the SmartLCD in the field, away from the PC, by changing a single jumper, with the application program residing in the battery-backed SRAM. When the field test is complete, application ROMs can be produced to replace the DEBUG ROM. The .HEX or .BIN file can be easily generated using Paradigm C/C++. You may also use the DV-P Kit to download your application code to on-board Flash.

The three steps in the development of a C/C++ application program are explained in detail below.

1.4.1 Step 1

STEP 1: Debugging

- Write your C/C++ application program in C/C++.
- Connect your controller to your PC via the PC-V25 serial link cable.
- Use Paradigm C/C+ to compile, link, locate, download, and debug your C/C++ application program.

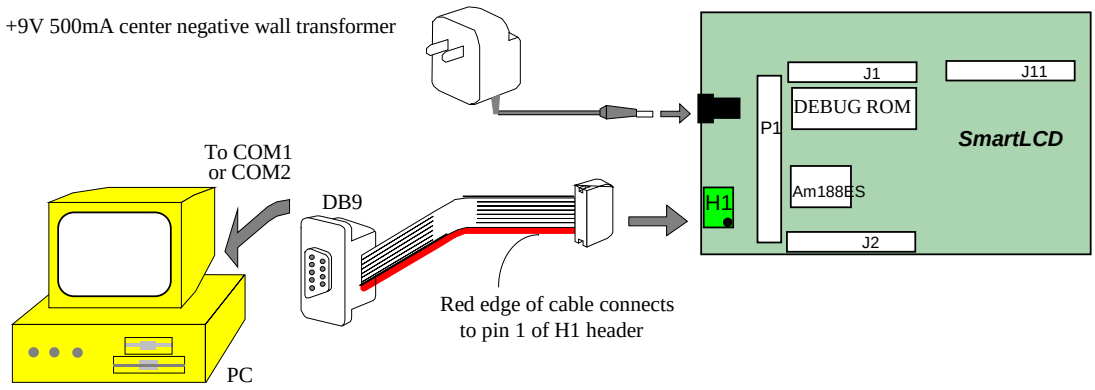


Figure 1.4 Step 1 connections for the SmartLCD

1.4.2 Step 2

STEP 2: Standalone Field Test.

With Debug ROM installed and your application program residing in SRAM,

- Set the jumper on J2 pins 38-40 (Figure 1.5).
- At power-on or reset, if J2 pin 38 (P4) is low, the CPU will run the code that resides in the battery-backed SRAM.
- If a jumper is on J2 pins 38-40 at power-on or reset, the SmartLCD will operate in Step 2 mode. If the jumper is off J2 pins 38-40 at power-on or reset, the SmartLCD will operate in Step 1 mode. The status of J2 pin 38 (signal P4) of the 188 CPU is only checked at power-on or at reset.

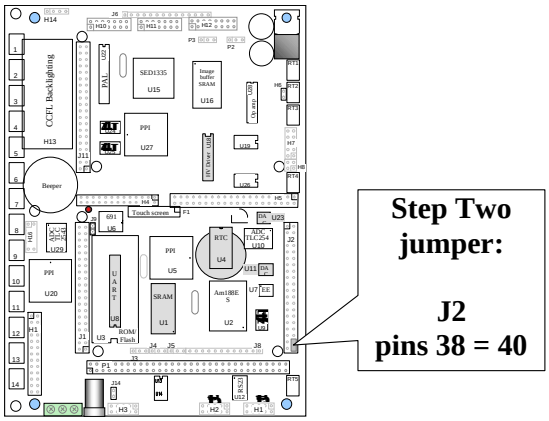


Figure 1.5 Location of Step 2 jumper on the SmartLCD

1.4.3 Step 3

STEP 3: Production (DV-P Kit only)

Generate the application .BIN or .HEX file, make production ROMs or download your program to FLASH via ACTF.

- If you are satisfied with the Step 2 standalone test, you can go back to your PC to generate your application ROM to replace the DEBUG ROM (*AE_0_115*). You need to change **PDREMOTE/ROM** to **No Target/ROM** in the Target Expert, in the Paradigm C/C++ environment.

The DV-P Kit is required to complete Step 3.

Please refer to the Tutorial of the Technical Manual of the EV-P/DV-P Kit for further details on programming the SmartLCD.

1.5 Minimum Requirements for SmartLCD System Development

1.5.1 Minimum Hardware Requirements

- PC or PC-compatible computer with serial COMx port that supports 115,200 baud
- SmartLCD controller with DEBUG ROM *AE_0_115*

- PC-V25 serial cable (RS232; DB9 connector for PC COM port and IDE 2x5 connector for controller)
- Center negative wall transformer (+9V 500 mA)

1.5.2 Minimum Software Requirements

- TERN CD-Rom for EV-P or DV-P
- PC software environment: Windows95/98/2000/ME/NT/XP

The C/C++ Evaluation Kit (EV-P) and C/C++ Development Kit (DV-P) are available from TERN. The EV-P Kit is a limited-functionality version of the DV-P Kit. With the EV-P Kit, you can program and debug the SmartLCD in Step 1 and Step 2, but you cannot run Step 3. In order to generate an application ROM/Flash file, make production version ROMs, and complete the project, you will need the Development Kit (DV-P).

Chapter 2: Installation

2.1 Software Installation

Please refer to the Technical manual for the "C/C++ Development Kit and Evaluation Kit for TERN Embedded Microcontrollers" for information on installing software.

The README.TXT file on the TERN EV/DV disk contains important information about the installation and evaluation of TERN controllers.

2.2 Hardware Installation

Hardware installation for the SmartLCD consists primarily of connecting the microcontroller to your PC.

Overview

- Connect PC-V25 cable:
  For debugging (STEP 1), place the 5x2 pin header on SER0 (H1) with red edge of cable at pin 1 of H1
- Connect wall transformer:
  Connect 9V wall transformer to power and plug into power jack

2.2.1 Connecting the SmartLCD to the PC

The following diagram (Figure 2.1) illustrates the connection between the SmartLCD and the PC. The SmartLCD is linked to the PC via a serial cable (PC-V25).

The *AE_0_115* DEBUG ROM communicates through SER0 by default. Install the 5x2 IDC connector on the SER0 header (H1). ***IMPORTANT:*** *Note that the* ***red*** *side of the cable must point to pin 1 of the H1 header*. The DB9 connector should be connected to one of your PC's COM Ports (COM1 or COM2).

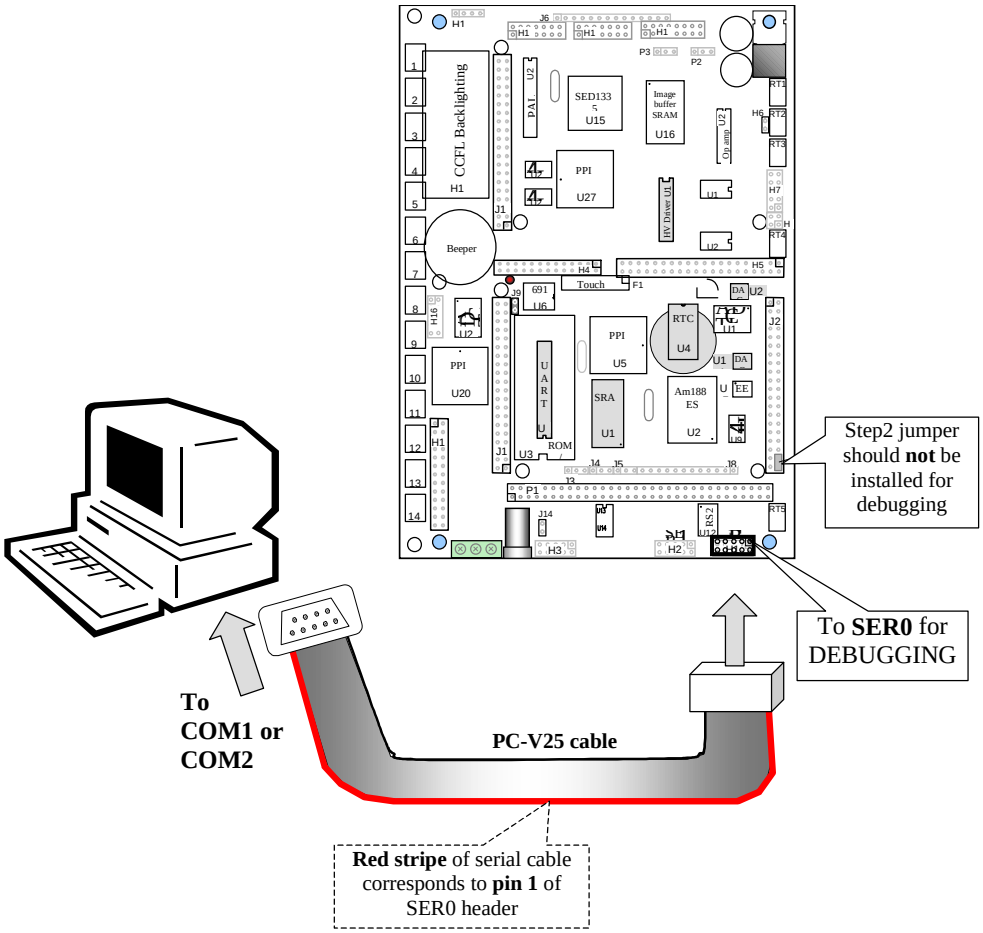


Figure 2.1 Connecting the SmartLCD to the PC

2.2.2 Powering-on the SmartLCD

Connect a wall transformer +9V DC output to the DC power jack.

The on-board LED should blink twice and remain on after the SmartLCD is powered-on or reset, as shown in Figure 2.2.

While the on-board switching regulator is rated for an input of up to +24V, you may provide a maximum input voltage of +12V. The input voltage is also routed to the CCFL power supply which only allows a maximum input of +12V. Any higher voltage will damage the CCFL power supply.

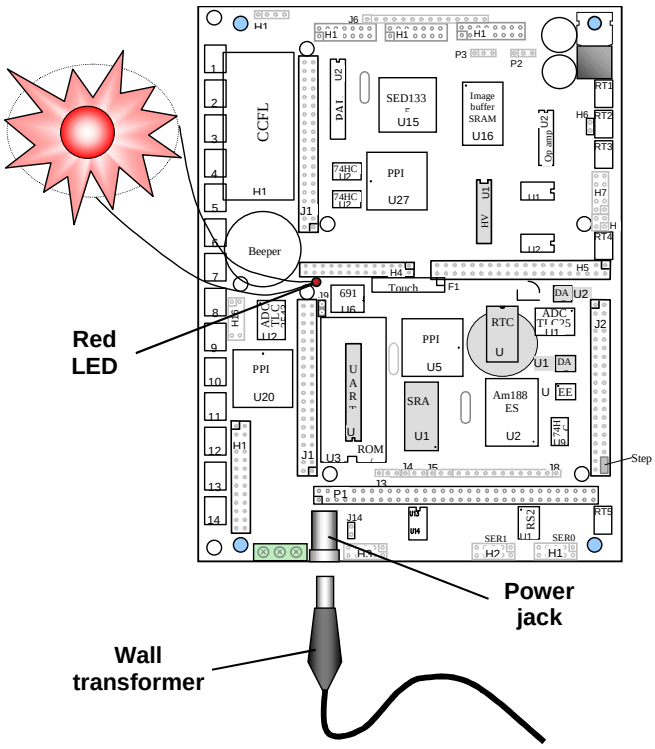


Figure 2.2 The LED blinks twice after the SmartLCD is powered-on or reset

Chapter 3: Hardware

3.1 188 CPU – Introduction

The 188 CPU is based on industry-standard x86 architecture. The 188 CPU controllers are higher-performance, more integrated versions of the 80C188 microprocessors. In addition, the 188 CPU has new peripherals. The on-chip system interface logic can minimize total system cost. The 188 CPU has two asynchronous serial ports, 32 PIOs, a watchdog timer, additional interrupt pins, a pulse width demodulation option, DMA to and from serial ports, a 16-bit reset configuration register, and enhanced chip-select functionality.

3.2 188 CPU – Features

3.2.1 Clock

Due to its integrated clock generation circuitry, the 188 CPU microcontroller allows the use of a times-one crystal frequency. The design achieves 40 MHz CPU operation, while using a 40 MHz crystal.

The system CLKOUTA and CLKOUTB signal is not routed out from the CPU in the SmartLCD.

CLKOUTA remains active during reset and bus hold conditions. The SmartLCD initial function ae_init(); disables CLKOUTA and CLKOUTB with clka_en(0); and clkb_en(0);

You may use clka_en(1); to enable CLKOUTA.

3.2.2 External Interrupts and Schmitt Trigger Input Buffer

There are eight external interrupts: INT0-INT6 and NMI.

/INT0, J2 pin 8, is used by SCC2691 UART, if it is installed.
/INT1, J2 pin 6
/INT2, J2 pin 19
/INT3, J2 pin 21
/INT4, J2 pin 33
INT5=P12=DRQ0, J2 pin 5, used as output for LED/EE/HWD
INT6=P13=DRQ1, J2 pin 11
/NMI, J2 pin 7

Six external interrupt inputs, /INT0-4 and /NMI, are buffered by Schmitt-trigger inverters (U9 74HC14), in order to increase noise immunity and transform slowly changing input signals to fast changing and jitter-free signals. As a result of this buffering, these pins are capable of only acting as input.

These buffered external interrupt inputs require a falling edge (HIGH-to-LOW) to generate an interrupt.

The SmartLCD uses vector interrupt functions to respond to external interrupts. Refer to the 188 CPU User's manual for information about interrupt vectors.

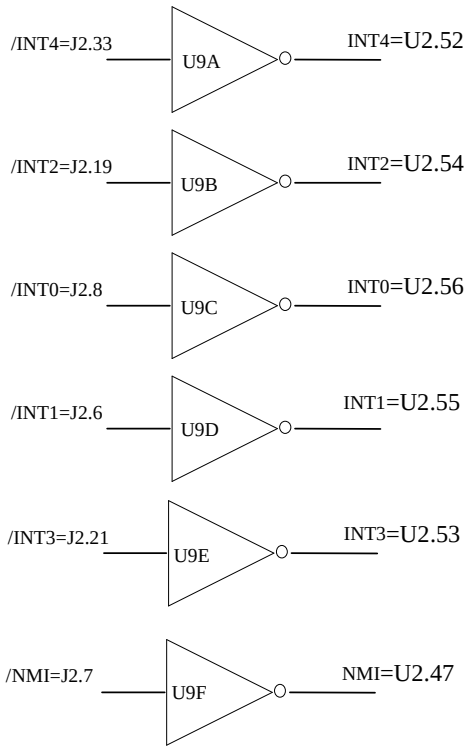


Figure 3.1 External interrupt inputs with Schmitt-trigger inverters

3.2.3 Asynchronous Serial Ports

The 188 CPU has two asynchronous serial channels: SER0 and SER1. Both asynchronous serial ports support the following:

- Full-duplex operation, 8-bit data transfers, no parity, one stop bit
- Error detection, no hardware flow control
- DMA receive from serial ports, transmit interrupts for each port
- Maximum baud rate of 1/16 of the CPU clock speed, independent baud rate generators

The software drivers for each serial port implement a ring-buffered DMA receiving and ring-buffered interrupt transmitting arrangement. See the sample files **s1_echo.c** and **s0_echo.c** in the tern\186\samples\ae directory.

The optional external SCC2691 UART is located in the U8 underneath the ROM. For more information about the external UART SCC2691, please refer to section 3.4.5 and Appendix B.

3.2.4 Timer Control Unit

The timer/counter unit has three 16-bit programmable timers: Timer0, Timer1, and Timer2.

Timer0 and Timer1 are connected to four external pins:

Timer0 output = P10 = J2 pin 12
Timer0 input = P11 = J2 pin 14
Timer1 output = P1 = J2 pin 29
Timer1 input = P0 = J2 pin 20

These two timers can be used to count or time external events, or they can generate non-repetitive or variable-duty-cycle waveforms.

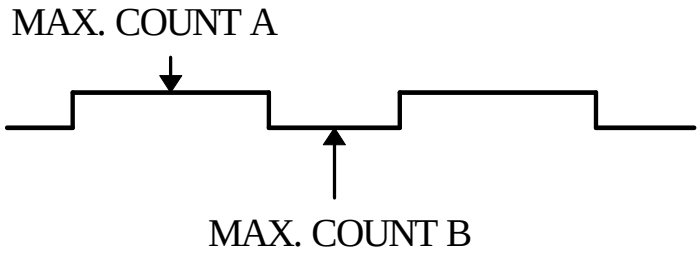
Timer2 is not connected to any external pin. It can be used as an internal timer for real-time coding or time-delay applications. It can also prescale timer 0 and timer 1 or be used as a DMA request source.

The maximum rate at which each timer can operate is 10 MHz, since each timer is serviced once every fourth clock cycle. Timer output takes up to six clock cycles to respond to clock or gate events. See the sample programs ***timer02.c*** and ***ae_cnt0.c*** in the **`\samples\ae`** directory.

3.2.5 PWM outputs and PWD

The Timer0 and Timer1 outputs can also be used to generate non-repetitive or variable-duty-cycle waveforms. The timer output takes up to 6 clock cycles to respond to the clock input. Thus the minimum timer output cycle is 25 ns x 6 = 150 ns (at 40 MHz).

Each timer has a maximum count register that defines the maximum value the timer will reach. Both Timer0 and Timer1 have secondary maximum count registers for variable duty cycle output. Using both the primary and secondary maximum count registers lets the timer alternate between two maximum values.



Pulse Width Demodulation can be used to measure the input signal's high and low phases on the /INT2=J2 pin 19. See the sample code ae_pwd.c in the tern\186\samples\ae directory.

3.2.6 Power-save Mode

The SmartLCD can be used for low power consumption applications. The power-save mode of the 188 CPU reduces power consumption and heat dissipation, thereby extending battery life in portable systems. In power-save mode, operation of the CPU and internal peripherals continues at a slower clock frequency. When an interrupt occurs, it automatically returns to its normal operating frequency.

The RTC72423 on the SmartLCD has a VOFF signal routed to H7 pin 10. VOFF is controlled by the battery-backed RTC72423. The VOFF signal can be programmed by software to be in tri-state or to be active low. The RTC72423 can be programmed in interrupt mode to drive the VOFF pin at 1/64-second, 1 second-1 minute, or 1-hour intervals. The user can use the VOFF line to control the 5V switching power regulator on/off. More details are available in the sample file ***poweroff.c*** in the **`186\samples\ae`** sub-directory. In power-off mode, the VOFF pin is pulled high via a 1M-ohm resistor, and the switching regulator will be turned off. While in power-off mode, less than 1 mA overall current consumption can be achieved. An external push button (RT1) or any external signal can short the VOFF pin to GND, to "wake up" from power-off mode.

3.3 188 CPU PIO lines

The 188 CPU has 32 pins available as user-programmable I/O lines. Each of these pins can be used as a user-programmable input or output signal, if the normal-shared function is not needed. A PIO line can be configured to operate as an input or output with or without a weak pull-up or pull-down, or as an open-drain output. A pin's behavior, either pull-up or pull-down, is pre-determined and shown in the table below.

After power-on/reset, PIO pins default to various configurations. The initialization routine provided by TERN libraries reconfigures some of these pins as needed for specific on-board usage, as well. These

configurations, as well as the processor-internal peripheral usage configurations, are listed below in Table 3.1.

PIO	*Function*	*Power-On/Reset status*	*SmartLCD Pin Number*	*SmartLCD Initial*
P0	Timer1 in	Input with pull-up	J2 pin 20	Input with pull-up
P1	Timer1 out	Input with pull-down	J2 pin 29	CLK_1
P2	/PCS6/A2	Input with pull-up	J2 pin 24	RTC select
P3	/PCS5/A1	Input with pull-up	J2 pin 15	SCC2691 select
P4	DT/R	Normal	J2 pin 38	Input with pull-up **Step 2**
P5	/DEN/DS	Normal	J2 pin 30	Input with pull-up
P6	SRDY	Normal	J2 pin 35	Input with pull-down
P7	A17	Normal	J8 pin 3	A17
P8	A18	Normal	J8 pin 2	A18
P9	A19	Normal	J8 pin 1	A19
P10	Timer0 out	Input with pull-down	J2 pin 12	Input with pull-down
P11	Timer0 in	Input with pull-up	J2 pin 14	Input with pull-up
P12	DRQ0/INT5	Input with pull-up	J2 pin 5	Output for LED/EE/HWD
P13	DRQ1/INT6	Input with pull-up	J2 pin 11	Input with pull-up
P14	/MCS0	Input with pull-up	J2 pin 37	Input with pull-up
P15	/MCS1	Input with pull-up	J2 pin 23	Input with pull-up
P16	/PCS0	Input with pull-up	U22 pin 1	PAL chip select
P17	/PCS1	Input with pull-up	J2 pin 13	U5 PPI, 82C55 select
P18	CTS1/PCS2	Input with pull-up	J2 pin 22	J11 pin 19, chip select
P19	RTS1/PCS3	Input with pull-up	J2 pin 31	Input with pull-up
P20	RTS0	Input with pull-up	J2 pin 27	Input with pull-up
P21	CTS0	Input with pull-up	J2 pin 36	Input with pull-up
P22	TxD0	Input with pull-up	J2 pin 34	TxD0
P23	RxD0	Input with pull-up	J2 pin 32	RxD0
P24	/MCS2	Input with pull-up	J2 pin 17	Input with pull-up
P25	/MCS3	Input with pull-up	J2 pin 18	Input with pull-up
P26	UZI	Input with pull-up	J2 pin 4	Input with pull-up*
P27	TxD1	Input with pull-up	J2 pin 28	TxD1
P28	RxD1	Input with pull-up	J2 pin 26	RxD1
P29	/CLKDIV2	Input with pull-up	J2 pin 3	Input with pull-up*
P30	INT4	Input with pull-up	J2 pin 33	Input with pull-up
P31	INT2	Input with pull-up	J2 pin 19	Input with pull-up

Note: * P26 and P29 must NOT be forced low during power-on or reset

Table 3.1 I/O pin default configuration after power-on or reset

Four external interrupt lines are not shared with PIO pins:

INT0 = J2 pin 2
INT1 = J2 pin 6
INT3 = J2 pin 21

The 32 PIO lines, P0-P31, are configurable via two 16-bit registers, PIOMODE and PIODIRECTION. The settings are as follows:

MODE	PIOMODE reg.	PIODIRECTION reg.	PIN FUNCTION
0	0	0	Normal operation
1	0	1	INPUT with pull-up/pull-down
2	1	0	OUTPUT
3	1	1	INPUT without pull-up/pull-down

SmartLCD initialization on PIO pins in **ae_init()** is listed below:

```
outport(0xff78,0xe73c);     // PDIR1, TxD0, RxD0, TxD1, RxD1, P16=PCS0, P17=PCS1=PPI
outport(0xff76,0x0000);     // PIOM1
outport(0xff72,0xec7b);     // PDIR0, P12,A19,A18,A17,P2=PCS6=RTC
outport(0xff70,0x1000);     // PIOM0, P12=LED
```

The C function in the library **ae_lib** can be used to initial PIO pins.

void ***pio_init***(char bit, char mode);

Where bit = 0-31 and mode = 0-3, see the table above.

Example:

pio_init(12, 2); will set P12 as output
pio_init(1, 0); will set P1 as Timer1 output

void ***pio_wr***(char bit, char dat);
pio_wr(12,1); set P12 pin high, if P12 is in output mode
pio_wr(12,0); set P12 pin low, if P12 is in output mode
unsigned int ***pio_rd***(char port);
pio_rd (0); return 16-bit status of P0-P15, if corresponding pin is in input mode,
pio_rd (1); return 16-bit status of P16-P31, if corresponding pin is in input mode,

Most of the I/O lines are used by the SmartLCD system for on-board components (Table 3.2). We suggest that you not use these lines unless you are sure that you are not interfering with the operation of such components (i.e., if the component is not installed).

You should also note that the external interrupt PIO pins INT2, 4, 5, and 6 are not available for use as output because of the inverters attached. The input values of these PIO interrupt lines will also be inverted for the same reason. As a result, calling ***pio_rd*** to read the value of P31 (**INT2**) will return 1 when pin 19 on header J2 is pulled low, with the result reversed if the pin is pulled high.

Signal	Pin	Function
P2	/PCS6	U4 RTC72423 chip select at base I/O address 0x0600
P3	/PCS5	U8 SCC2691 UART chip select at base I/O address 0x0500
P4	/DT	STEP 2 jumper
P11	Timer0 input	Shared with U10 TLC2543 ADC and U7 24C04 EE data input
P12	DRQ0/INT5	Output for LED, CLK for U7 EE, U11 DAC, U23 DAC, Hit watchdog
P16	/PCS0	PAL16CEV8 (U22) chip select
P17	/PCS1	U5 PPI 82C55 chip select at base I/O address 0x0100
P18	/PCS2=/CTS1	J11 pin 19, Second PCMCIA MemCard interface
P22	TxD0	Default SER0 debug
P23	RxD0	Default SER0 debug
INT0	J2 pin 2	U8 SCC2691 UART interrupt.

Table 3.2 188 CPU PIO lines used for on-board components

3.4 I/O Mapped Devices

3.4.1 I/O Space

External I/O devices can use I/O mapping for access. You can access such I/O devices with ***inportb***(port) or ***outportb***(port,dat). These functions will transfer one byte or word of data to the specified I/O address. The external I/O space is 64K, ranging from 0x0000 to 0xffff.

The default I/O access time is 15 wait states. You may use the function void ***io_wait***(char wait) to define the I/O wait states from 0 to 15. The system clock is 25 ns, giving a clock speed of 40 MHz. Details regarding this can be found in the Software chapter, and in the 188 CPU User's Manual. Slower components, such as most LCD interfaces, might find the maximum programmable wait state of 15 cycles still insufficient. Due to the high bus speed of the system, some components need to be attached to I/O pins directly.

For details regarding the chip select unit, please see Chapter 5 of the 188 CPU User's Manual.

The table below shows more information about I/O mapping.

I/O space	Select	Location	Usage
0x0000-0x00ff	/PCS0	U17 pin 14=P16	PAL16V8CE (U22), J1, SED1335 (U15), PPI (U20), PPI1 (U27)
0x0100-0x01ff	/PCS1	U5 pin 7=P17	PPI (U5), 82C55
0x0200-0x02ff	/PCS2	J11 pin 19=CTS1	Second PCMCIA interface
0x0300-0x03ff	/PCS3	J2 pin 31 = P19	User
0x0400-0x04ff	/PCS4		Reserved
0x0500-0x05ff	/PCS5	J2 pin 15=P3	UART, SCC2691
0x0600-0x06ff	/PCS6	J2 pin 24=P2	RTC72423

3.4.2 SED1335

The SED1335, from SMOS Systems, Inc., is a versatile LCD controller that can display text and graphics on a 320x240-pixel LCD panel. The SED1335 communicates with the host 188 CPU via 8-bit high-speed data bus. The SED1335 can display layered text and graphics, scroll the display in any direction and partition the display into multiple screens. The SED1335 stores text, character codes, and bit-mapped graphics data in the external image frame buffer SRAM. Display functions include transferring data from the host microprocessor to the image buffer, reading image data, converting data to display pixels and generating timing signals for the buffer memory and the LCD panel. The SED1335 has an internal character generator with 160 5x7 pixel characters.

For complete, detailed information on the hardware and software of the SED1335, users may contact SMOS Systems, Inc., at telephone number 408-922-0200, or through their website at http://www.smos.com.

3.4.3 Programmable Peripheral Interface (82C55A)

U5, U20, and U27 PPIs (82C55) are low-power CMOS programmable parallel interface units for use in microcomputer systems. They each provide 24 I/O pins that may be individually programmed in two groups of 12 and used in three major modes of operation.

In MODE 0, the two groups of 12 pins can be programmed in sets of 4 and 8 pins to be inputs or outputs. In MODE 1, each of the two groups of 12 pins can be programmed to have 8 lines of input or output. Of the four remaining pins, three are used for handshaking and interrupt control signals. MODE 2 is a strobed bi-directional bus configuration.

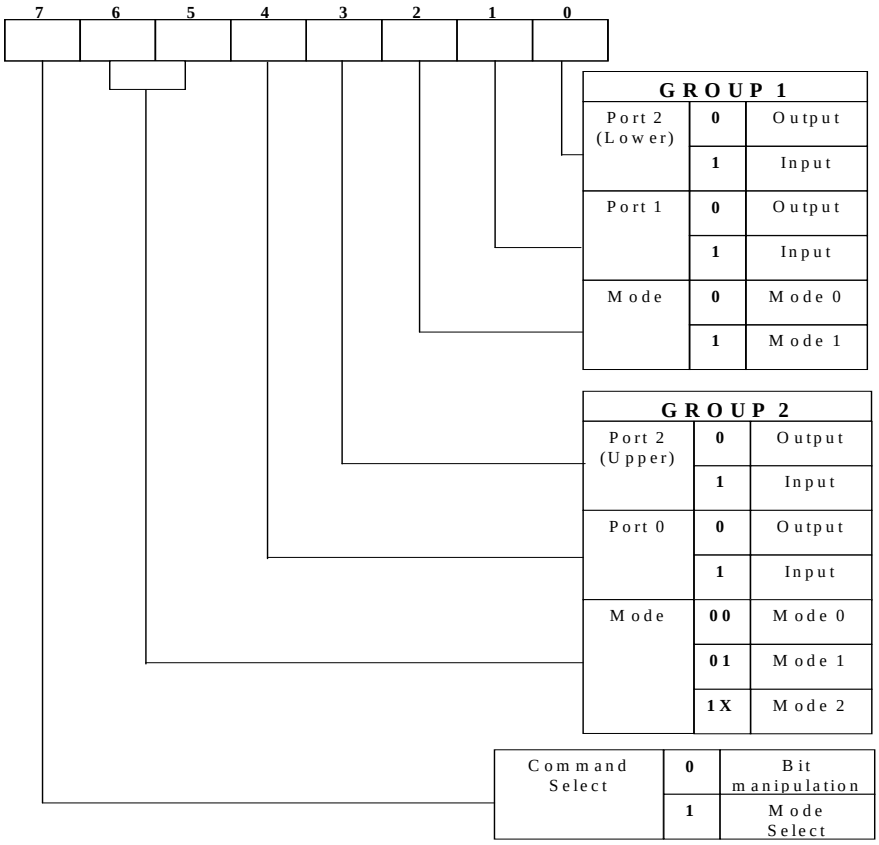


Figure 3.2 Mode Select Command Word

The SmartLCD maps U5, the 82C55/uPD71055, at base I/O address 0x0100.

The SmartLCD maps U20, the 82C55/uPD71055, at base I/O address 0x0050.

The SmartLCD maps U27, the 82C55/uPD71055, at base I/O address 0x70.

To use U5 PPI as a program example, the Command Register = 0x0103; Port 0 = 0x0100; Port 1 = 0x0101; and Port 2 = 0x0102.

The following code example will set all ports to output mode:

```
outportb(0x0103,0x80); /* Mode 0 all output selection. */
outportb(0x0100,0x55); /* Sets port 0 to alternating high/low I/O pins. */
outportb(0x0101,0x55); /* Sets port 1 to alternating high/low I/O pins. */
outportb(0x0102,0x55); /* Sets port 2 to alternating high/low I/O pins. */
```

To set all ports to input mode:

```
outportb(0x0103,0x9f);     /* Mode 0 all input selection. */
```

You may read the ports with:

```
inportb(0x0100); /* Port 0 */
inportb(0x0101); /* Port 1 */
inportb(0x0102); /* Port 2 */
```

This returns an 8-bit value for each port, with each bit corresponding to the appropriate line on the port.

You will find that numerous on-board components are controlled using PPI lines only. You will need to use PPI access methods to control these, as well.

3.4.4 Real-time Clock RTC72423

If installed, the real-time clock RTC72423 (EPSON, U4) is mapped in the I/O address space 0x0600. It must be backed up with a lithium coin battery. The RTC is accessed via software drivers ***rtc_init()*** or ***rtc_rd()*** (see Appendix C and the Software chapter for details).

It is also possible to configure the real-time clock to raise an output line attached to an external interrupt, at 1/64 second, 1 second, 1 minute, or 1 hour intervals. This can be used in a time-driven application, or the **VOFF** signal can be used to turn on/off the controller using an external switching power supply. An example of a program showing a similar application can be found in ***tern\186\samples\ae\poweroff.c***.

3.4.5 UART SCC2691

The UART SCC2691 (Signetics, U8) is mapped into the I/O address space at 0x0500. The SCC2691 has a full-duplex asynchronous receiver/transmitter, a quadruple buffered receiver data register, an interrupt control mechanism, programmable data format, selectable baud rate for the receiver and transmitter, a multi-functional and programmable 16-bit counter/timer, an on-chip crystal oscillator, and a multi-purpose input/output including RTS and CTS mechanism.

For more information, refer to Appendix B. The SCC2691 on the SmartLCD may be used as a network 9-bit UART (for the TERN NT-Kit).

3.5 Other Devices

A number of other devices are also available on the SmartLCD. Some of these are optional, and might not be installed on the particular controller you are using. For a discussion regarding the software interface for these components, please see the Software chapter.

3.5.1 On-board Supervisor with Watchdog Timer

The MAX691/LTC691 (U6) is a supervisor chip. With it installed, the SmartLCD has several functions: watchdog timer, battery backup, power-on-reset delay, power-supply monitoring, and power-failure warning. These will significantly improve system reliability.

Watchdog Timer

The watchdog timer is activated by setting a jumper on J9 of the SmartLCD. The watchdog timer provides a means of verifying proper software execution. In the user's application program, calls to the function **hitwd()** (a routine that toggles the P12=HWD pin of the MAX691) should be arranged such that the HWD pin is accessed at least once every 1.6 seconds. If the J9 jumper is on and the HWD pin is not accessed within this time-out period, the watchdog timer pulls the WDO pin low, which asserts /RESET. This automatic assertion of /RESET may recover the application program if something is wrong. After the SmartLCD is reset, the WDO remains low until a transition occurs at the WDI pin of the MAX691. When controllers are shipped from the factory the J9 jumper is off, which disables the watchdog timer.

The 188 CPU has an internal watchdog timer. This is disabled by default with **ae_init()**.

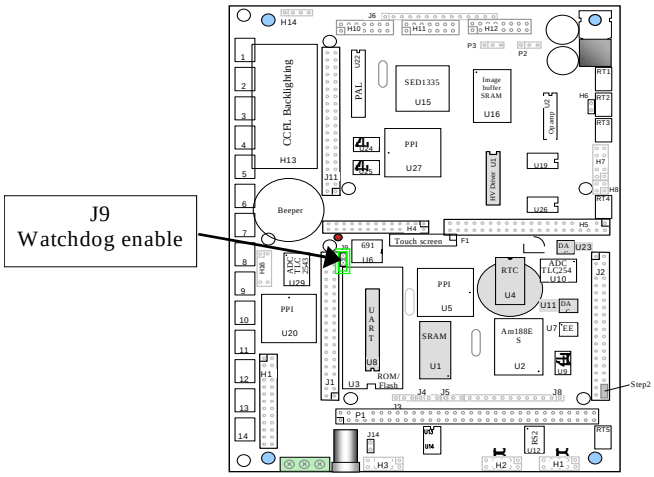


Figure 3.3 Location of watchdog timer enable jumper

Battery Backup Protection

The backup battery protection protects data stored in the SRAM and RTC. The battery-switch-over circuit compares VCC to VBAT (+3 V lithium battery positive pin), and connects whichever is higher to the VRAM (power for SRAM and RTC). Thus, the SRAM and the real-time clock RTC72423 are backed up. In normal use, the lithium battery should last approximately 3-5 years without external power being supplied. When the external power is on, the battery-switch-over circuit will select the VCC to connect to the VRAM.

3.5.2 EEPROM

A serial EEPROM of 512 bytes (24C04), or 2K bytes (24C16) by special request, can be installed in U7. The SmartLCD uses the P12=SCL (serial clock) and P11=SDA (serial data) to interface with the EEPROM. The EEPROM can be used to store important data such as a node address, calibration coefficients, and configuration codes. It typically has 1,000,000 erase/write cycles. The data retention is more than 40 years. EEPROM can be read and written by simply calling functions the **ee_rd()** and **ee_wr()**.

The EEPROM and the 12-bit ADC (U10) share the same data input signal line, P11. The 12-bit ADC uses I20 (PPI U5) line as chip select. If the I20 line is low, the ADC will be enabled and holds the P11 data line, prohibiting EEPROM operation. The ae_init(); function sets I20 high. The ae_ad12(); function brings I20 low only when it needs to. The user should be aware that they must always keep I20 high in order to disable the ADC and free P11 line for the EEPROM. Any outportb(0x103, ??)l to access the PPI mode register will set I20 low. The user should use outportb(0x102, 0x01 | inportb(0x102)); to bring I20 high.

A range of lower addresses in the EEPROM is reserved for TERN use. Details regarding which addresses are reserved, and for what purpose, can be found in Appendix D of this manual.

3.6 Inputs and Outputs

3.6.1 12-bit ADC (TLC2543)

The TLC2543 is a 12-bit, switched-capacitor, successive-approximation, 11 channels, serial interface, analog-to-digital converter.

Two 12-bit ADC chips, U10 and U29, can be installed on the SmartLCD.

For the U10 ADC:

Three PPI I/O lines are used, with /CS=I20; CLK=I22; and DIN=I21.

The U10 ADC digital data output communicates with a host through a serial tri-state output (DOUT=P11). If I20=/CS is low, the U10 TLC2543 will have output on P11. If I20=/CS is high, the U10 TLC2543 is disabled and P11 is free. I20 and P11 are pulled high by 10K resistors on board.

For the U29 ADC:

Three PPI I/O lines are used, with /CS=T17; CLK=I22; and DIN=I21.

The U29 ADC digital data output communicates with a host through a U27 PPI input line T00 (DOUT=I0). If T17=/CS is low, the U29 TLC2543 will have output on I0 and T00. If T17=/CS is high, the U29 TLC2543 is disabled and I0 is pulled high by 10K resistors on board. I0 is the Schmitt-trigger inverter input and T00 is the Schmitt-trigger inverter output.

The TLC2543 has an on-chip 14-channel multiplexer that can select any one of 11 inputs or any one of three internal self-test voltages. The sample-and-hold function is automatic. At the end of conversion, the end-of-conversion (EOC) output is not connected, although it goes high to indicate that conversion is complete.

TLC2543 features differential high-impedance inputs that facilitate ratiometric conversion, scaling, and isolation of analog circuitry from logic and supply noise. A switched-capacitor design allows low-error conversion over the full operating temperature range. The analog input signal source impedance should be less than 50Ω and capable of slewing the analog input voltage into a 60 pf capacitor.

A reference voltage less than VCC (+5V) can be provided for the TLC2543 if additional precision is required. A voltage above 2.5V and less than +5V can be used for this purpose, and can be connected to the **REF+** pin.

The CLK signal to the ADC is toggled through an I/O pin, and serial access allows a conversion rate of up to approximately 10 KHz, for a 40 MHz CPU.

In order to operate the U10 and U29 TLC2543, I/O lines are used, as listed below:

/CS	Chip select = I20 for U10 and T17 for U29 , high to low transition enables DOUT, DIN and CLK. Low to high transition disables DOUT, DIN and CLK.
DIN	PPI I21 for both U10 and U29, serial data input
DOUT	U10 use P11 of 188 CPU, U29 use I0/T00.
EOC	Not Connected, End of Conversion, high indicates conversion complete and data is ready
CLK	I/O clock = PPI I22
REF+	Upper reference voltage (normally VCC)
REF-	Lower reference voltage (normally GND)
VCC	Power supply, +5 V input
GND	Ground

The analog inputs AD0 to AD10 are available at H5 header. AD11 to AD20 are routed to H16. The reference of U29 is fixed to 5V. The reference for U10 is available at H5 pin 17.

AD10 is connected to the operational amplifier U28 output to monitor the input power voltage (divided by 5).

3.6.2 Dual 12-bit DAC

The LTC1446/LTC1446L is a dual 12-bit digital-to-analog converter (DAC) in an SO-8 package. It is complete with a rail-to-rail voltage output amplifier, an internal reference and a 3-wire serial interface. The LTC1446 outputs a full-scale of 4.096V, making 1 LSB equal to 1 mV. The LTC1446L outputs a full-scale of 2.5 V, making 1 LSB equal to 0.61 mV.

The buffered outputs can source or sink 5 mA. The outputs swing to within a few millivolts of supply rail when unloaded. They have an equivalent output resistance of 40 Ω when driving a load to the rails. The buffer amplifiers can drive 1000 pf without going into oscillation.

The DAC is installed in U11 and/or U23 on the SmartLCD, and the outputs are routed to H5.

One DAC output (VA) is used to control the LCD contrast voltage. The U28 LM324 operational amplifier is used to convert the VA signal to negative LCD contrast adjustable voltage.

The DAC uses P12 as CLK, P26 as DI, and P29 (U11) or I23 (U23) as LD/CS. Note that P26 and P29 are also used by the high voltage driver U19. Writing to the DAC will cause these two high voltage outputs to toggle. Please refer to the LT1446 technical data sheets from Linear Technology (1-408-432-1900) for more information. See also the sample program ***ae_da.c*** in the **`\samples\ae`** directory.

3.6.3 High-Voltage, High-Current Drivers

ULN2003 has high voltage, high current Darlington transistor arrays, consisting of seven silicon NPN Darlington pairs on a common monolithic substrate. All channels feature open-collector outputs for sinking 350 mA at 50V, and integral protection diodes for driving inductive loads. Peak inrush currents of up to 600 mA sinking are allowed. U18 has seven high-voltage drivers (HV1-HV7). These outputs may be paralleled to achieve high-load capability, although each driver has a maximum continuous collector current rating of 350 mA at 50V. The maximum power dissipation allowed is 2.20 W per chip at 25 degrees C (°C). The common substrate G is routed to H7 GND pins. All currents sinking in must return to the H7 GND pin. A heavy gauge (20) wire must be used to connect the H7 GND terminal to an external common ground return. K connects to the protection diodes in the ULN2003 chips and should be tied to highest voltage in the external load system. K can be connected to an unregulated on board +12V via H7 pin 8. **ULN2003 is a *sinking* driver, not a sourcing driver.** An example of typical application wiring is shown in Figure 3.4.

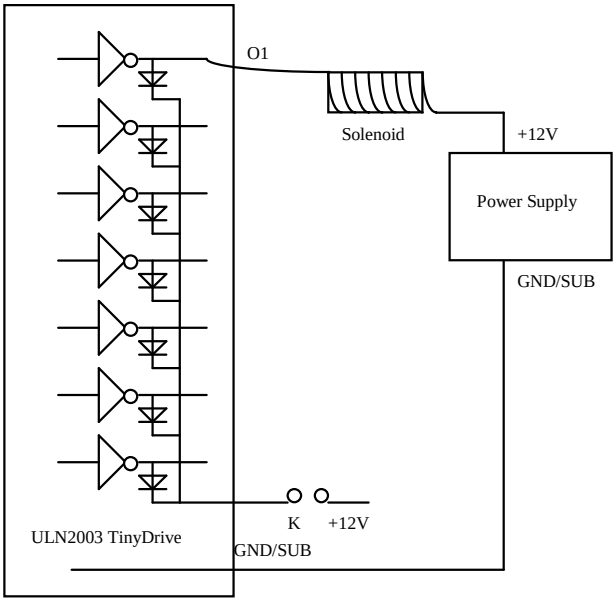


Figure 3.4 Drive inductive load with high voltage/current drivers.

The SmartLCD uses HV5 to control the negative power source and uses HV6 and HV4 to control the CCFL inverter.

3.6.4 CCFL Inverter

A Cold Cathode Fluorescent Lighting (CCFL) plate is inserted behind the LCD. It is a replaceable backlighting device. A CCFL inverter (H13) is on the SmartLCD to provide a modulated high voltage in order to maintain constant current in the CCFL lamp. The CCFL lamp connector should plug into H14 with the white wire pointing to H14 pin 1. The CCFL inverter is powered by 12V DC and is switched on and off by the on-board solenoid drivers HV6 and HV4. The software driver C function `ccfl(1);` can power-on the CCFL, and `ccfl(0);` can turn it off in order to save power and lengthen the life of the lamp and LCD. These functions are located in the **`c:\tern\186\samples\sl`** directory.

3.6.5 Negative 24V Power Supply and adjustable contrast voltage

The graphic LCD requires –24V power supply and approximately –17V adjustable negative voltage for contrast adjustment of the display. In order to reduce power consumption, the negative power supply can be turned off by solenoid driver HV5 with the software function `neg_24(0);`. The function `neg_24(1);` can be used to turn it on.

A 12-bit DAC channel, VA, is dedicated to drive the contrast voltage by using the software function `contrast(int dat);`.

These functions are located in the **`c:\tern\186\samples\sl`** directory.

3.6.6 Touchscreen and Mechanical Pushbuttons

A transparent 10x7 keypad, or touchscreen, is installed in front of the LCD. The user can see the LCD clearly through the touchscreen. The flex cable of the touchscreen connects to the F1 header. There are two Schmitt Trigger inverters, U24 and U25, to reduce the touchscreen key input noise.

The U27 PPI I/O pins are mainly used to scan the touchscreen. The 70 touchscreen keys can be scanned and return keycodes by using I/O pins via the software function `scan_touch_screen();`.

In addition to the 70 touchscreen keys, 14 mechanical push buttons can be installed on the SmartLCD. You may use the same function, `scan_touch_screen();`, to return the push-button status. For more details, see the samples in

`c:\tern\186\samples\sl\sl_grid.c`

The following are HEX values for the 14 push buttons located at the lower edge of the SmartLCD:

BUTTON	1	2	3	4	5	6	7	8	9	10	11	12	13	14
HEX	B0	C0	B1	C1	B6	B7	B2	C2	B3	C3	C4	B4	B5	C5

Five additional right-angle push buttons are located on the top edge of the board:

RT1 = VOFF
RT2 = /RESET
RT3 = NC
RT4 = /INT1
RT5 = /INT2=P31

3.7 Headers and Connectors

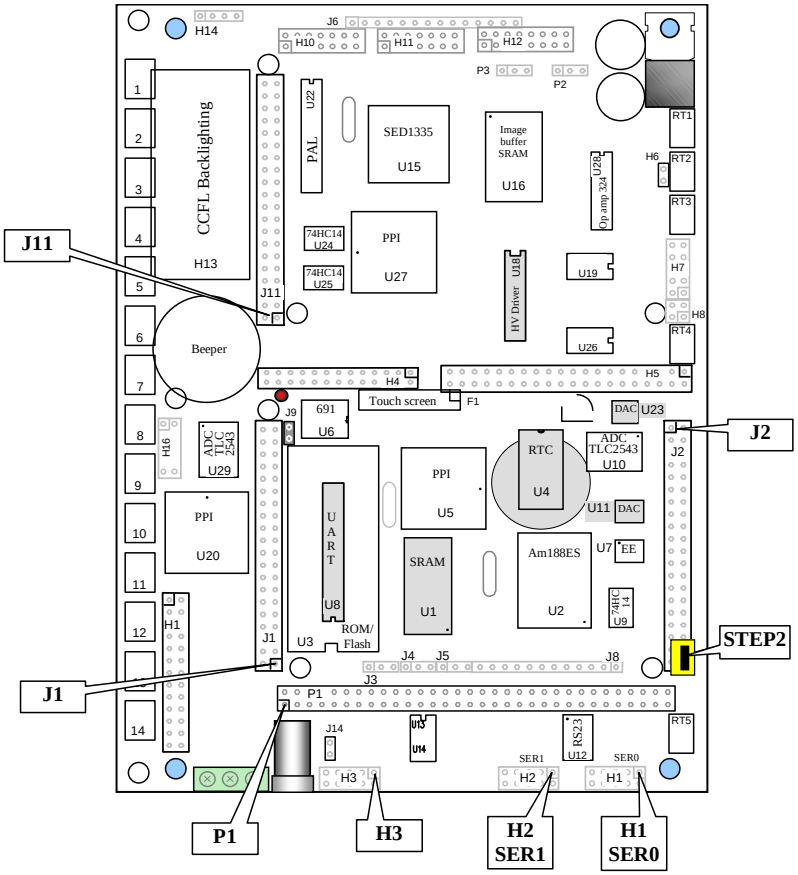


Figure 3.5 SmartLCD Headers and Connectors.

3.7.2 Expansion Ports

The pin layouts of the headers on the SmartLCD are listed below.

J1 Signals			
VCC	1	2	GND
MPO	3	4	P1
RxD	5	6	GND
TxD	7	8	D0
VOFF	9	10	D1
PFI	11	12	D2
GND	13	14	D3
/RST	15	16	D4
RST	17	18	D5
P16	19	20	D6
MPI	21	22	D7
CLK	23	24	GND
HLDA	25	26	A7
HOLD	27	28	A6
/WR	29	30	A5
/RD	31	32	A4
VRAM	33	34	A3
VBAT	35	36	A2
GND	37	38	A1
VCC	39	40	A0

J2 Signals			
GND *(Step 2 Jumper)*	40	39	VCC
P4 *(Step 2 Jumper)*	38	37	P14
/CTS0	36	35	P6
TxD0	34	33	/INT4
RxD0	32	31	/RTS1
P5	30	29	P1
TxD1	28	27	/RTS0
RxD1	26	25	GND
P2	24	23	P15
/CTS1	22	21	/INT3
P0	20	19	/INT2
P25	18	17	P24
/WR	16	15	P3
P11	14	13	P17
P10	12	11	P13
VCC	10	9	NC
/INT0	8	7	/NMI
/INT1	6	5	P12
P26	4	3	P29
GND	2	1	U11 12-bit DAC VB

Signal definitions for J1:

VCC	+5V power supply
GND	Ground
CLK	188 CPU pin 16, system clock, 40 MHz (25 ns) as default
RxD	data receive of UART SCC2691, U8
TxD	data transmit of UART SCC2691, U8
MPO	Multi-Purpose Output of SCC2691, U8
MPI	Multi-Purpose Input of SCC2691, U8
VOFF	real-time clock output of RTC72423 U4, open collector
D0-D7	188 CPU 8-bit external data lines
A0-A7	188 CPU address lines
PFI	power failure input signal of MAX691
/RST	reset signal, active low
RST	reset signal, active high
P16	/PCS0, 188 CPU pin 66
HLDA	188 CPU pin 44
HOLD	188 CPU pin 45
/WR	188 CPU pin 5
/RD	188 CPU pin 6
VBAT	+3V lithium battery positive pin
VRAM	power for backing up SRAM and RTC

Signal definitions for J2:

VCC	+5V power supply, < 200 mA
GND	Ground
Pxx	188 CPU PIO pins
/WR	188 CPU pin 5
TxD0	188 CPU pin 2, transmit data of serial channel 0
RxD0	188 CPU pin 1, receive data of serial channel 0
TxD1	188 CPU pin 98, transmit data of serial channel 1
RxD1	188 CPU pin 99, receive data of serial channel 1
/CTS0	188 CPU pin 100, Clear-to-Send signal for SER0
/CTS1	188 CPU pin 63, Clear-to-Send signal for SER1
/RTS0	188 CPU pin 3, Request-to-Send signal for SER0
/RTS1	188 CPU pin 62, Request-to-Send signal for SER1
/INT0-4	Schmitt-trigger inputs

H1
SER0 SERIAL DEBUG PORT, RS-232

H2
SER1 SERIAL PORT, RS-232

H3
SCC2691 UART PORT, RS-232 or RS-485

3.7.3 Jumpers and Headers

The following table lists the jumpers and connectors on the SmartLCD.

Name	Size	Function	Possible Configuration
F1		Touchscreen	
H1	5x2	SER0, RS-232	
H2	5x2	SER1, RS-232	
H3	5x2	SCC2691, RS-232/485	
H4	13x2	PPI U27, T00-T27	
H5	20x2	U10 ADC, U11 & U23 DAC, U5 PPI	
H6	2x1	RESET	
H7	5x2	7 HV drivers	
H8	2x2	GND, VCC, P25	
H10	7x2	LCD Sharp LM32K071	
H11	7x2	LCD Hantronix HDM3224TS-1	
H12	8x2	LCD Pixtech FE524M1	
H13		CCFL converter CXA-L10L (TDK)	
H14	4x1	CCFL-LCD	
H15	13x2	U20 PPI L00-L27	
H16	5x2	U29 ADC	

Name	Size	Function	Possible Configuration
J1	20x2	Expansion header, MemCard1 interface	
J2	20x2	Expansion header	*NOTE:* pin 38-40: Step 2 jumper (if not installed, controller runs in Step 1 mode)
J3	3x1	SRAM selection:	pin 2-3: SRAM 256KB-512KB pin 1-2: SRAM 32KB-128KB
J4	3x1	ROM/Flash size selection:	pin 1-2: ROM or Flash size 32KB-128KB pin 2-3: ROM or Flash size 256KB-512KB
J5	3x1	ROM 512KB selection:	pin 1-2: ROM size 512KB pin 2-3: Flash 128KB-512KB, or ROM <512 KB
J6	14x1	LCD Hantronix HDM3224TS-1	
J8	12x1	High address lines, A8-A19	
J9	2x1	Watchdog timer	Enabled if Jumper is on Disabled if jumper is off
J11	20x2	MemCard2 interface	
J14	2x1	External battery power input: VBAT, GND	
P1	32x2	PC/104	

Chapter 4: Software

Please refer to the Technical Manual of the "C/C++ Development Kit for TERN 16-bit Embedded Microcontrollers" for details on debugging and programming tools.

For details regarding software function prototypes and sample files demonstrating their use, please refer to the Software Glossary in Appendix E.

Guidelines, awareness, and problems in an interrupt driven environment
Although the C/C++ Development Kit provides a simple, low cost solution to application engineers, some guidelines must be followed. If they are not followed, you may experience system crashes, PC hang-ups, and other problems.

The debugging of interrupt handlers with the Remote Debugger can be a challenge. It is possible to debug an interrupt handler, but there is a risk of experiencing problems. Most problems occur in multi-interrupt-driven situations. Because the remote kernel running on the controller is interrupt-driven, it demands interrupt services from the CPU. If an application program enables interrupt and occupies the interrupt controller for longer than the remote debugger can accept, the debugger will time-out. As a result, your PC may hang-up. In extreme cases, a power reset may be required to restart your PC.

For your reference, be aware that our system is remote kernel interrupt-driven for debugging.

The run-time environment on TERN controllers consists of an I/O address space and a memory address space. I/O address space ranges from **0x0000** to **0xffff**, or 64 KB. Memory address space ranges from **0x00000** to **0xfffff** in real-mode, or 1 MB. These are accessed differently, and not all addresses can be translated and handled correctly by hardware. I/O and memory mappings are done in software to define how translations are implemented by the hardware. Implicit accesses to I/O and memory address space occur throughout your program from TERN libraries as well as simple memory accesses to either code or global and stack data. You can, however, explicitly access any address in I/O or memory space, and you will probably need to do so in order to access processor registers and on-board peripheral components (which often reside in I/O space) or non-mapped memory.

This is done with four different sets of similar functions, described below.

poke/pokeb
Arguments: unsigned int segment, unsigned int offset, unsigned int/unsigned char data
Return value: none

These standard C functions are used to place specified data at any memory space location. The **segment** argument is left shifted by four and added to the **offset** argument to indicate the 20-bit address within memory space. **poke** is used for writing 16 bits at a time, and **pokeb** is used for writing 8 bits.

The process of placing data into memory space means that the appropriate address and data are placed on the address and data-bus, and any memory-space mappings in place for this particular range of memory will be used to activate appropriate chip-select lines and the corresponding hardware component responsible for handling this data.

peek/peekb
Arguments: unsigned int segment, unsigned int offset
Return value: unsigned int/unsigned char data

These functions retrieve the data for a specified address in memory space. Once again, the **segment** address is shifted left by four bits and added to the **offset** to find the 20-bit address. This address is then output over the address bus, and the hardware component mapped to that address should return either an 8-bit or 16-bit value over the data bus. If there is no component mapped to that address, this function will return random garbage values every time you try to peek into that address.

outport/outportb
Arguments: unsigned int address, unsigned int/unsigned char data
Return value: none

This function is used to place the **data** into the appropriate **address** in I/O space. It is used most often when working with processor registers that are mapped into I/O space and must be accessed using either one of these functions. This is also the function used in most cases when dealing with user-configured peripheral components.

When dealing with processor registers, be sure to use the correct function. Use **outport** if you are dealing with a 16-bit register.

inport/inportb
Arguments: unsigned int address
Return value: unsigned int/unsigned char data

This function can be used to retrieve data from components in I/O space. You will find that most hardware options added to TERN controllers are mapped into I/O space, since memory space is valuable and is reserved for uses related to the code and data. Using I/O mappings, the address is output over the address bus, and the returned 16 or 8-bit value is the return value.

For a further discussion of I/O and memory mappings, please refer to the Hardware chapter of this technical manual.

4.1 AE.LIB

AE.LIB is a C library for basic SmartLCD operations. It includes the following modules: AE.OBJ, SER0.OBJ, SER1.OBJ, SCC.OBJ, and AEEE.OBJ. You need to link AE.LIB in your applications and include the corresponding header files. The following is a list of the header files:

Include-file name	Description
AE.H	PPI, timer/counter, ADC, DAC, RTC, Watchdog,
SER0.H	Internal serial port 0
SER1.H	Internal serial port 1
SCC.H	External UART SCC2691
AEEE.H	on-board EEPROM

4.2 Functions in AE.OBJ

4.2.1 SmartLCD Initialization

ae_init

This function should be called at the beginning of every program running on SmartLCD core controllers. It provides default initialization and configuration of the various I/O pins, interrupt vectors, sets up expanded DOS I/O, and provides other processor-specific updates needed at the beginning of every program.

There are certain default pin modes and interrupt settings you might wish to change. With that in mind, the basic effects of **ae_init** are described below. For details regarding register use, you will want to refer to the 188 CPU Microcontroller User's manual in the Amd_docs directory.

Initialize the upper chip select to support the default ROM. The CPU registers are configured such that:

Address space for the ROM is from 0x80000-0xfffff (to map MemCard I/O window)
512K ROM Block size operation.
Three wait state operation (allowing it to support up to 120 ns ROMs). With 70 ns ROMs, this can actually be set to zero wait state if you require increased performance (at a risk of stability in noisy environments). For details, see the UMCS (Upper Memory Chip Select Register) reference in the processor User's manual.

```
outport(0xffa0, 0x80bf); // UMCS, 512K ROM, 0x80000-0xfffff
```

Initialize LCS (*Lower Chip Select*) for use with the SRAM. It is configured so that:

Address space starts 0x00000, with a maximum of 512K RAM.
Three wait state operation. Reducing this value can improve performance.
Disables PSRAM, and disables need for external ready.

```
outport(0xffa2, 0x7fbf); // LMCS, base Mem address 0x0000
```

Initialize MMCS and MPCS so that **MCS0** and **PCS0**-**PCS6** (except for PCS4) are configured so:

MCS0 is mapped also to a 256K window at 0x80000. If used with MemCard, this chip select line is used for the I/O window.
Sets up **PCS5-6** lines as chip-select lines, with three wait state operation.

```
outport(0xffa8, 0xa0bf); // s8, 3 wait states
outport(0xffa6, 0x81ff); // CS0MSKH
```

Initialize PACS so that **PCS0**-**PCS3** are configured so that:

Sets up **PCS0**-**3** lines as chip-select lines, with fifteen wait state operation.
The chip select lines starts at I/O address 0x0000, with each successive chip select line addressed 0x100 higher in I/O space.

```
outport(0xffa4, 0x007f); // CS0MSKL, 512K, enable CS0 for RAM
```

Configure the two PIO ports for default operation. All pins are set up as default input, except for P12 (used for driving the LED), and peripheral function pins for SER0 and SER1, as well as chip selects for the PPI.

```
outport(0xff78,0xe73c);     //   PDIR1,   TxD0,   RxD0,   TxD1,   RxD1,
                            // P16=PCS0, P17=PCS1=PPI
 outport(0xff76,0x0000);    // PIOM1
 outport(0xff72,0xec7b);    // PDIR0, P12,A19,A18,A17,P2=PCS6=RTC
 outport(0xff70,0x1000);    // PIOM0, P12=LED
```

Configure the PPI 82C55 to all inputs, except for lines I20-23 which are used as output for the ADC. You can reset these to inputs if not being used for that function.

```
outportb(0x0103,0x9a);     // all pins are input, I20-23 output
outportb(0x0100,0);
outportb(0x0101,0);
```

```
outportb(0x0102,0x01);     // I20=ADCS high
```

The chip select lines are by default set to 15 wait state. This makes it possible to interface with many slower external peripheral components. If you require faster I/O access, you can modify this number down as needed. Some TERN components, such as the Real-Time-Clock, might fail if the wait state is decreased too dramatically. A function is provided for this purpose.

void io_wait
Arguments: char wait
Return value: none.

This function sets the current wait state depending on the argument ***wait***.

```
wait=0, wait states = 0, I/O enable for 100 ns
wait=1, wait states = 1, I/O enable for 100+25 ns
wait=2, wait states = 2, I/O enable for 100+50 ns
wait=3, wait states = 3, I/O enable for 100+75 ns
wait=4, wait states = 5, I/O enable for 100+125 ns
wait=5, wait states = 7, I/O enable for 100+175 ns
wait=6, wait states = 9, I/O enable for 100+225 ns
wait=7, wait states = 15, I/O enable for 100+375 ns
```

4.2.2 External Interrupt Initialization

There are up to eight external interrupt sources on the SmartLCD, consisting of seven maskable interrupt pins (**INT6-INT0**) and one non-maskable interrupt (**NMI**). There are also an additional eight internal interrupt sources not connected to the external pins, consisting of three timers, two DMA channels, both asynchronous serial ports, and the **NMI** from the watchdog timer. For a detailed discussion involving the ICUs, the user should refer to Chapter 7 of the AMD 188 CPU Microcontroller User's Manual.

TERN provides functions to enable/disable all of the eight external interrupts. The user can call any of the interrupt init functions listed below for this purpose. The first argument indicates whether the particular interrupt should be enabled, and the second is a function pointer to an appropriate interrupt service routine that should be used to handle the interrupt. The TERN libraries will set up the interrupt vectors correctly for the specified external interrupt line.

At the end of interrupt handlers, the appropriate in-service bit for the IR signal currently being handled must be cleared. This can be done using the **Nonspecific EOI command**. At initialization time, interrupt priority was placed in **Fully Nested** mode. This means the current highest priority interrupt will be handled first, and a higher priority interrupt will interrupt any current interrupt handlers. So, if the user chooses to clear the in-service bit for the interrupt currently being handled, the interrupt service routine just needs to issue the nonspecific EOI command to clear the current highest priority IR.

To send the nonspecific EOI command, you need to write the **EOI** register word with 0x8000.

```
outport(0xff22, 0x8000);
```

void int*x*_init
Arguments: unsigned char i, void interrupt far(* int*x*_isr) ())
Return value: none

These functions can be used to initialize any one of the external interrupt channels (for pin locations and other physical hardware details, see the Hardware chapter). The first argument **i** indicates whether this particular interrupt should be enabled or disabled. The second argument is a function pointer which will act as the interrupt service routine. The overhead on the interrupt service routine, when executed, is about 20 µs.

By default, the interrupts are all disabled after initialization. To disable them again, you can repeat the call but pass in 0 as the first argument.

The NMI (Non-Maskable Interrupt) is special in that it can not be masked (disabled). The default ISR will return on interrupt.

```
void int0_init( unsigned char i, void interrupt far(* int0_isr)() );
void int1_init( unsigned char i, void interrupt far(* int1_isr)() );
void int2_init( unsigned char i, void interrupt far(* int2_isr)() );
void int3_init( unsigned char i, void interrupt far(* int3_isr)() );
void int4_init( unsigned char i, void interrupt far(* int4_isr)() );
void int5_init( unsigned char i, void interrupt far(* int5_isr)() );
void int6_init( unsigned char i, void interrupt far(* int6_isr)() );
void int7_init( unsigned char i, void interrupt far(* int7_isr)() );
void int8_init( unsigned char i, void interrupt far(* int8_isr)() );
void int9_init( unsigned char i, void interrupt far(* int9_isr)() );
void nmi_init(void interrupt far (* nmi_isr)());
```

4.2.3 I/O Initialization

Two ports of 16 I/O pins each are available on the SmartLCD. Hardware details regarding these PIO lines can be found in the Hardware chapter.

Several functions are provided for access to the PIO lines. At the beginning of any application where you choose to use the PIO pins as input/output, you will probably need to initialize these pins in one of the four available modes. Before selecting pins for this purpose, make sure that the peripheral mode operation of the pin is not needed for a different use within the same application.

You should also confirm the PIO usage that is described above within **ae_init()**. During initialization, several lines are reserved for TERN usage and you should understand that these are not available for your application. There are several PIO lines that are used for other on-board purposes. These are all described in some detail in the Hardware chapter of this technical manual. For a detailed discussion toward the I/O ports, please refer to Chapter 11 of the AMD 188 CPU User's Manual.

Please see the sample program **ae_pio.c** in **`tern\186\samples\ae`**. You will also find that these functions are used throughout TERN sample files, as most applications do find it necessary to re-configure the PIO lines.

The function **pio_wr** and **pio_rd** can be quite slow when accessing the PIO pins. Depending on the pin being used, it might require from 5-10 µs. The maximum efficiency you can get from the PIO pins occur if you instead modify the PIO registers directly with an **outport** instruction Performance in this case will be around 1-2 µs to toggle any pin.

The data register is **0xff74** for PIO port 0, and **0xff7a** for PIO port 1.

void pio_init
Arguments: char bit, char mode
Return value: none

bit refers to any one of the 32 PIO lines, 0-31.

mode refers to one of four modes of operation.

- 0, High-impedance Input operation
- 1, Open-drain output operation
- 2, output
- 3, peripheral mode

unsigned int pio_rd:
Arguments: char port
Return value: byte indicating PIO status

Each bit of the returned 16-bit value indicates the current I/O value for the PIO pins in the selected port.

void pio_wr:
Arguments: char bit, char dat
Return value: none

Writes the passed in dat value (either 1/0) to the selected PIO.

4.2.4 Timer Units

The three timers present on the SmartLCD can be used for a variety of applications. All three timers run at 1/4 of the processor clock rate, which determines the maximum resolution that can be obtained. Be aware that if you enter power save mode, that means the timers will operate at a reduced speed as well.

These timers are controlled and configured through a mode register which is specified using the software interfaces. The mode register is described in detail in chapter 8 of the AMD 188 CPU User's Manual.

Pulse width demodulation is done by setting the PWD bit in the **SYSCON** register. Before doing this, you will want to specify your interrupt service routines, which are used whenever the incoming digital signal switches from high to low, and low to high.

The timers can be used to time execution of your user defined code by reading the timer values before and after execution of any piece of code. For a sample file demonstrating this application, see the sample file ***timer.c*** in the directory ***tern\186\samples\ae***.

Two of the timers, **Timer0** and **Timer1** can be used to do pulse-width modulation with a variable duty cycle. These timers contain two max counters, where the output is high until the counter counts up to maxcount A before switching and counting up to maxcount B.

It is also possible to use the output of **Timer2** to pre-scale one of the other timers, since 16-bit resolution at the maximum clock rate specified gives you only 150 Hz. Only by using **Timer2** can you slow this down even further. The sample files ***timer02.c*** and ***timer12.c***, located in ***tern\186\samples\ae***, demonstrate this.

The specific behavior that you might want to implement is described in detail in chapter 8 of the AMD 188 CPU User's Manual.

void t0_init
void t1_init
Arguments: int tm, int ta, int tb, void interrupt far(*t_isr)()
Return values: none

Both of these timers have two maximum counters (MAXCOUNTA/B) available. These can all be specified using **ta** and **tb**. The argument **tm** is the value that you wish placed into the **T0CON/T1CON** mode registers for configuring the two timers.

The interrupt service routine **t_isr** specified here is called whenever the full count is reached, with other behavior possible depending on the value specified for the control register.

void t2_init
Arguments: int tm, int ta, void interrupt far(*t_isr)()
Return values: none.

Timer2 behaves like the other timers, except it only has one max counter available.

4.2.5 Analog-to-Digital Conversion

Up to two ADC units, U10 and U29, may be installed. Each provides 11 channels of analog inputs based on the reference voltage supplied to **REF+**. For details regarding the hardware configuration, see the Hardware chapter.

In order to operate the U10 ADC, lines I20, I21,I22 from the U5 PPI must be configured as output. P11 must also be configured to be input. This line is also shared with the RTC and EEPROM, and left high at power-on/reset. You should be sure not to re-program these pins for your own use. Be careful when using the EEPROM concurrently with the ADC. If the ADC is enabled, the line P11 will be reserved for its use and any attempt to access the EEPROM will time-out after some time.

To operate the U29 ADC, lines T17 from PPI U27 and lines I21 and I22 from PPI U5 must be configured as output. T00 (U27) must also be configured to be input. The T00 signal is buffered by U25 Schmitt Trigger input I0.

For a sample file demonstrating the use of the U10 ADC, please see **ae_ad12.c** in **`tern\186\samples\ae`**. For the U29 ADC, see **sl_ad12.c** in **`tern\186\samples\sl`**.

int ae_ad12
int sl_ad12
Arguments: char c
Return values: int ad_value

The argument **c** selects the channel from which to do the next Analog to Digital conversion. A value of 0 corresponds to channel **AD0**, 1 corresponds to channel **AD1**, and so on.

The return value **ad_value** is the latched-in conversion value from the previous call to this function. This means each call to this function actually returns the value latched-in from the previous analog-to-digital conversion.

For example, this means the first analog-to-digital conversion done in an application will be similar to the following:

```
ae_ad12(0); // Read from channel 0
chn_0_data = ae_ad12(0); // Start the next conversion, retrieve value.
```

4.2.6 Digital-to-Analog Conversion

Up to two LTC 1446 chips are available on the SmartLCD in positions **U11** and **U23**. U11 offers two channels, VA and VB, for digital-to-analog conversion. U23 offers channels VC and VD. Details regarding hardware, such as pin-outs and performance specifications, can be found in the Hardware chapter.

A sample program demonstrating the U11 DAC can be found in **ae_da.c** in the directory **`tern\186\samples\ae`**. For the U23 DAC, refer to **a104da.c** in **`tern\186\samples\a104`**.

void ae_da
void a104_da
Arguments: int dat1, int dat2
Return value: none

Argument **dat1** is the current value to drive to channel VA of the chip, while argument **dat2** is the value to drive channel VB of the U11 chip (or channels VC and VD, respectively, of the U23 chip).

These argument values should range from 0-4095, with units of millivolts. This makes it possible to drive a maximum of 4.906 volts to each channel.

A 12-bit DAC channel, VA, is dedicated to drive the contrast voltage by using the software function `contrast(int dat);`.

4.2.7 Other library functions

On-board supervisor MAX691 or LTC691

The watchdog timer offered by the MAX691 or LTC691 offers an excellent way to monitor improper program execution. If the watchdog timer (**J9**) jumper is set, the function **hitwd()** must be called every 1.6 seconds of program execution. If this is not executed because of a run-time error, such as an infinite loop or stalled interrupt service routine, a hardware reset will occur.

void hitwd
Arguments: none
Return value: none

Resets the supervisor timer for another 1.6 seconds.

void led
Arguments: int ledd
Return value: none

Turns the on-board LED on or off according to the value of **ledd**.

Real-Time Clock

The real-time clock can be used to keep track of real time. Backed up by a lithium-coin battery, the real time clock can be accessed and programmed using two interface functions.

The real time clock only allows storage of two digits of the year code, as reflected below. As a result, application developers should be careful to account for a roll-over in digits in the year 2000. One solution might be to store an offset value in non-volatile storage such as the EEPROM.

There is a common data structure used to access and use both interfaces.

```
typedef struct{
  unsigned char sec1; One second digit.
  unsigned char sec10; Ten second digit.
  unsigned char min1; One minute digit.
  unsigned char min10; Ten minute digit.
  unsigned char hour1; One hour digit.
  unsigned char hour10; Ten hour digit.
  unsigned char day1; One day digit.
  unsigned char day10; Ten day digit.
  unsigned char mon1; One month digit.
  unsigned char mon10; Ten month digit.
  unsigned char year1; One year digit.
  unsigned char year10; Ten year digit.
  unsigned char wk; Day of the week.
} TIM;
```

int rtc_rd
Arguments: TIM *r
Return value: int error_code

This function places the current value of the real time clock within the argument **r** structure. The structure should be allocated by the user. This function returns 0 on success and returns 1 in case of error, such as the clock failing to respond.

Void rtc_init
Arguments: char* t
Return value: none

This function is used to initialize and set a value into the real-time clock. The argument **t** should be a null-terminated byte array that contains the new time value to be used.

The byte array should correspond to { ***weekday, year10, year1, month10, month1, day10, day1, hour10, hour1, minute10, minute1, second10, second1,*** 0 }.

If, for example, the time to be initialized into the real time clock is June 5, 1998, Friday, 13:55:30, the byte array would be initialized to:

```
unsigned char t[14] = { 5, 9, 8, 0, 6, 0, 5, 1, 3, 5, 5, 3, 0 };
```

Delay
In many applications it becomes useful to pause before executing any further code. There are functions provided to make this process easy. For applications that require precision timing, you should use hardware timers provided on-board for this purpose.

void delay0
Arguments: unsigned int t
Return value: none

This function is just a simple software loop. The actual time that it waits depends on processor speed as well as interrupt latency. The code is functionally identical to:

```
While(t) { t--; }
```

Passing in a **t** value of 600 causes a delay of approximately 1 ms.

void delay_ms
Arguments: unsigned int
Return value: none

This function is similar to delay0, but the passed in argument is in units of milliseconds instead of loop iterations. Again, this function is highly dependent upon the processor speed.

unsigned int crc16
Arguments: unsigned char *wptr, unsigned int count
Return value: unsigned int value

This function returns a simple 16-bit CRC on a byte-array of **count** size pointed to by **wptr**.

void ae_reset

Arguments: none
Return value: none

This function is similar to a hardware reset, and can be used if your program needs to re-start the board for any reason. Depending on the current hardware configuration, this might either start executing code from the DEBUG ROM or from some other address.

4.3 Functions in SER0.OBJ/SER1.OBJ

The functions described in this section are prototyped in the header file **ser0.h** and **ser1.h** in the directory **`tern\186\include`**.

The internal asynchronous serial ports are functionally identical. SER0 is used by the DEBUG ROM provided as part of the TERN EV/DV software kits for communication with the PC. As a result, you will not be able to debug code directly written for serial port 0.

Two asynchronous serial ports are integrated in the 188 CPU CPU: SER0 and SER1. Both ports have baud rates based on the 40 MHz clock, and can operate at a maximum of 1/16 of that clock rate.

By default, SER0 is used by the DEBUG ROM for application download/debugging in Step One and Step Two. We will use SER1 as the example in the following discussion; any of the interface functions which are specific to SER1 can be easily changed into function calls for SER0. While selecting a serial port for use, please realize that some pins might be shared with other peripheral functions. This means that in certain limited cases, it might not be possible to use a certain serial port with other on-board controller functions. For details, you should see both chapter 10 of the 188 CPU Microprocessor User's Manual and the schematic of the SmartLCD provided at the end of this manual.

TERN interface functions make it possible to use one of a number of predetermined baud rates. These baud rates are achieved by specifying a divisor for 1/16 of the processor frequency.

The following table shows the function arguments that express each baud rate, to be used in TERN functions. These are based on a 40 MHz system clock.

Function Argument	Baud Rate
1	110
2	150
3	300
4	600
5	1200
6	2400
7	4800
8	9600
9	19,200 (default)
10	38,400
11	57,600
12	115,200
13	250,000
14	500,000
15	1,250,000

Table 4.1 Baud rate values

After initialization by calling **`s1_init()`**, SER1 is configured as a full-duplex serial port and is ready to transmit/receive serial data at one of the specified 15 baud rates.

An input buffer, **`ser1_in_buf`** (whose size is specified by the user), will automatically store the receiving serial data stream into the memory by DMA1 operation. In terms of receiving, there is no software overhead or interrupt latency for user application programs even at the highest baud rate. DMA transfer allows efficient handling of incoming data. The user only has to check the buffer status with **`serhit1()`** and take out the data from the buffer with **`getser1()`**, if any. The input buffer is used as a circular ring buffer, as shown in Figure 4.1. However, the transmit operation is interrupt-driven.

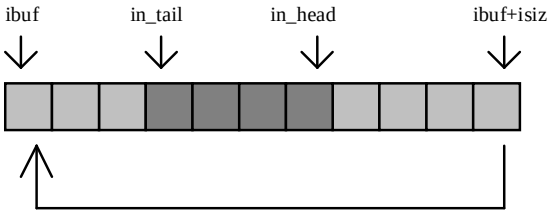


Figure 4.1 Circular ring input buffer

The input buffer (**ibuf**), buffer size (**isiz**), and baud rate (**baud**) are specified by the user with **`s1_init()`** with a default mode of 8-bit, 1 stop bit, no parity. After **`s1_init()`** you can set up a new mode with different numbers for data-bit, stop bit, or parity by directly accessing the Serial Port 0/1 Control Register (SP0CT/SP1CT) if necessary, as described in chapter 10 of the 188 CPU manual for asynchronous serial ports.

Due to the nature of high-speed baud rates and possible effects from the external environment, serial input data will automatically fill in the buffer circularly without stopping, regardless of overwrite. If the user does not take out the data from the ring buffer with **`getser1()`** before the ring buffer is full, new data will overwrite the old data without warning or control. Thus it is important to provide a sufficiently large buffer if large amounts of data are transferred. For example, if you are receiving data at 9600 baud, a 4 KB buffer will be able to store data for approximately four seconds.

However, it is always important to take out data early from the input buffer, before the ring buffer rolls over. You may designate a higher baud rate for transmitting data out and a slower baud rate for receiving data. This will give you more time to do other things, without overrunning the input buffer. You can use **`serhit1()`** to check the status of the input buffer and return the offset of the in_head pointer from the in_tail pointer. A return value of 0 indicates no data is available in the buffer.

You can use **`getser1()`** to get the serial input data byte by byte using FIFO from the buffer. The in_tail pointer will automatically increment after every **`getser1()`** call. It is not necessary to suspend external devices from sending in serial data with /RTS. Only a hardware reset or **`s1_close()`** can stop this receiving operation.

For transmission, you can use **`putser1()`** to send out a byte, or use **`putsers1()`** to transmit a character string. You can put data into the transmit ring buffer, **`s1_out_buf`**, at any time using this method. The transmit ring buffer address (**obuf**) and buffer length (**osiz)** are also specified at the time of initialization. The transmit interrupt service will check the availability of data in the transmit buffer. If there is no more data (the head and tail pointers are equal), it will disable the transmit interrupt. Otherwise, it will continue to take out the data from the out buffer, and transmit. After you call **`putser1()`** and transmit functions, you are free to do other tasks with no additional software overhead on the transmitting operation. It will automatically send out all the data you specify. After all data has been sent, it will clear the busy flag and be ready for the next transmission.

The sample program **`ser1_0.c`** demonstrates how a protocol translator works. It would receive an input HEX file from SER1 and translate every ':' character to '?'. The translated HEX file is then transmitted out of SER0. This sample program can be found in **`tern\186\samples\ae`**.

Software Interface

Before using the serial ports, they must be initialized.

There is a data structure containing important serial port state information that is passed as argument to the TERN library interface functions. The **COM** structure should normally be manipulated only by TERN libraries. It is provided to make debugging of the serial communication ports more practical. Since it allows you to monitor the current value of the buffer and associated pointer values, you can watch the transmission process.

The two serial ports have similar software interfaces. Any interface that makes reference to either **s0** or **ser0** can be replaced with **s1** or **ser1**, for example. Each serial port should use its own **COM** structure, as defined in **ae.h**.

```
typedef struct  {
  unsigned char ready;          /* TRUE when ready */
  unsigned char baud;
  unsigned char mode;
  unsigned char iflag;     /* interrupt status     */
  unsigned char            *in_buf;        /* Input buffer */
  int  in_tail;        /* Input buffer TAIL ptr */
  int  in_head;        /* Input buffer HEAD ptr */
  int  in_size;        /* Input buffer size */
  int  in_crcnt;       /* Input <CR> count */
  unsigned char   in_mt;          /* Input buffer FLAG */
  unsigned char   in_full;        /* input buffer full */
  unsigned char   *out_buf;       /* Output buffer */
  int  out_tail;       /* Output buffer TAIL ptr */
  int  out_head;       /* Output buffer HEAD ptr */
  int  out_size;       /* Output buffer size */
 unsigned char  out_full;       /* Output buffer FLAG */
  unsigned char  out_mt;         /* Output buffer MT */
  unsigned char tmso;   // transmit macro service operation
  unsigned char rts;
  unsigned char dtr;
  unsigned char en485;
  unsigned char err;
  unsigned char node;
  unsigned char cr; /* scc CR register    */
  unsigned char slave;
  unsigned int in_segm;       /* input buffer segment */
  unsigned int in_offs;       /* input buffer offset */
  unsigned int out_segm;       /* output buffer segment */
  unsigned int out_offs;       /* output buffer offset */
  unsigned char byte_delay;   /* V25 macro service byte delay */
} COM;
```

s*n*_init

Arguments: unsigned char b, unsigned char* ibuf, int isiz, unsigned char* obuf, int osiz, COM* c
Return value: none

This function initializes either SER0 or SER1 with the specified parameters. **b** is the baud rate value shown in Table 4.1. Arguments **ibuf** and **isiz** specify the input-data buffer, and **obuf** and **osiz** specify the location and size of the transmit ring buffer.

The serial ports are initialized for 8-bit, 1 stop bit, no parity communication.

There are a couple different functions used for transmission of data. You can place data within the output buffer manually, incrementing the head and tail buffer pointers appropriately. If you do not call one of the following functions, however, the driver interrupt for the appropriate serial-port will be disabled, which means that no values will be transmitted. This allows you to control when you wish the transmission of data within the outbound buffer to begin. Once the interrupts are enabled, it is dangerous to manipulate the values of the outbound buffer, as well as the values of the buffer pointer.

putser*n*
Arguments: unsigned char outch, COM *c
Return value: int return_value

This function places one byte **outch** into the transmit buffer for the appropriate serial port. The return value returns one in case of success, and zero in any other case.

putsers*n*
Arguments: char* str, COM *c
Return value: int return_value

This function places a null-terminated character string into the transmit buffer. The return value returns one in case of success, and zero in any other case.

DMA transfer automatically places incoming data into the inbound buffer. **serhit*n*()** should be called before trying to retrieve data.

serhit*n*
Arguments: COM *c
Return value: int value

This function returns 1 as **value** if there is anything present in the in-bound buffer for this serial port.

getser*n*
Arguments: COM *c
Return value: unsigned char value

This function returns the current byte from **s*n*_in_buf**, and increments the **in_tail** pointer. Once again, this function assumes that **serhit*n*** has been called, and that there is a character present in the buffer.

getsers*n*
Arguments: COM c, int len, char* str
Return value: int value

This function fills the character buffer **str** with at most **len** bytes from the input buffer. It also stops retrieving data from the buffer if a carriage return (ASCII: **0x0d**) is retrieved.

This function makes repeated calls to **getser**, and will block until **len** bytes are retrieved. The return **value** indicates the number of bytes that were placed into the buffer.

Be careful when you are using this function. The returned character string is actually a byte array terminated by a null character. This means that there might actually be multiple null characters in the byte array, and the returned **value** is the only definite indicator of the number of bytes read. Normally, we suggest that the **getsers** and **putsers** functions only be used with ASCII character strings. If you are working with byte arrays, the single-byte versions of these functions are probably more appropriate.

Miscellaneous Serial Communication Functions

One thing to be aware of in both transmission and receiving of data through the serial port is that TERN drivers only use the basic serial-port communication lines for transmitting and receiving data. Hardware flow control in the form of **CTS** (Clear-To-Send) and **RTS** (Ready-To-Send) is not implemented. There are, however, functions available that allow you to check and set the value of these I/O pins appropriate for whatever form of flow control you wish to implement. Before using these functions, you should once again be aware that the peripheral pin function you are using might not be selected as needed. For details, please refer to the 188 CPU User's Manual.

char s*n*_cts(void)
Retrieves value of **CTS** pin.

void s*n*_rts(char b)
Sets the value of **RTS** to **b**.

Completing Serial Communications

After completing your serial communications, there are a few functions that can be used to reset default system resources.

s*n*_close
Arguments: COM *c
Return value: none

This closes down the serial port, by shutting down the hardware as well as disabling the interrupt.

clean_ser*n*
Arguments: COM *c
Return value: none

This flushes the input buffer by resetting the tail and header buffer pointers.

The asynchronous serial I/O ports available on the 188 CPU Processor have many other features that might be useful for your application. If you are truly interested in having more control, please read Chapter 10 of the manual for a detailed discussion of other features available to you.

4.4 Functions in SCC.OBJ

The functions found in this object file are prototyped in **scc.h** in the **`tern\186\include`** directory.

The SCC is a component that is used to provide a third asynchronous port. It uses an 8 MHz crystal, different from the system clock speed, for driving serial communications. This means the divisors and function arguments for setting up the baud rate for this third port are different than for SER0 and SER1.

The SCC2691 component has its own 8 MHz crystal providing the clock signal. By default, this is set to 8 MHz to be consistent with earlier TERN controller designs. The highest standard baud rate is 19,200, as shown in the table below. If your application requires a higher standard baud rate (115,200, for example), it is possible to replace this crystal with a custom 3.6864 MHz crystal. A sample file demonstrating how the software would be changed for this application is **ae_scc1.c**, found in the **tern\186\samples\ae** directory.

Function Argument	Baud Rate
1	110
2	150
3	300
4	600

Function Argument	Baud Rate
5	1200
6	2400
7	4800
8	9600 (default)
9	19,200
10	31,250
11	62,500
12	125,000
13	250,000

Unlike the other serial ports, DMA transfer is not used to fill the input buffer for SCC. Instead, an interrupt-service-routine is used to place characters into the input buffer. If the processor does not respond to the interrupt—because it is masked, for example—the interrupt service routine might never be able to complete this process. Over time, this means data might be lost in the SCC as bytes overflow.

Special control registers are used to define how the SCC operates. For a detailed description of registers **MR1** and **MR2**, please see Appendix B of this manual. In most TERN applications, MR1 is set to ***0x57***, and MR2 is set to ***0x07***. This configures the SCC for no flow control (RTS, CTS not used/checked), no parity, 8-bit, normal operation. Other configurations are also possible, providing self-echo, even-odd parity, up to 2 stop bits, 5 bit operation, as well as automatic hardware flow control.

Initialization occurs in a manner otherwise similar to SER0 and SER1. A **COM** structure is once again used to hold state information for the serial port. The in-bound and out-bound buffers operate as before, and must be provided upon initialization.

scc_init
Arguments: unsigned char m1, unsigned char m2, unsigned char b, unsigned char* ibuf, int isiz, unsigned char* obuf, int osiz, COM *c
Return value: none

This initializes the SCC2691 serial port to baud rate **b**, as defined in the table above. The values in **m1** and **m2** specify the values to be stored in to **MR1** and **MR2**. As discussed above, these values are normally ***0x57*** and ***0x07***, as shown in TERN sample programs.

ibuf and **isiz** define the input buffer characteristics, and **obuf** and **osiz** define the output buffer.

After initializing the serial port, you must also set up the interrupt service routine. The SCC2691 UART takes up external interrupt **/INT0** on the CPU, and you must set up the appropriate interrupt vector to

handle this. An interrupt service routine, **scc_isr()**, has been written to handle the interrupt, and it enables/disables the interrupt as needed to transmit and receive data with the data buffers. So, after initialization, you will need to make a call to do this:

```
int0_init(1, scc_isr);
```

By default, the SCC is disabled for both *transmit* and *receive*. Before using the port, you will need to enable these functionalities.

When using RS232 in full-duplex mode, *transmit* and *receive* functions should both be enabled. Once this is done, you can transmit and receive data as needed. If you do need to do limited flow control, the MPO pin on the J1 header can be used for RTS. For a sample file showing RS232 full duplex communications, please see **ae_scc.c** in the directory **`tern\186\samples\ae`**.

RS485 is slightly more complex to use than RS232. RS485 operation is half-duplex only, which means transmission does not occur concurrently with reception. The RS485 driver will echo back bytes sent to the SCC. As a result, assuming you are using the RS485 driver installed on another TERN peripheral board,

you will need to disable *receive* while transmitting. While transmitting, you will also need to place the RS485 driver in transmission mode as well. This is done by using **scc_rts(1)**. This uses pin MPO (multi-purpose output) found on the J1 header. While you are receiving data, the RS485 driver will need to be placed in receive mode using **scc_rts(0)**. For a sample file showing RS485 communication, please see **ae_rs485.c** in the directory **`tern\186\samples\ae`**.

en485
Arguments: int i
Return value: none

This function sets the pin MPO either high (i = 1) or low (i = 0). The function scc_rts() actually has a similar function, by pulling the same pin high or low, but is intended for use in flow control.

scc_send_e/scc_rec_e
Arguments: none
Return value: none

This function enables transmission or reception on the SCC2691 UART. After initialization, both of these functions are disabled by default. If you are using RS485, only one of these two functions should be enabled at any one time.

scc_send_reset/scc_rec_reset
Arguments: none
Return value: none

This function resets the state of the send and receive function of the SCC2691. One major use of these functions is to disable *transmit* and *receive*. If you are using RS485, you will need to use this feature when transitioning from transmission to reception, or from reception to transmission.

Transmission and reception of data using the SCC is in most ways identical to SER0 and SER1. The functions used to transmit and receive data are similar. For details regarding these functions, please refer to the previous section.

putser_scc
See: **`putser`*`n`***

putsers_scc
See: **`putsers`*`n`***

getser_scc
See: **`getser`*`n`***

getsers_scc
See: **`getsers`*`n`***

Flow control is also handled in a mostly similar fashion. The CTS pin corresponds to the MPI pin, which is not connected to either one of the headers. The RTS pin corresponds to the MPO pin found on the J1 header.

scc_cts
See: **`s`*`n`*`_cts`**
scc_rts
See: **`s`*`n`*`_rts`**

Other SCC functions are similar to those for SER0 and SER1.

scc_close
See: **s*n*_close**
serhit_scc
See: **s*n*_hit**
clean_ser_scc
See: **clean_s*n***

Occasionally, it might also be necessary to check the state of the SCC for information regarding errors that might have occurred. By calling **scc_err**, you can check for framing errors, parity errors (if parity is enabled), and overrun errors.

scc_err
Arguments: none
Return value: unsigned char val

The returned value **val** will be in the form of 0ABC0000 in binary. Bit A is 1 to indicate a framing error. Bit B is 1 to indicate a parity error, and bit C indicates an over-run error.

4.5 Functions in AEEE.OBJ

The 512-byte serial EEPROM (***24C04***) provided on-board allows easy storage of non-volatile program parameters. This is usually an ideal location to store important configuration values that do not need to be changed often. Access to the EEPROM is quite slow, compared to memory access on the rest of the controller.

Part of the EEPROM is reserved for TERN use specifically for this purpose.

Addresses **0x00** to **0x1f** on the EEPROM is reserved for system use, including configuration information about the controller itself, jump address for Step Two, and other data that is of a more permanent nature.

The rest of the EEPROM memory space, **0x20** to **0x1ff**, is available for your application use.

ee_wr
Arguments: int addr, unsigned char dat
Return value: int status

This function is used to write the passed in **dat** to the specified **addr**. The return value is 0 in success.

ee_rd
Arguments: int addr
Return value: int data

This function returns one byte of data from the specified address.

4.6 Functions in SL.LIB

C function prototypes supporting SmartLCD hardware can be found in SL.H, in the **\tern\186\include** directory.

The following sample programs can be found in the **\tern\186\samples\SL** directory:

sl_grid.c
sl_bird.c
sl_ppi.c
sl_ccfl.c
sl_pb.c

sl_ad12.c
sl_text.c

4.7 Functions in SLB.LIB

Additional software drivers can be found in slb.h and other slb header files. The source code for these functions is included in the tern\186\samples\sl\source directory. Functions in this library include drivers to create buttons and handle button events. The following samples illustrate some of the basic operations:

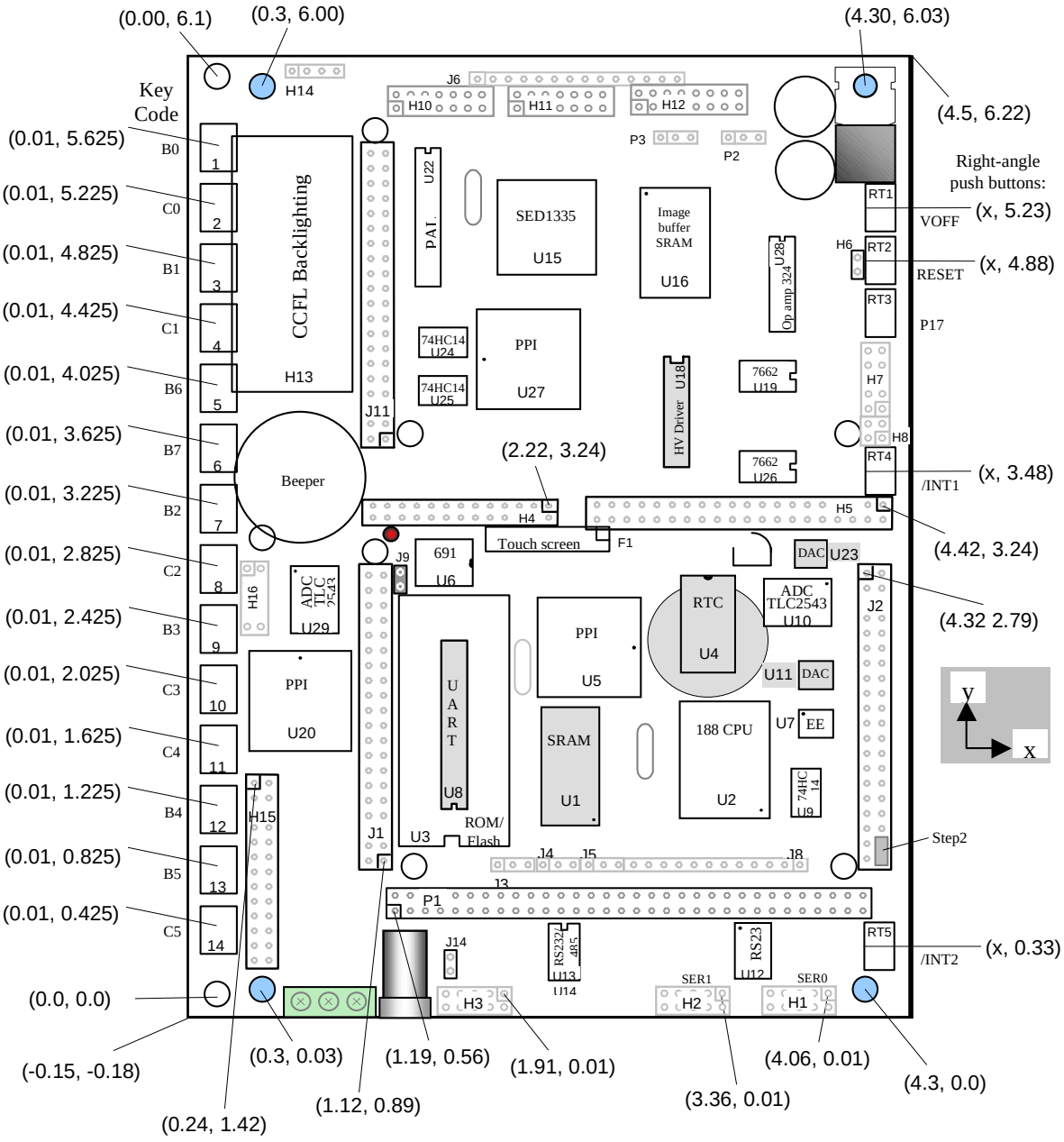
sl_bttn.c

sl_demo.c

sl_gpx.c

Appendix A: SmartLCD Layout

The **SmartLCD** measures 6.40 x 4.65 inches. Its layout is shown below.
All dimensions are in inches. Shaded mounting holes are used for the LCD.



Appendix B: UART SCC2691

1. Pin Description

D0-D7	Data bus, active high, bi-directional, and having 3-State
/CEN	Chip enable, active-low input
/WRN	Write strobe, active-low input
/RDN	Read strobe, active-low input
A0-A2	Address input, active-high address input to select the UART registers
RESET	Reset, active-high input
INTRN	Interrupt request, active-low output
X1/CLK	Crystal 1, crystal or external clock input
X2	Crystal 2, the other side of crystal
RxD	Receive serial data input
TxD	Transmit serial data output
MPO	Multi-purpose output
MPI	Multi-purpose input
Vcc	Power supply, +5 V input
GND	Ground

2. Register Addressing

A2	A1	A0	READ (RDN=0)	WRITE (WRN=0)
0	0	0	MR1,MR2	MR1, MR2
0	0	1	SR	CSR
0	1	0	BRG Test	CR
0	1	1	RHR	THR
1	0	0	1x/16x Test	ACR
1	0	1	ISR	IMR
1	1	0	CTU	CTUR
1	1	1	CTL	CTLR

Note:

ACR = Auxiliary control register
BRG = Baud rate generator
CR = Command register
CSR = Clock select register
CTL = Counter/timer lower
CTLR = Counter/timer lower register
CTU = Counter/timer upper
CTUR = Counter/timer upper register
MR = Mode register
SR = Status register
RHR = Rx holding register
THR = Tx holding register

3. Register Bit Formats

MR1 (Mode Register 1):

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RxRTS 0 = no 1 = yes	RxINT 0=RxRDY 1=FFULL	Error 0 = char 1 = block	___Parity Mode___ 00 = with parity 01 = Force parity 10 = No parity 11 = Special mode		Parity Type 0 = Even 1 = Odd In Special mode: 0 = Data 1 = Addr	Bits per Character 00 = 5 01 = 6 10 = 7 11 = 8	

MR2 (Mode Register 2):

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Channel Mode	TxRTS	CTS Enable Tx	Stop Bit Length (add 0.5 to cases 0-7 if channel is 5 bits/character)
00 = Normal 01 = Auto echo 10 = Local loop 11 = Remote loop	0 = no 1 = yes	0 = no 1 = yes	0 = 0.563 4 = 0.813 8 = 1.563 C = 1.813 1 = 0.625 5 = 0.875 9 = 1.625 D = 1.875 2 = 0.688 6 = 0.938 A = 1.688 E = 1.938 3 = 0.750 7 = 1.000 B = 1.750 F = 2.000

CSR (Clock Select Register):

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Receiver Clock Select	Transmitter Clock Select
when ACR[7] = 0: 0 = 50 1 = 110 2 = 134.5 3 = 200 4 = 300 5 = 600 6 = 1200 7 = 1050 8 = 2400 9 = 4800 A = 7200 B = 9600 C = 38.4k D = Timer E = MPI-16x F = MPI-1x when ACR[7] = 1: 0 = 75 1 = 110 2 = 134.5 3 = 150 4 = 300 5 = 600 6 = 1200 7 = 2000 8 = 2400 9 = 4800 A = 7200 B = 1800 C = 19.2k D = Timer E = MPI-16x F = MPI-1x	when ACR[7] = 0: 0 = 50 1 = 110 2 = 134.5 3 = 200 4 = 300 5 = 600 6 = 1200 7 = 1050 8 = 2400 9 = 4800 A = 7200 B = 9600 C = 38.4k D = Timer E = MPI-16x F = MPI-1x when ACR[7] = 1: 0 = 75 1 = 110 2 = 134.5 3 = 150 4 = 300 5 = 600 6 = 1200 7 = 2000 8 = 2400 9 = 4800 A = 7200 B = 1800 C = 19.2k D = Timer E = MPI-16x F = MPI-1x

CR (Command Register):

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Miscellaneous Commands		Disable Tx	Enable Tx	Disable Rx	Enable Rx
0 = no command 1 = reset MR pointer 2 = reset receiver 3 = reset transmitter 4 = reset error status 5 = reset break change INT 6 = start break 7 = stop break	8 = start C/T 9 = stop counter A = assert RTSN B = negate RTSN C = reset MPI change INT D = reserved E = reserved F = reserved	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes

SR (Channel Status Register):

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Received Break	Framing Error	Parity Error	Overrun Error	TxEMT	TxRDY	FFULL	RxRDY
0 = no 1 = yes *	0 = no 1 = yes *	0 = no 1 = yes *	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes

Note:
* These status bits are appended to the corresponding data character in the receive FIFO. A read of the status register provides these bits [7:5] from the top of the FIFO together with bits [4:0]. These bits are cleared by a reset error status command. In character mode they are reset when the corresponding data character is read from the FIFO.

ACR (Auxiliary Control Register):

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

BRG Set Select	Counter/Timer Mode and Source	Power-Down Mode	MPO Pin Function Select
0 = Baud rate set 1, see CSR bit format 1 = Baud rate set 2, see CSR bit format	0 = counter, MPI pin 1 = counter, MPI pin divided by 16 2 = counter, TxC-1x clock of the transmitter 3 = counter, crystal or external clock (x1/CLK) 4 = timer, MPI pin 5 = timer, MPI pin divided by 16 6 = timer, crystal or external clock (x1/CLK) 7 = timer, crystal or external clock (x1/CLK) divided by 16	0 = on, power down active 1 = off normal	0 = RTSN 1 = C/TO 2 = TxC (1x) 3 = TxC (16x) 4 = RxC (1x) 5 = RxC (16x) 6 = TxRDY 7 = RxRDY/FFULL

ISR (Interrupt Status Register):

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

MPI Pin Change	MPI Pin Current State	Not Used	Counter Ready	Delta Break	RxRDY/ FFULL	TxEMT	TxRDY
0 = no 1 = yes	0 = low 1 = high		0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes

IMR (Interrupt Mask Register):

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

MPI Change Interrupt	MPI Level Interrupt	Not Used	Counter Ready Interrupt	Delta Break Interrupt	RxRDY/ FFULL Interrupt	TxEMT Interrupt	TxRDY Interrupt
0 = off 1 = 0n	0 = off 1 = 0n		0 = off 1 = 0n	0 = off 1 = 0n	0 = off 1 = 0n	0 = off 1 = 0n	0 = off 1 = 0n

CTUR (Counter/Timer Upper Register):

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
C/T [15]	C/T [14]	C/T [13]	C/T [12]	C/T [11]	C/T [10]	C/T [9]	C/T [8]

CTLR (Counter/Timer Lower Register):

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
C/T [7]	C/T [6]	C/T [5]	C/T [4]	C/T [3]	C/T [2]	C/T [1]	C/T[0]

Appendix C: RTC72421 / 72423

Function Table

Address					Data					
A_3	A_2	A_1	A_0	Register	D_3	D_2	D_1	D_0	Count Value	Remarks
0	0	0	0	S_1	s_8	s_4	s_2	s_1	0~9	1-second digit register
0	0	0	1	S_{10}		s_{40}	s_{20}	s_{10}	0~5	10-second digit register
0	0	1	0	MI_1	mi_8	mi_4	mi_2	mi_1	0~9	1-minute digit register
0	0	1	1	MI_{10}		mi_{40}	mi_{20}	mi_{10}	0~5	10-minute digit register
0	1	0	0	H_1	h_8	h_4	h_2	h_1	0~9	1-hour digit register
0	1	0	1	H_{10}		PM/AM	h_{20}	h_{10}	0~2 or 0~1	PM/AM, 10-hour digit register
0	1	1	0	D_1	d_8	d_4	d_2	d_1	0~9	1-day digit register
0	1	1	1	D_{10}			d_{20}	d_{10}	0~3	10-day digit register
1	0	0	0	MO_1	mo_8	mo_4	mo_2	mo_1	0~9	1-month digit register
1	0	0	1	MO_{10}				mo_{10}	0~1	10-month digit register
1	0	1	0	Y_1	y_8	y_4	y_2	y_1	0~9	1-year digit register
1	0	1	1	Y_{10}	y_{80}	y_{40}	y_{20}	y_{10}	0~9	10-year digit register
1	1	0	0	W		w_4	w_2	w_1	0~6	Week register
1	1	0	1	Reg D	30s Adj	IRQ Flag	Busy	Hold		Control register D
1	1	1	0	Reg E	t_1	t_0	INT/ STD	Mask		Control register E
1	1	1	1	Reg F	Test	24/ 12	Stop	Rest		Control register F

Note: 1) INT/STD = Interrupt/Standard, Rest = Reset;

2) Mask AM/PM bit with 10's of hours operations;

3) Busy is read only, IRQ can only be set low ("0");

4)

Data bit	PM/AM	INT/STD	24/12
1	PM	INT	24
0	AM	STD	12

5) Test bit should be "0".

Appendix D: Serial EEPROM Map

Part of the on-board serial EEPROM is used by system software. Application programs must not use these locations.

0x00-0x1F: Reserved
0x20-0x1FF: Free for application

0x00	Node Address, for networking
0x01	Board Type
0x02	
0x03	
0x04	SER0_receive, used by ser0.c
0x05	SER0_transmit, used by ser0.c
0x06	SER1_receive, used by ser1.c
0x07	SER1_transmit, used by ser1.c
0x10	CS high byte, used by ACTR™
0x11	CS low byte, used by ACTR™
0x12	IP high byte, used by ACTR™
0x13	IP low byte, used by ACTR™
0x14 – 0x1F	Reserved
0x20 – 0x1FF	Free for application

Appendix E: Software Glossary

The following is a glossary of library functions for the SmartLCD. Reference sample programs can be found in two directories:

```
C:\tern\186\samples\AE
C:\tern\186\samples\SL
```

---

void ae_init(void) ae.h

Initializes the Am188ES processor. The following is the source code for ***ae_init()***

```
outport(0xffa0,0xc0bf);     // UMCS, 256K ROM, 3 wait states, disable AD15-0
outport(0xffa2,0x7fbc);     // 512K RAM, 0 wait states
outport(0xffa8,0xa0bf); // 256K block, 64K MCS0, PCS I/O
outport(0xffa6,0x81ff);     // MMCS, base 0x80000
outport(0xffa4,0x007f); // PACS, base 0, 15 wait

outport(0xff78,0xe73c);     // PDIR1, TxD0, RxD0, TxD1, RxD1, P16=PCS0, P17=PCS1=PPI
outport(0xff76,0x0000);     // PIOM1
outport(0xff72,0xec7b);     // PDIR0, P12,A19,A18,A17,P2=PCS6=RTC
outport(0xff70,0x1000);     // PIOM0, P12=LED

outportb(0x0103,0x9a);     // all pins are input, I20-23 output
outportb(0x0100,0);
outportb(0x0101,0);
outportb(0x0102,0x01);     // I20=ADCS high
clka_en(0);
enable();
```

Reference: led.c

---

void ae_reset(void) ae.h

Resets Am188ES processor.

---

void delay_ms(int m) ae.h

Approximate microsecond delay. Does not use timer.

```
Var:  m – Delay in approximate ms
```

Reference: led.c

---

void led(int i) ae.h

Toggles P12 used for led.

```
Var:  i - Led on or off
```

Reference: led.c

---

void delay0(unsigned int t) ae.h

Approximate loop delay. Does not use timer.

```
Var:  m – Delay using simple for loop up to t.
```

Reference:

---

void pwr_save_en(int i) ae.h

Enables power save mode, which reduces clock speed. Timers and serial ports will be effected. Disabled by external interrupt.

```
Var:  i – 1 enables power save only.  Does not disable.
```

Reference: ae_pwr.c

---

void clka_en(int i) ae.h

Enables signal CLK respectively for external peripheral use.

```
Var: i – 1 enables clock output, 0 disables (saves current when
disabled).
```

Reference:

---

void hitwd(void) ae.h

Hits the watchdog timer using P03. P03 must be connected to WDI of the MAX691 supervisor chip.

Reference: *See Hardware chapter of this manual for more information on the MAX691.*

---

void pio_init(char bit, char mode) ae.h

Initializes a PIO line to the following:
mode=0, Normal operation
mode=1, Input with pullup/down
mode=2, Output
mode=3, input without pull

```
Var:  bit – PIO line 0 - 31
      Mode – above mode select
```

Reference: ae_pio.c

---

void pio_wr(char bit, char dat) ae.h

Writes a bit to a PIO line. PIO line must be in an output mode
mode=0, Normal operation
mode=1, Input with pullup/down
mode=2, Output
mode=3, input without pull

```
Var:  bit – PIO line 0 - 31
      dat – 1/0
```

Reference: ae_pio.c

---

unsigned int pio_rd(char port) ae.h

Reads a 16-bit PIO port.

```
Var:  port – 0: PIO 0 - 15
             1: PIO 16 – 31
```

Reference: ae_pio.c

---

void outport(int portid, int value) dos.h

Writes 16-bit *value* to I/O address *portid*.

```
Var:  portid – I/O address
      value – 16 bit value
```

Reference: ae_ppi.c

---

void outportb(int portid, int value) dos.h

Writes 8-bit *value* to I/O address *portid*.

```
Var:  portid – I/O address
      value – 8 bit value
```

Reference: ae_ppi.c

---

int inport(int portid) dos.h

Reads from an I/O address *portid*. Returns 16-bit value.

```
Var:  portid – I/O address
```

Reference: ae_ppi.c

---

int inportb(int portid) dos.h

Reads from an I/O address *portid*. Returns 8-bit value.

```
Var:  portid – I/O address
```

Reference: ae_ppi.c

---

int ee_wr(int addr, unsigned char dat) aeee.h

Writes to the serial EEPROM.

```
Var:  addr – EEPROM data address
      dat - data
```

Reference: ae_ee.c

---

int ee_rd(int addr) aeee.h

Reads from the serial EEPROM. Returns 8-bit data

```
Var:  addr – EEPROM data address
```

Reference: ae_ee.c

---

int ae_ad12(unsigned char c) ae.h

Reads from the 11-channel 12-bit ADC. Returns 12 bit AD data of the previous channel.
In order to operate ADC, I20,I21,I22 must be output and P11 must be input.
P11 is shared by RTC, EE. It must left high at power-on/reset.
Unipolar:

Vref- = 0x000
Vref+ = 0xfff

Use 1 wait state for Memory and I/O without RDY, < 300 us execution time
Use 0 wait state for Memory and I/O with VEP010, < 270 us execution time

```
Var:  c – ADC channel.
                  c = {0 … a}, input ch = 0 – 10
                  c = b,       input ch = (vref+ - vref-) /2
                  c = c,       input ch = vref-
                  c = d,       input ch = vref+
                  c = e,       software power down
```

Reference: ae_ad12.c

---

void io_wait(char wait) ae.h

Setup I/O wait states for I/O instructions.

```
Var:  wait – wait duration {0…7}
      wait=0, wait states = 0, I/O enable for 100 ns
      wait=1, wait states = 1, I/O enable for 100+25 ns
      wait=2, wait states = 2, I/O enable for 100+50 ns
      wait=3, wait states = 3, I/O enable for 100+75 ns
      wait=4, wait states = 5, I/O enable for 100+125 ns
      wait=5, wait states = 7, I/O enable for 100+175 ns
      wait=6, wait states = 9, I/O enable for 100+225 ns
      wait=7, wait states = 15, I/O enable for 100+375 ns
```

Reference:

---

void rtc_init(unsigned char * time) ae.h

Sets real time clock date, year and time.

```
Var:  time – time and date string
      String sequence is the following:
            time[0] = weekday
            time[1] = year10
            time[2] = year1
            time[3] = mon10
            time[4] = mon1
            time[5] = day10
            time[6] = day1
            time[7] = hour10
            time[8] = hour1
            time[9] = min10
            time[10] = min1
            time[11] = sec10
            time[12] = sec1
unsigned char time[]={2,9,8,0,7,0,1,1,3,1,0,2,0};
/* Tuesday, July 01, 1998, 13:10:20 */
```

Reference: rtc_init.c

---

int rtc_rd(TIM *r) ae.h

Reads from the real time clock.

```
Var:  *r – Struct type TIM for all of the RTC data
      typedef struct{
            unsigned char sec1, sec10,  min1, min10, hour1, hour10;
            unsigned char day1, day10, mon1, mon10, year1, year10;
            unsigned char wk;
            } TIM;
```

Reference: rtc.c

---

void t2_init(int tm, int ta, void interrupt far(*t2_isr)()); ae.h

void t1_init(int tm, int ta, int tb, void interrupt far(*t1_isr)());
void t0_init(int tm, int ta, int tb, void interrupt far(*t0_isr)());

Timer 0, 1, 2 initialization.

```
Var:  tm – Timer mode. See pg. 8-3 and 8-5 of the AMD CPU Manual
      ta – Count time a (1/4 clock speed).
      tb – Count time b for timer 0 and 1 only (1/4 clock).
           Time a and b establish timer duty cycle (PWM). See
           hardware chapter.
      t#_isr – pointer to timer interrupt routine.
```

Reference: timer.c, timer1.c, timer02.c, timer2.c, timer0.c timer12.c

---

void nmi_init(void interrupt far (* nmi_isr)()); ae.h
void int0_init(unsigned char i, void interrupt far (*int0_isr)());
void int1_init(unsigned char i, void interrupt far (*int1_isr)());
void int2_init(unsigned char i, void interrupt far (*int2_isr)());
void int3_init(unsigned char i, void interrupt far (*int3_isr)());
void int4_init(unsigned char i, void interrupt far (*int4_isr)());
void int5_init(unsigned char i, void interrupt far (*int5_isr)());
void int6_init(unsigned char i, void interrupt far (*int6_isr)());

Initialization for interrupts 0 through 6 and NMI (Non-Maskable Interrupt).

```
Var:  i – 1: enable, 0: disable.
      int#_isr – pointer to interrupt service.
```

Reference: intx.c

---

void s0_init(unsigned char b, unsigned char* ibuf, int isiz, ser0.h
unsigned char* obuf, int osiz, COM *c) (void);
void s1_init(unsigned char b, unsigned char* ibuf, int isiz, ser1.h
unsigned char* obuf, int osiz, COM *c) (void);

Serial port 0, 1 initialization.

```
Var:  b – baud rate. Table below for 40MHz and 20MHz Clocks.
      ibuf – pointer to input buffer array
      isiz – input buffer size
      obuf – pointer to output buffer array
      osiz – ouput buffer size
      c – pointer to serial port structure. See AE.H for COM
      structure.
```

b	baud (40MHz)	baud (20MHz)
1	110	55
2	150	110
3	300	150
4	600	300
5	1200	600
6	2400	1200
7	4800	2400
8	9600	4800

b	baud (40MHz)	baud (20MHz)
9	19200	9600
10	38400	19200
11	57600	38400
12	115200	57600
13	23400	115200
14	460800	23400
15	921600	460800

Reference: s0_echo.c, s1_echo.c, s1_0.c

---

void scc_init(unsigned char m1, unsigned char m2, unsigned char b, unsigned char* ibuf,int isiz, unsigned char* obuf,int osiz, COM *c) scc.h

Serial port 0, 1 initialization.

Var: m1 = SCC691 MR1
m2 = SCC691 MR2
b – baud rate. T*able below for 8MHz Clock.*
ibuf – pointer to input buffer array
isiz – input buffer size
obuf – pointer to output buffer array
osiz – ouput buffer size
c – pointer to serial port structure. *See* AE.H *for* COM *structure.*

m1 bit	Definition
7	(RxRTS) receiver request-to-send control, 0=no, 1=yes
6	(RxINT) receiver interrupt select, 0=RxRDY, 1=FIFO FULL
5	(Error Mode) Error Mode Select, 0 = Char., 1=Block
4-3	(Parity Mode), 00=with, 01=Force, 10=No, 11=Special
2	(Parity Type), 0=Even, 1=Odd
1-0	(# bits) 00=5, 01=6, 10=7, 11=8

m2 bit	Definition
7-6	(Modes) 00=Normal, 01=Echo, 10=Local loop, 11=Remote loop
5	(TxRTS) Transmit RTS control, 0=No, 1= Yes
4	(CTS Enable Tx), 0=No, 1=Yes
3-0	(Stop bit), 0111=1, 1111=2

b	baud (8MHz)
1	110
2	150
3	300
4	600
5	1200
6	2400
7	4800
8	9600
9	19200
10	31250
11	62500
12	125000
13	250000

Reference: s0_echo.c, s1_echo.c, s1_0.c

---

int putser0(unsigned char ch, COM *c); ser0.h

int putser1(unsigned char ch, COM *c); ser1.h
int putser_scc(unsigned char ch, COM *c); scc.h

Output 1 character to serial port. Character will be sent to serial output with interrupt isr.

```
Var:  ch – character to output
      c – pointer to serial port structure
```

Reference: s0_echo.c, s1_echo.c, s1_0.c

---

int putsers0(unsigned char *str, COM *c); ser0.h
int putsers1(unsigned char *str, COM *c); ser1.h
int putsers_scc(unsigned char ch, COM *c); scc.h

Output a character string to serial port. Character will be sent to serial output with interrupt isr.

```
Var:  str – pointer to output character string
      c – pointer to serial port structure
```

Reference: ser1_sin.c

---

int serhit0(COM *c); ser0.h
int serhit1(COM *c); ser1.h
int serhit_scc(COM *c); scc.h

Checks input buffer for new input characters. Returns 1 if new character is in input buffer, else 0.

```
Var:  c – pointer to serial port structure
```

Reference: s0_echo.c, s1_echo.c, s1_0.c

---

unsigned char getser0(COM *c); ser0.h
unsigned char getser1(COM *c); ser1.h
unsigned char getser_scc(COM *c); scc.h

Retrieve 1 character from the input buffer. Assumes that *serhit* routine was evaluated.

```
Var:  c – pointer to serial port structure
```

Reference: s0_echo.c, s1_echo.c, s1_0.c

---

int getsers0(COM *c, int len, unsigned char *str); ser0.h
int getsers1(COM *c, int len, unsigned char *str); ser1.h
int getsers_scc(COM *c, int len, unsigned char *str); scc.h

Retrieves a fixed length character string from the input buffer. If the buffer contains less characters than the length requested, *str* will contain only the remaining characters from the buffer. Appends a '\0' character to the end of *str*. Returns the retrieved string length.

```
Var:  c – pointer to serial port structure
      len – desired string length
      str – pointer to output character string
```

Reference: ser1.h, ser0.h for source code.

---

void draw_H_line(int x1, int x2, int y) sl.h

Draws a horizontal line from (x1,y) to (x2,y).

```
Var:  x1, x2, y
```

Reference: sl_text.c

---

void clear_H_line(int x1, int x2, int y) sl.h

Clears a horizontal line from (x1,y) to (x2,y).

```
Var:  x1, x2, y
```

Reference: sl_text.c

---

void fill_square(int upx, int upy, int length, int width) sl.h

Fills a square area at coordinate (upx, upy) with “1”s.

```
Var:  upx, upy, length, width
```

Reference: sl_grid.c

---

void zero_square(int upx, int upy, int length, int width) sl.h

Fills a square area at coordinate (upx, upy) with zeros (“0”s).

```
Var:  upx, upy, length, width
```

Reference: sl_grid.c

---

void write_text(char row, char col, char* buf) sl.h

Row = 1-30, col = 1-40
Places string text char*buf at row and column.

```
Var:  row, col
```

Reference: sl_text.c

---

void clear_text(char row, char col, char num) sl.h

Row = 1-30, col = 1-40, col+num < 40
Clears text at row and col, fills in num " ", up to 40 = col+num.

```
Var:  row, col, num
```

Reference: sl_text.c

---

void sl_init(void) sl.h

Initial SmartLCD 320x240.
Graphics memory starts at 0x800. Text memory starts at 0x000.

Reference: sl_bird.c, sl_grid.c

---

void contrast(int vlc) sl.h

Outputs LCD contrast voltage with a 12-bit DAC, where vlc = 0 to 4095.

```
Var:  vlc
```

Reference: sl_bird.c, sl_grid.c

---

int power_in(void) sl.h

Returns 12-bit ADC input reading at ADC channel 10 (U10), related to the DC power input.

Reference: ae_ad12.c

---

void ccfl(char onoff) sl.h

Turns the CCFL backlighting on/off with HV6 + HV4.
onoff=1, CCFL backlighting is on
onoff=0, CCFL backlighting is off

```
Var:  onoff
```

Reference: sl_ccfl.c

---

void neg_24(char onoff) sl.h

Turns the –24V on/off with HV5.
onoff=1, -24V is on
onoff=0, -24V is off

```
Var: onoff
```

Reference: sl_bird.c, sl_grid.c

---

void put_grid(unsigned char kpad) sl.h

Displays a grid on the LCD matching the touchscreen keypad.

`Var: kpad`

Reference: sl_grid.c

---

unsigned char scan_touch_screen(void) sl.h

Scans the 10x7 touchscreen (10 columns x 7 rows). Refer to Appendix F for details.
Returns a HEX value, 0xCR, which represents Row and Column coordinates as follows:

C = 1 to A
R = 1 to 7

Returns HEX values for the 14 push buttons located at the lower edge of the SmartLCD:

BUTTON	1	2	3	4	5	6	7	8	9	10	11	12	13	14
HEX	B0	C0	B1	C1	B6	B7	B2	C2	B3	C3	C4	B4	B5	C5

If any key or keys are activated while the function is executing, the returned value will be for the first key activated. If none of the touchscreen keys or pushbuttons is activated while the function is being executed, the function will return zero (0).

Reference: sl_grid.c

---

void beep(int t, int l) sl.h

Toggles P26 to drive the beeper.

```
void beep (     )
{
for (j=0; j<t; j++){
   pio_wr(26,0);
   delay0(l);
   pio_wr(26,1);
   delay0(l);
   }

   Var:  t, l
```

Reference: sl_bird.c

---

int sl_ad12(unsigned char c) sl.h

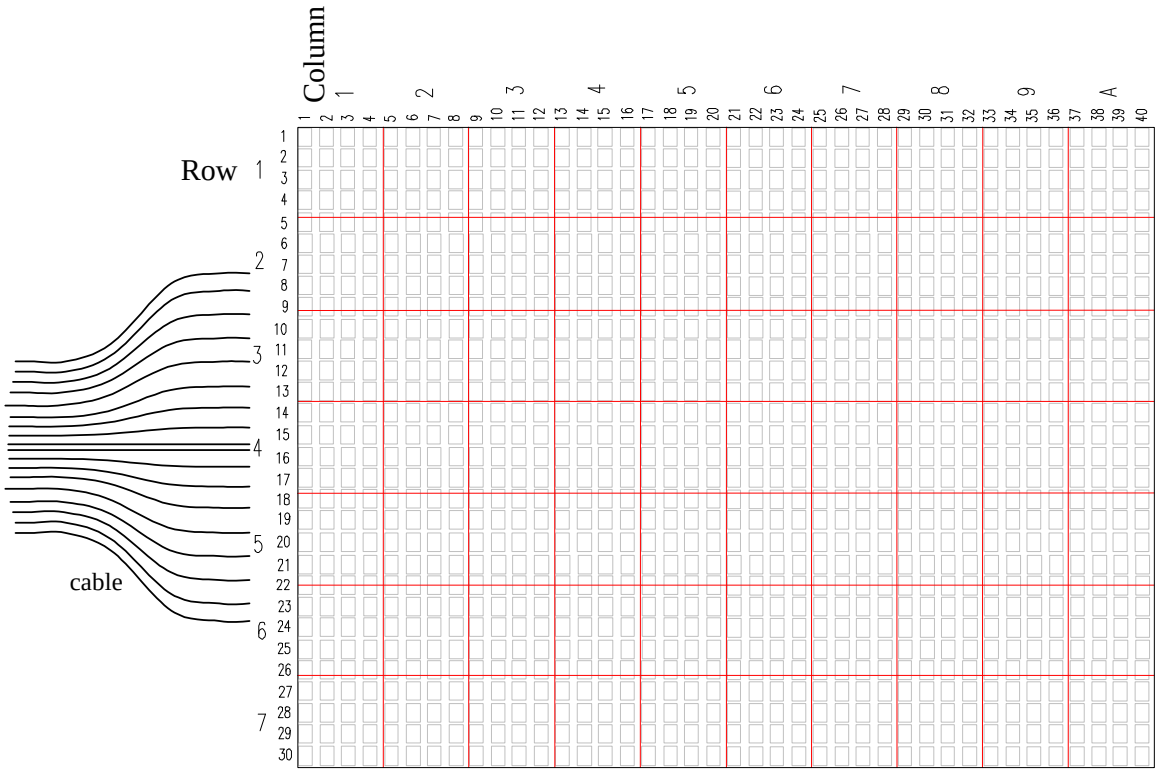
In order to operate the U29 ADC,
T22=CLK, T21=DIN, T17=/CS must be output and T00=DOUT must be input.

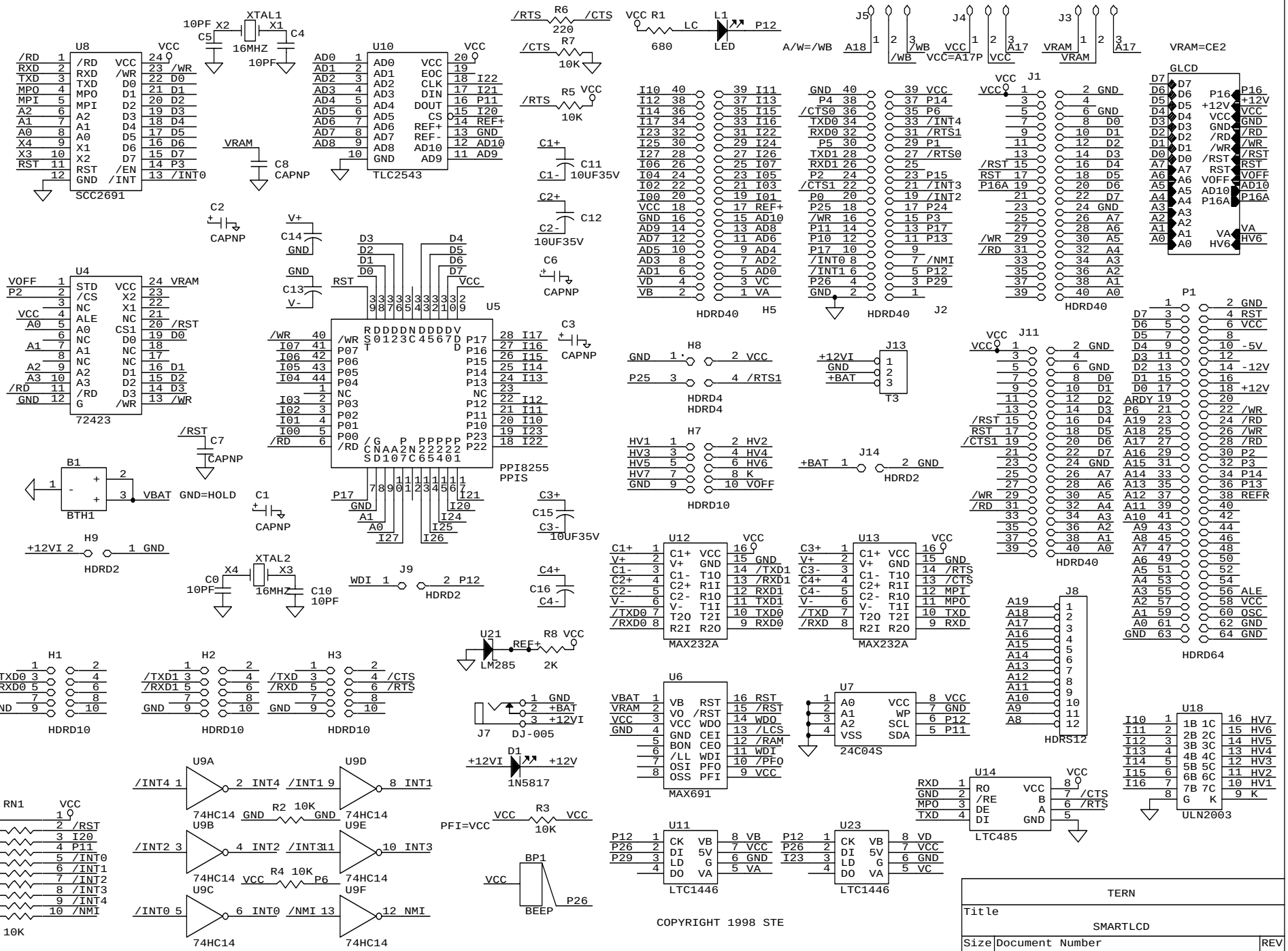
`Var: c`

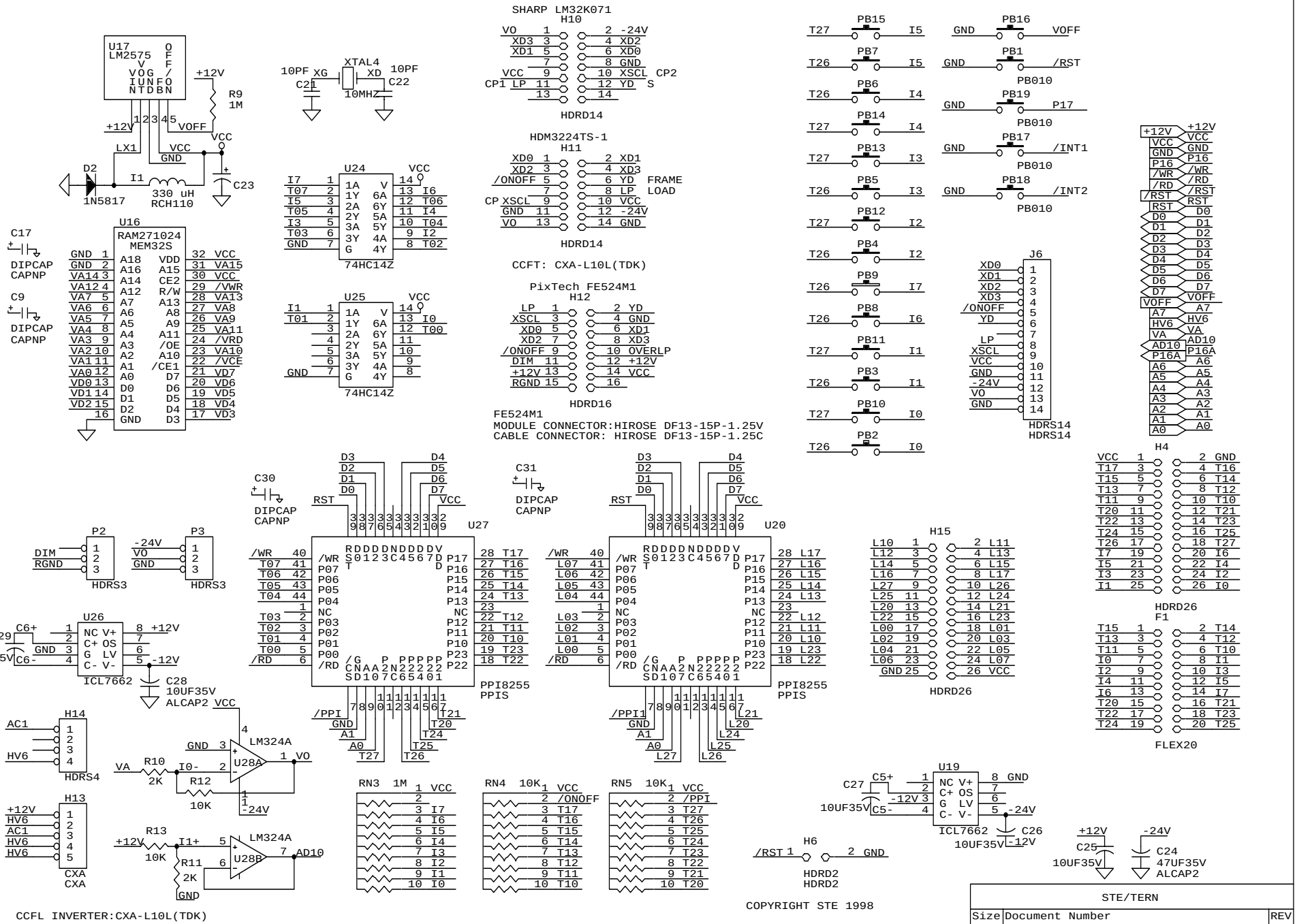
Reference: sl_ad12.c

Appendix F: Touchscreen Layout Template

10x7-key screen/keypad layout template:




U8
SCC2691
U10
TLC2543
U4
72423
U5
PPI8255
PPIS
U12
MAX232A
U13
MAX232A
U6
MAX691
U7
24C04S
U14
LTC485
U18
ULN2003
U11
LTC1446
U23
LTC1446
U21
LM285
U9A
U9B
U9C
U9D
U9E
U9F
74HC14
RN1
10K
XTAL1
16MHZ
XTAL2
16MHZ
B1
BTH1
GND=HOLD
J1
J2
J3
J4
J5
J8
J9
J11
J13
J14
J7
DJ-005
D1
1N5817
BP1
BEEP
PFI=VCC
VRAM=CE2
GLCD
HDRD40
HDRD4
HDRD10
HDRD2
HDRD64
HDRS12
H1
H2
H3
H5
H7
H8
H9
L1
LED
P1
P12
P26
A/W=/WB
VCC=A17P
COPYRIGHT 1998 STE
TERN
Title
SMARTLCD
Size
B
Document Number
SL-MAN.SCH
REV
Date: October 30, 1998
Sheet 1 of 2


U17 LM2575 VOG/IUNTD FBN
+12V
R9 1M
VOFF
VCC
LX1
D2 1N5817
I1 330 uH RCH110
C23
C17 DIPCAP CAPNP
C9 DIPCAP CAPNP
U16 RAM271024 MEM32S
GND VA14 VA12 VA7 VA6 VA5 VA4 VA3 VA2 VA1 VA0 VD0 VD1 VD2
A18 A16 A14 A12 A7 A6 A5 A4 A3 A2 A1 A0 D0 D1 D2 GND
VDD A15 CE2 R/W A13 A8 A9 A11 /OE A10 /CE1 D7 D6 D5 D4 D3
VCC VA15 VCC /VWR VA13 VA8 VA9 VA11 /VRD VA10 /VCE VD7 VD6 VD5 VD4 VD3
XTAL4
10PF XG C21 XD 10PF C22 10MHZ
U24 74HC14Z
I7 T07 I5 T05 I3 T03 GND
1A 1Y 2A 2Y 3A 3Y G
6A 6Y 5A 5Y 4A 4Y 4Y
I6 T06 I4 T04 I2 T02
U25 74HC14Z
I1 T01
1A 1Y 2A 2Y 3A 3Y G
6A 6Y 5A 5Y 4A 4Y 4Y
I0 T00
SHARP LM32K071
H10
VO XD3 XD1 VCC CP1 LP
-24V XD2 XD0 GND XSCL CP2 YD S
HDRD14
HDM3224TS-1
H11
XD0 XD2 /ONOFF CP XSCL GND VO
XD1 XD3 FRAME LOAD YD LP VCC -24V GND
HDRD14
CCFT: CXA-L10L(TDK)
PixTech FE524M1
H12
LP XSCL XD0 XD2 /ONOFF DIM +12V RGND
YD XD1 XD3 OVERLP +12V VCC GND
HDRD16
FE524M1
MODULE CONNECTOR:HIROSE DF13-15P-1.25V
CABLE CONNECTOR: HIROSE DF13-15P-1.25C
PB15 T27 I5
PB7 T26 I5
PB6 T26 I4
PB14 T27 I4
PB13 T27 I3
PB5 T26 I3
PB12 T27 I2
PB4 T26 I2
PB9 T26 I7
PB8 T26 I6
PB11 T27 I1
PB3 T26 I1
PB10 T27 I0
PB2 T26 I0
PB16 GND VOFF
PB1 GND /RST
PB010
PB19 GND P17
PB010
PB17 GND /INT1
PB010
PB18 GND /INT2
PB010
J6
XD0 XD1 XD2 XD3 /ONOFF YD LP XSCL VCC GND -24V VO GND
1 2 3 4 5 6 7 8 9 10 11 12 13 14
HDRS14
HDRS14
+12V VCC GND P16 /WR /RD /RST RST D0 D1 D2 D3 D4 D5 D6 D7 VOFF A7 HV6 VA AD10 P16A A6 A5 A4 A3 A2 A1 A0
H4
VCC T17 T15 T13 T11 T20 T22 T24 T26 I7 I5 I3 I1
GND T16 T14 T12 T10 T21 T23 T25 T27 I6 I4 I2 I0
HDRD26
F1
T15 T13 T11 I0 I2 I4 I6 T20 T22 T24
T14 T12 T10 I1 I3 I5 I7 T21 T23 T25
FLEX20
C30 DIPCAP CAPNP
C31 DIPCAP CAPNP
P2 DIM RGND HDRS3
P3 -24V VO GND HDRS3
U27
D3 D2 D1 D0 RST
D4 D5 D6 D7 VCC
/WR T07 T06 T05 T04 T03 T02 T01 T00 /RD
/WR P07 P06 P05 P04 NC P03 P02 P01 P00 /RD
RDDDDNDDDDV S0123C4567T D /G P CNAA2N22222 SD107C65401 PPPPP
P17 P16 P15 P14 P13 NC P12 P11 P10 P23 P22
28 T17 27 T16 26 T15 25 T14 24 T13 23 22 T12 21 T11 20 T10 19 T23 18 T22
/PPI GND A1 A0 T27 T26
7 8 9 0 1 2 3 4 5 6 7 T21 T20 T24 T25
PPI8255 PPIS
U20
D3 D2 D1 D0 RST
D4 D5 D6 D7 VCC
/WR L07 L06 L05 L04 L03 L02 L01 L00 /RD
/WR P07 P06 P05 P04 NC P03 P02 P01 P00 /RD
RDDDDNDDDDV S0123C4567T D /G P CNAA2N22222 SD107C65401 PPPPP
P17 P16 P15 P14 P13 NC P12 P11 P10 P23 P22
28 L17 27 L16 26 L15 25 L14 24 L13 23 22 L12 21 L11 20 L10 19 L23 18 L22
/PPI1 GND A1 A0 L27 L26
7 8 9 0 1 2 3 4 5 6 7 L21 L20 L24 L25
PPI8255 PPIS
H15
L10 L12 L14 L16 L27 L25 L20 L22 L00 L02 L04 L06 GND
L11 L13 L15 L17 L26 L24 L21 L23 L01 L03 L05 L07 VCC
HDRD26
U26 ICL7662
C29 10UF35V C6+ GND C6-
NC V+ C+ OS G LV C- V-
+12V -12V
C28 10UF35V ALCAP2
H14 HDRS4
AC1 HV6
H13 CXA
+12V HV6 AC1 HV6 HV6
LM324A U28A
VA R10 2K I0- R12 10K GND VO -24V
LM324A U28B
+12V R13 10K I1+ R11 2K GND AD10
CCFL INVERTER:CXA-L10L(TDK)
RN3 1M 1 VCC 2 3 I7 4 I6 5 I5 6 I4 7 I3 8 I2 9 I1 10 I0
RN4 10K 1 VCC 2 /ONOFF 3 T17 4 T16 5 T15 6 T14 7 T13 8 T12 9 T11 10 T10
RN5 10K 1 VCC 2 /PPI 3 T27 4 T26 5 T25 6 T24 7 T23 8 T22 9 T21 10 T20
H6 /RST 1 2 GND HDRD2 HDRD2
U19 ICL7662
C27 10UF35V C5+ C5- -12V
NC V+ C+ OS G LV C- V-
GND -24V
C26 10UF35V -12V
+12V C25 10UF35V
-24V C24 47UF35V ALCAP2
COPYRIGHT STE 1998
STE/TERN
Size B
Document Number SL-MAN.2
REV
Date: October 30, 1998
Sheet 2 of 2