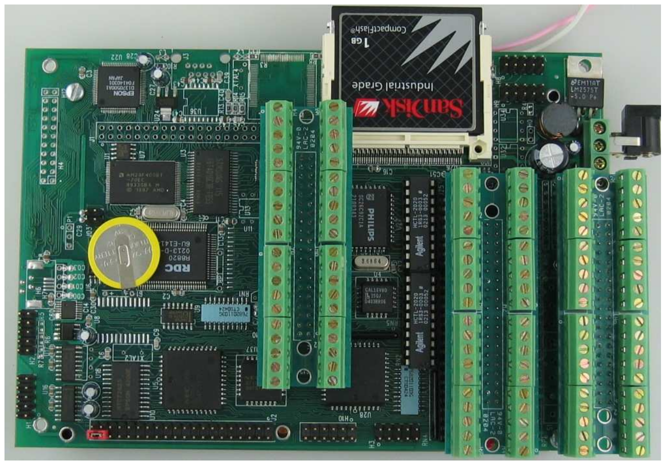
SmartTFT™

1/4 VGA Color Graphic TFT Controller with Compact Flash, 10-BaseT,
5 UARTS, 16-bit ADC / DAC based on 186 16-bit CPU and S1D13075 LCD controller



Technical Manual

TERN INC.
1724 Picasso Avenue, Davis, CA 95616, USA
Tel: 530-758-0180 Fax: 530-758-0181
Email: sales@tern.com *http://www.tern.com*

Version 1.01

October 25, 2004

1724 Picasso Avenue, 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.

Chapter 1: Introduction

1.1 Functional Description

The ***SmartTFT (ST)*** is a low cost industrial controller ideal for OEM applications. Those with premium graphic requirements will benefit from the ultra-bright, wide viewing angle active TFT color display. An LCD controller (S1D13075, EPSON) has internal 80KB image buffer, supporting QVGA color graphic TFT/LCD with 320x240 pixels. All components are installed on single PCB which is mounted on the backside of the QVGA LCD for easy integration into user applications.

Comprehensive, user-friendly software libraries and samples are provided. User can easily design their custom functions keys, text, logos, and graphics. The ***ST*** supports 50-pin CompactFlash cards for removable mass data storage. TERN software supports Linear Block Address Mode and Windows compatible FAT16 flash file system. The CF interface allows for 20 impressive color images to be displayed per second via DMA transfer from the CompactFlash card to the image buffer.

The 40 MHz, 16-bit 186 CPU with 256KW flash, 256 KW battery-backed SRAM is C/C++ programmable. The ***ST*** is designed to be mounted on the back side of a Kyocera QVGA TFT/LCD for easy integration. In addition, the ***ST*** also supports 16-bit ADC/DACs, RS232/485/422 ports, timer/counters, RTC, opto-couplers, relays, quadrature decoders, 4-20 mA drivers, Ethernet, and CompactFlash. All features can be installed simultaneously on the same PCB. Up to four ***W40S*** can be used to access I/O signals to provide 160 field removable industrial screw terminals.

Two DMA-driven serial ports from the CPU support high-speed, reliable serial communication at a rate of up to 115,200 baud, configured as RS-232. 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. A UART SCC2692 is on-board (providing two additional RS-232 ports; one optional to RS-458/422), for a total of five serial ports.

There are six 16-bit programmable timers/counters and a watchdog timer. Three of the timers/counters are supplied by the CPU, two of which 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, 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 third of the three CPU timers can be used as a pre-scale input to the two other timers. There are an additional three 16-bit hardware Programmable Interval Counters (U37, 82C54) which support high speed external event counting in multiple modes without software overhead.

The 32 I/O pins (PIO) from the CPU are multifunctional and user-programmable. Some of the I/O pins are reserved for serial ports, timer I/Os, or clocks. There can be 15 or more lines free for application use, depending on optionally installed hardware.

To interface to a ¼ VGA, 320x240 pixel color graphic LCD, an LCD controller (S1D13075, SMOS) with internal 80KB image RAM buffer can be installed. The 186 CPU communicates with the S1D13075 via high speed 16-bit data bus. Software drivers and sample programs are available for applications requiring both graphics and text display. Power supplies for the CCFL back lighting can be controlled by software for low power consumption in portable battery applications.

A 4-wire transparent resistive touch screen can be adhered to the front of the 320x240 graphic LCD. The touch screen controller (ADS7846) provides the interface between the touch screen and CPU. A sample program "slc_grid.c" and "slc_cali.c" can be used for calibrating and testing the touch screen.

Two optional 16-bit ADC chips (U14 ADS8344 and U12 AD7655) can be installed. The 8 channel serial ADC (ADS8344, 10 KHz, 0-5V, U14) provides sampling at 10 KHz and a synchronous serial interface to the CPU. It provides 8 single-ended or 4 differential inputs. TERN software drivers support 8 single-ended inputs by default but can be easily modified for differential inputs. A high-speed 16-bit parallel 4 channel ADC (AD7655, U12) can be installed. This ADC offers sampling rates at 1MHz and provides 0-5V input range with a 2.5V precision reference (U13, LT1019). The AD7655 contains two low noise, high bandwidth track-and-hold amplifiers that allow *simultaneous* sampling on two channels. Each track-and hold amplifier incorporates a multiplexer to provide a total of 4 channels analog inputs. The parallel ADC achieves very high throughput by requiring only two CPU I/O operations (one start, one read) to complete a 16-bit ADC reading.

Up to three DAC chips can be installed on-board supporting a total of 14 channels. The DAC (DAC7612, 10 KHz U15) provides 2 channels of 12-bit, 0-4.095V analog voltage outputs capable of sinking or sourcing 5 mA. The second DAC (DAC7625, U11, 200 KHz) provides a 12-bit parallel interface to four channels with analog output of 0-2.5V. These four channels are routed to J4 pins 21-24. A third serial DAC (LTC2600, 0-5V, 10 KHz) can provide 8 channels of 16-bit analog output. These analog outputs are routed to J4 pins 13-20.

Three chips (ULN2003, U25, U33, U38) are on-board providing twenty-one solenoid drivers, each capable of sinking 350mA at 50V. U25 is driven by output pins from DUART (U27, SC26C92), U33 is driven by U28 PPI Port1 (L10-L17) and U38 is driven by U5 OOI port1 (I10-I17). U33 can be re-configured for 7 high voltage inputs. Optional two sourcing drivers (UDS2982) can be installed. There are 4 mechanical relays with contact ratings of 200V at 0.5 Amps, and 12 opto-couplers for handling high voltage with opto-isolation. Two channels of 4-20 mA current drivers support many types of industrial devices. Two quadrature decoders (HTLC2020) can be used for motion position feedback.

A 10-baseT Ethernet LAN controller (CS8900) can be installed. Software stack library is available, supporting network protocols such as ARP, DHCP, UDP, ICMP, and of course TCP over the Ethernet network.

A supervisor chip (U6, MAX691) is installed to provide power failure detection, reset and a watchdog timer. An optional real-time clock provides information on the year, month, date, hour, minute, second, and an interrupt signal. On-board switching regulator (LM2575) allows 8-30V unregulated DC power input without generating excess heat. For additional expansion capabilities, two ***ST***s can be stacked with one ***ST*** responsible for managing the TFT display and the second used for I/O.

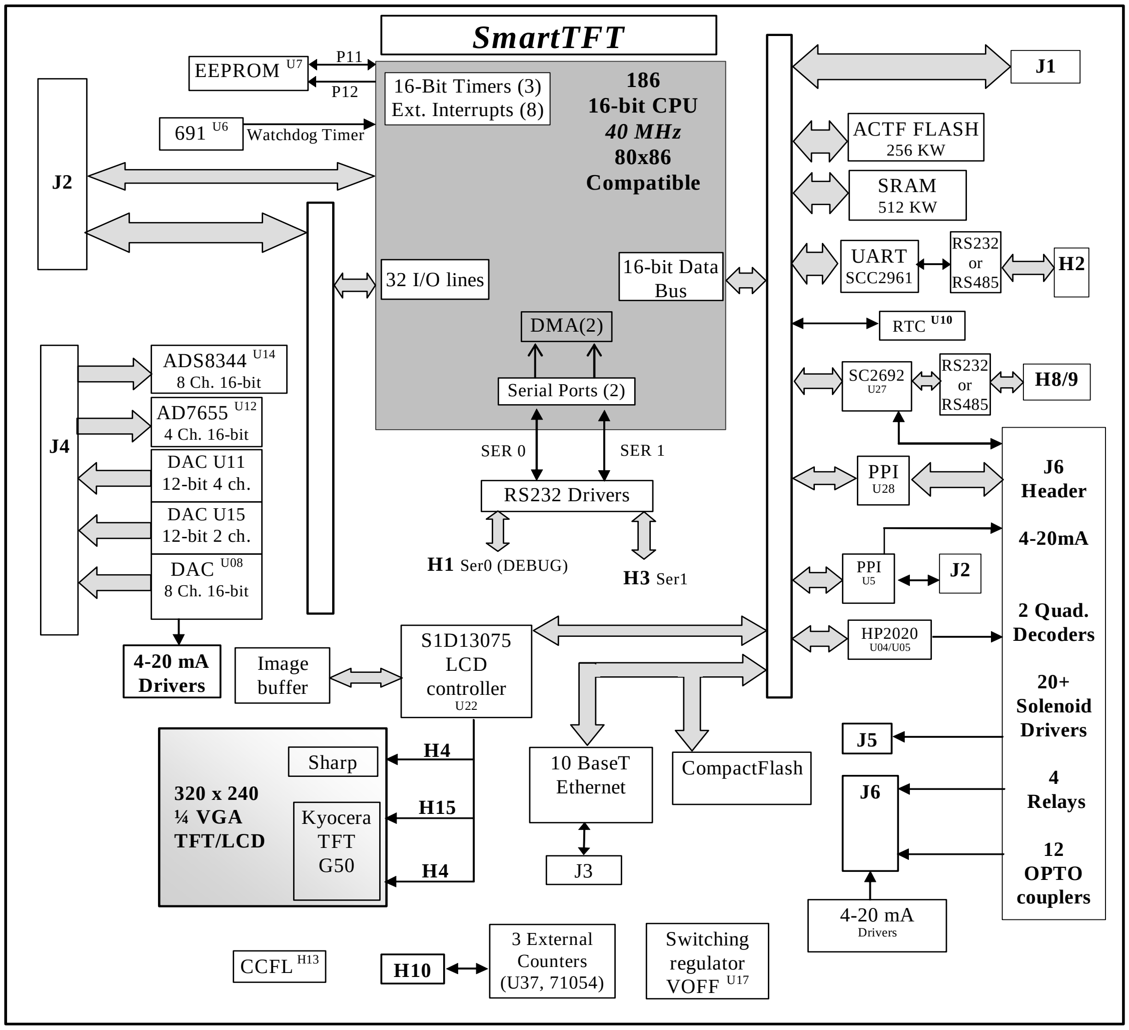


Figure 1.1 Functional block diagram of the SmartLCD

Features:

* 6.5 x 4.5 inches, 40 MHz 186 CPU, program in C/C++
* Switching regulator, 50 mA standby, 160 mA at 12V
* 256 KW SRAM, 256 KW Flash, 512 bytes EEPROM
* 20+ TTL I/O, 6 16-bit timer/counters, RTC, Battery
* 4 mechanical relays, 12 Opto-coupler inputs.
* 21 solenoid drivers, sinking or sourcing
* 12 ch. 16-bit ADCs, 14 ch. 16/12-bit DACs
* Precision reference and on-board temperature sensor.
* Multi-vendor's QVGA LCD/TFT support
* 4-wire analog touch screen controller
* 5 RS232/RS485 serial ports, 10-baseT Ethernet with TCP/IP
* CompactFlash with Windows compatible file system support
* 2 ch. Quadrature decoders, 2 ch. 4-20 mA outputs
* 160 field removable industrial screw terminals.

1.2 Physical Layout

The following two diagrams describe the physical layout of the SmartTFT.

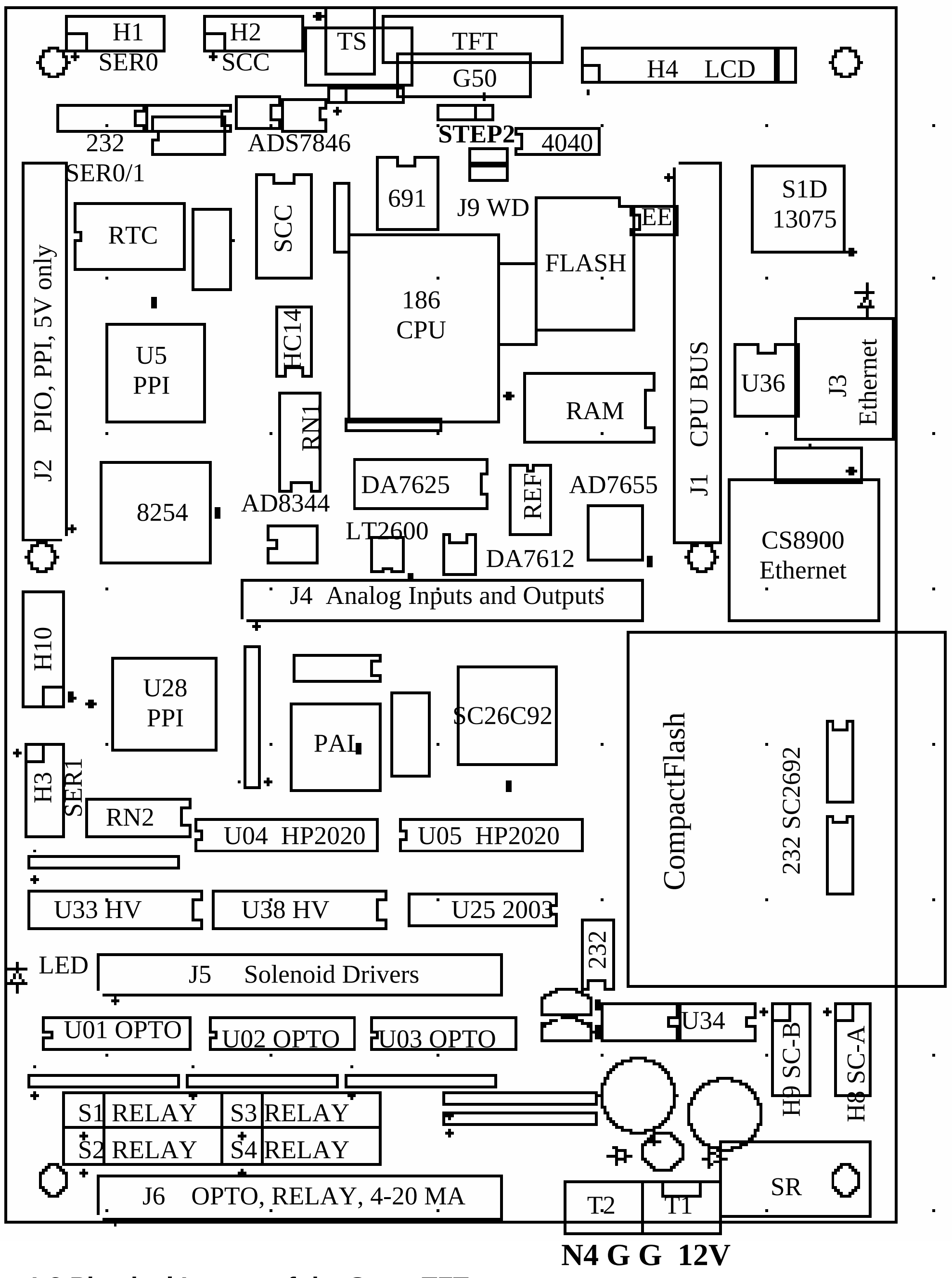


Figure 1.2 Physical Layout of the SmartTFT

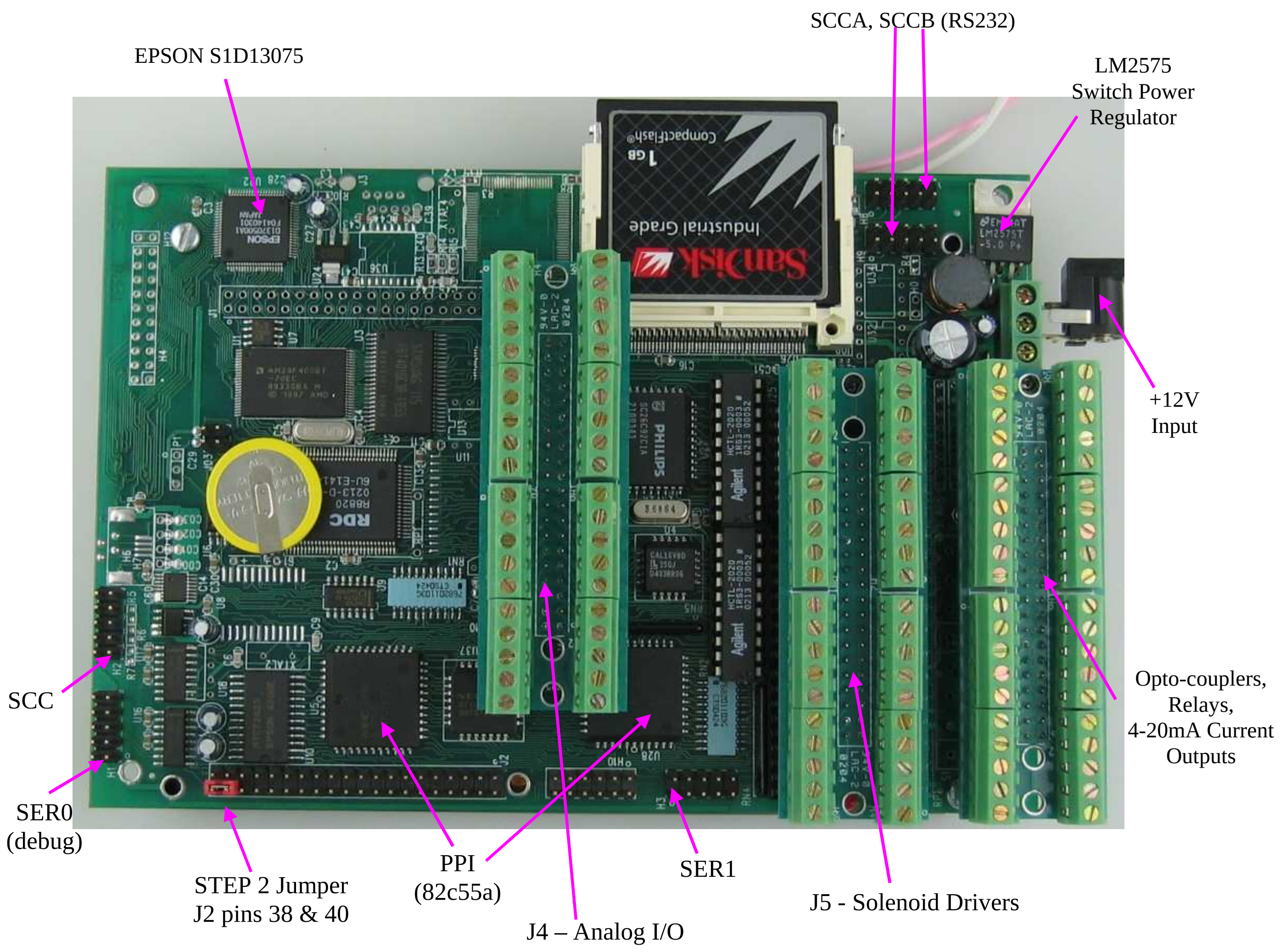


Figure 1.2 Physical Layout of the SmartTFT

1.3 SmartTFT Programming Overview

An "ACTF Boot Loader" resides in the top protected sector of the 256KW on-board flash chip (29F400). At power-on/reset, the ACTF Utility will check the STEP 2 jumper (J2 pins 38 & 40). If the STEP 2 jumper is installed, the "jump address" located in the on-board serial EEPROM will be read out and the CPU will jump to that address for immediate execution. A DEBUG kernel (already pre-programmed at the factory) can be downloaded and programmed into the flash starting at address 0xFA000. Using the ACTF Utility, the "GFA000 <enter>" command will set the jump address to 0xFA000. The command will also run the DEBUG kernel, preparing the ST for communication with the Paradigm C/C++ IDE for downloading and debugging applications. The following diagrams show the procedure for programming the ST. Steps include preparing the ST for debugging, debugging the ST, standalone field test, and production.

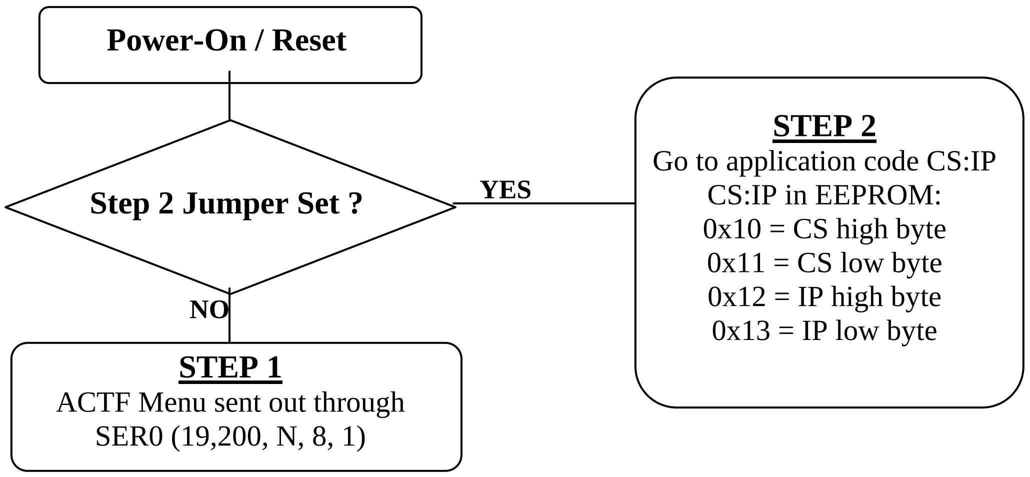


Figure 1.3 Flow Chart of ACTF Operation

By default, the DEBUG kernel has been loaded into the ACTF flash at the factory for your convenience. You may proceed directly to STEP 1: Debugging.

Preparation for Debugging:

This had already been done at the factory! You may proceed to STEP 1: Debugging. This step is only required if you have completed STEP 3 and would like to return to STEP 1.

● Connect the ST (SER0, H1) to PC (COMx) via serial debug cable provided with the EV-P/DV-P. Using the Windows "Hyper Terminal", create a serial link based on 19,200, 8 bits, 1 stop, no parity.
● Power on the ST WITHOUT the STEP 2 jumper installed (J2 pins 38 & 40). The ACTF text MENU should be sent out via serial port to "Hyper Terminal".
● Use the "D <enter>" command to initiate download. Select Transfer -> Send File, and select \tern\186\rom\ae86\l_tdrem.hex. Use the "G04000 <enter>" command to execute this script.
● Select Transfer -> Send File to select \tern\186\rom\ae86\ae86_115.hex. This is the debug kernel. Use the "GFA000 <enter>" command to set jump address and execute the debug kernel. The LED will blink twice and remain on.
● Set the STEP 2 jumper (J2 pins 38 & 40). The ST is now ready to communicate with the Paradigm C/C++ IDE for debugging and application development.

Step 1: Debugging:

● Launch the Paradigm C/C++ IDE. Select File -> Open. Chose the project file \tern\186\samples\st\st.ide.
● Use samples within the "st.ide" project to create application. Download, run, and debug application.

Step 2: Standalone Field Test:

● After completing STEP 1, by default, your application resides in the battery-backed SRAM starting at address 0x08000.
● Remove STEP 2 jumper and setup Hyper Terminal link with ST. (Open Windows "Hyper Terminal" program. Set for 19,200, 8 bits, 1 stop, no parity).
● At power-on, ACTF menu will be sent to Hyper Terminal. Use the "G08000 <enter>" command to execute application. Set STEP 2 jumper. At every power-on/reset, application at 0x08000 will execute.
● Complete STANDALONE FIELD TEST. If return to STEP 1 is required, remove STEP 2 jumper and use the "GFA000 <enter>" command to run debug kernel to prepare to setup for communication with Paradigm C/C++ IDE.

Step 3: Production:

The DV-P Kit is required for this step. If you do not have the DV-P Kit, visit http://tern.com/devkit.htm for upgrade details.

● Refer to Section 3.3 of the ACTF technical manual, found in the \tern_docs\manuals directory. Here you will find details on generating an ACTF downloadable HEX file based upon you application.
● Remove the STEP 2 jumper and create serial link using Hyper Terminal (19,200, N, 8, 1). At power-on/reset, you will see the ACTF menu at Hyper Terminal. Use the "D <enter>" command to initiate download process. Select Transfers -> Send File, and select \tern\186\rom\ae86\l_29f400.hex.
● This file will erase the flash and prepare the flash to accept ACTF downloadable application HEX file. Use the "G04000 <enter>" command to run script. Flash will be ready for application.
● Select Transfer -> Send File to select your ACTF downloadable application HEX file. Upon completion, use the "GC0000 <enter>" command to execute application. This command also sets the jump address to point you application in flash. Set STEP 2 jumper (J2 pins 38 & 40). At power-on/reset application will execute.

There is no ROM socket on the SmartTFT. The user's application program must reside in the SRAM (starting at address of 0x08000 by default based on \tern\186\config\186.cfg) for debugging in STEP 1, reside in the battery-backed SRAM for standalone field testing in STEP 2, and finally be programmed into the on-board flash for a complete product. For production, the user must produce an ACTF-downloadable HEX file for the application based on the DV-P Kit. From the ACTF Utility, use the command "GC0000 <enter>" to point to the user's application code in the flash. The STEP 2 jumper must installed for every production-version board.

1.4 Minimum Requirements for SmartTFT System Development

Minimum Hardware Requirements

- PC or PC-compatible computer with serial COMx port that supports 115,200 baud
- SmartTFT controller
- Debug Serial Cable (RS232; DB9 connector for PC COM port and IDE 5x2 connector for controller)
- Center Negative Wall Transformer

Minimum Software Requirements

- TERN EV-P installation CD-ROM and a PC running: Windows 95/98/2000/ME/NT/XP

With the EV-P, you can program and debug the ***ST*** in Step One and Step Two, but you cannot run Step Three. To generate an application Flash File and complete a project, the development kit, DV-P, is required. The EV-P kit can be upgraded to the DV-P Kit. See http://tern.com/devkit.htm for details.

Chapter 2: Installation

2.1 Software Installation

Please refer to the “software_kit.pdf” technical manual on the TERN installation CD, under tern_docs\manual\software_kit.pdf, for information on installing software.

2.2 Hardware Installation

Overview

- Connect PC-IDE serial cable:
  For debugging (STEP 1), place IDE connector on SER0 with red edge of cable at pin 1. This DEBUG cable is a 10-pin IDE to DB9 cable, made by TERN.
- Connect wall transformer:
  Connect 9V wall transformer to power and plug into power jack using power jack adapter supplied with EV-P/DV-P Kit

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

2.2.1 Connecting the ST to the PC

The following diagram (Fig 2.1) provides the location of the debug serial port and the power jack. The ***ST*** is linked to the PC via a serial cable (DB9-IDE) which is supplied with TERN’s EV-P / DV-P Kits.

The ***ST*** communicates through SER0 by default. Install the 5x2 IDC connector on the SER0 H1 5x2 pin header. ***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).

2.2.2 Powering-on the ST

By factory default setting:
1) The RED STEP2 Jumper is installed. (Default setting in factory)
2) The DEBUG kernel is pre-loaded into the on-board flash starting at address of 0xFA000. (Default setting in factory)
3) The EEPROM is set to jump address of 0xFA000. (Default setting in factory)

Connect +9-12V DC to the DC power terminal. The screw terminal at the corner of the board is positive 12V input and the other terminal is GND (see figure for details). A power jack adapter (seen below) is included with the TERN EV-P/DV-P kit. It can be used to connect the output of the power jack adapter and the ***ST***. Note that the output of the power jack adapter is center negative.

The on-board LED should blink twice and remain on, indicating the debug kernel is running and ready to communicate with Paradigm C++ TERN Edition for programming and debugging.

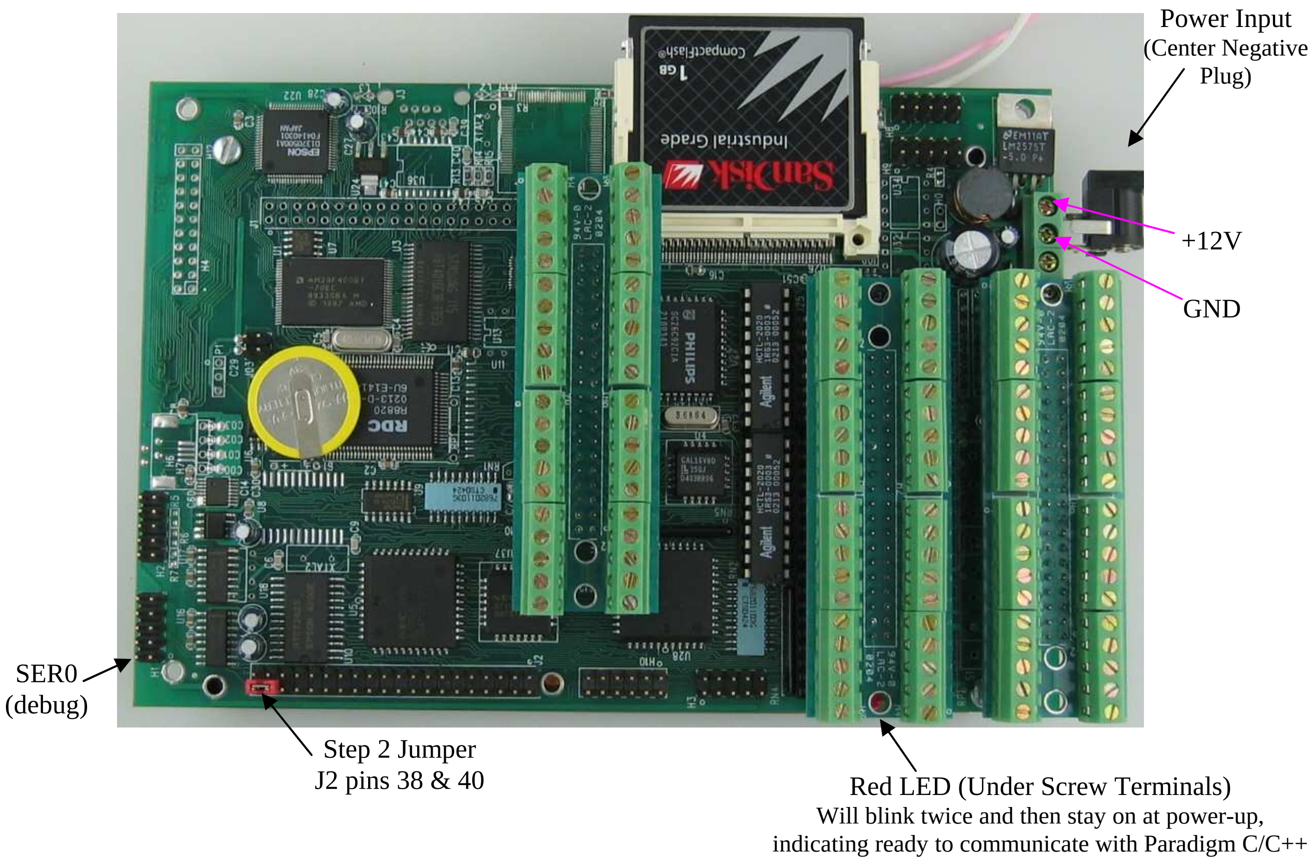

Power Input
(Center Negative Plug)
+12V
GND
SER0 (debug)
Step 2 Jumper
J2 pins 38 & 40
Red LED (Under Screw Terminals)
Will blink twice and then stay on at power-up, indicating ready to communicate with Paradigm C/C++

Figure 2.1 Locations of STEP2 Jumper, LED, Power input and DEBUG port

Chapter 3: Hardware

3.1 CPU 186 – Introduction

The 186 host processor is based on industry-standard x86 architecture. The 186 controllers are higher-performance, more integrated versions of the 80C188 microprocessors. In addition, the 186 has new peripherals. The on-chip system interface logic can minimize total system cost. The 186 has two asynchronous serial ports, 16-bit external data bus, 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 CPU 186 – Features

3.2.1 Clock

The 186 microcontroller achieves 40 MHz CPU operation, while using a 40 MHz crystal.

The CPU's system CLKOUTA signal is routed out to J1 pin 4, at 40 MHz.

CLKOUTA remains active during reset and bus hold conditions. The SmartTFT 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, used by SCC2691 UART, if installed.
/INT1, used by SCC2692 UART, if installed.
/INT2, used by ADS7843 touch screen controller.
INT3, J2 pin 21, used by CS8900 Ethernet controller if installed.
/INT4, J2 pin 33, buffered by 74HC14, free to use.
INT5=P12=DRQ0, used as output for LED/EE/HWD
/INT6=P13=DRQ1, J2 pin 11, buffered by 74HC14, free to use
/NMI, J2 pin 7, buffered by 74HC14, free to use

Five external interrupt inputs, /INT0-2,4 and /NMI, are buffered by Schmitt-trigger inverters (74HC14, U9), 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 ***ST*** uses vector interrupt functions to respond to external interrupts. Refer to the Am186ES User's manual for information about interrupt vectors (from the root of the installation CD, \amd_docs\am186es, "am186es_user_manual.pdf").

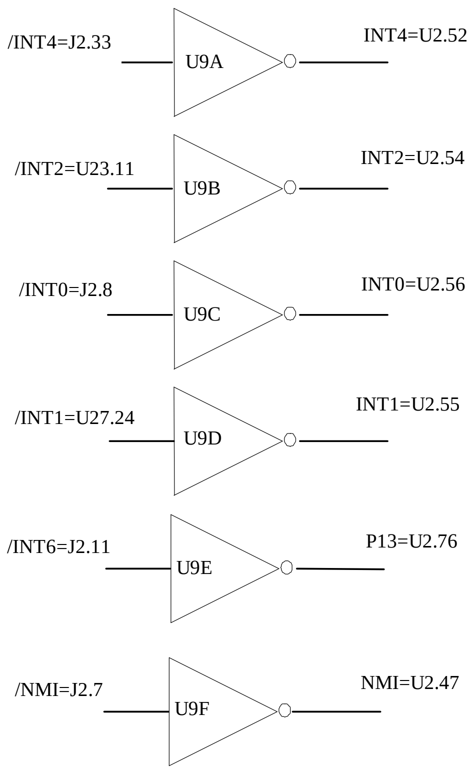


Figure 3.1 External interrupt inputs with Schmitt-trigger inverters

3.2.3 Asynchronous Serial Ports

The 186 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 at U8. For more information about the external UART SCC2691, please refer to the datasheet of the SCC2691 (\tern_docs\parts\scc2691.pdf), section 3.11 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 three external pins:

Timer0 output = P10 = J2 pin 12
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. Timer0 input is P11, which is shared by other system components and is not recomended for use as a timer input.

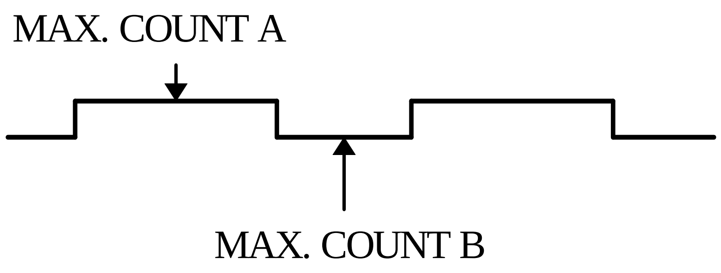
Timer2 is not connected to any external pin. It can be used as an internal timer for real-time coding or time-delay applications. It can also prescale timer 0 and timer 1 or be used as a DMA request source.

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 **`tern\186\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 system clock).

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.



3.2.6 Power-save Mode

The ***ST*** can be used for low power consumption applications.

The power-save mode of the 186 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 ***ST*** has a VOFF signal routed to H0 pin 1. 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 line. More details are available in the sample file ***poweroff.c*** in the **`\tern\186\samples\ae`** sub-directory. In the Power-off mode, the VOFF pin is pulled high via a 1M ohm resistor, and the switching regulator will be turnned off. While in the power-off mode, less than 1 mA overall current consumption can be achieved. Any external signal can short the VOFF pin to GND, to "wake up" from the power-off mode. Upon "wake up", it is the application's responsibility to drive the VOFF line low, to keep the switching regulator turned on. When a power down is required, the RTC can be programmed to put the VOFF line into tri-state, which will then be pulled high via external pull-up resistor, turing the switching regulator off. The high power consumption CCFL backlighting power can be controlled by software with the on-board relay S4 via T2 screw terminals.

3.3 186 PIO lines

The 186 CPU includes 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*	*ST Pin Number*	*ST 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, or J03.1	Input with pull-up **Step 2**
P5	/DEN/DS	Normal	J2 pin 30	Input with pull-up
P6	SRDY	Normal	S1D13075	Input with pull-down*
P7	A17	Normal		A17
P8	A18	Normal		A18
P9	A19	Normal	ADS7843	A19=P9=/INT2
P10	Timer0 out	Input with pull-down	J2 pin 12	Input with pull-down
P11	Timer0 in	Input with pull-up	EE	Input with pull-up
P12	DRQ0/INT5	Input with pull-up		Output for LED/EE/HWD
P13	DRQ1/INT6	Input with pull-up	J2 pin 11=/INT6	Input with pull-up
P14	/MCS0	Input with pull-up	S1D13075	LCD controller chip select
P15	/MCS1	Input with pull-up	J2 pin 23	Input with pull-up
P16	/PCS0	Input with pull-up	J1 pin 19	chip select
P17	/PCS1	Input with pull-up		PAL chip select
P18	CTS1/PCS2	Input with pull-up	74HC138	Input with pull-up
P19	RTS1/PCS3	Input with pull-up	PPI U28	Input with pull-up
P20	RTS0	Input with pull-up	ADS8344	Input with pull-up
P21	CTS0	Input with pull-up	ADS7843	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	AD7655	Input with pull-up*
P27	TxD1	Input with pull-up	RS232 driver	TxD1
P28	RxD1	Input with pull-up	RS232 driver	RxD1
P29	/CLKDIV2	Input with pull-up	DA7612	Input with pull-up*
P30	INT4	Input with pull-up	J2 pin 33=/INT4	Input with pull-up
P31	INT2	Input with pull-up	ADS7843	Input with pull-up

Note: *P6, 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 for SCC2691, /INT0=J2.8
INT1 for SC26C92
INT3 for CS8900

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

SmartTFT 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 ***ST*** 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 INT0, 1, 2, 4 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.

Signal	**Pin**	**Function**
P1	Timer1 output	Beeper
P2	/PCS6	U10 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
P5	J2 pin 30	Used by touch screen controller
P7	A17	Upper address line. Never use in application.
P8	A18	Upper address line. Never use in application.
P9	A19	Used in interface to touch screen controller.
P11	Timer0 input	U7 24C04 EE data input
P12	DRQ0/INT5	Output for LED, CLK for U7 EE, U15 DAC, Hit watchdog
P14	/MCS0	LCD controller chip select
P16	/PCS0	J1 pin 19=/PCS0, chip select free to use for expansion
P17	/PCS1	PAL16CEV8 (U4), U4 pin 9
P18	/PCS2=/CTS1	74HC138 decoder chip select
P19	/PCS3	U28 PPI chip select
P20	RTS0	U14 ADS8344 ADC data input

Signal	Pin	Function
P21	/CTS0	Used a serial data output from touch screen controller
P22	TxD0	Default SER0 debug
P23	RxD0	Default SER0 debug
P24	J2 pin 17	CLK input to touch screen controller
P25	/MCS3	DAC7612U (U15) data input
P26	UZI	AD7655 mux select
P29	/CLKDIV2	U15 DAC7612U DAC data latch
P31	INT2	Output from touch screen controller
INT0	J2 pin 8	U8 SCC2691 UART interrupt.
INT1	J2 pin 6	U27 SCC2692 DUART interrupt.
INT3	J2 pin 21	U35 CS8900 interrupt

Table 3.2 I/O lines used for on-board components

3.4 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 (inport**b**/outport**b** transfer one byte) or one word (inport/outport transfer two bytes) 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 Am186ES 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 Am186ES User's Manual (amd_docs\am186es directory).

The table below shows more information about I/O mapping. Devices such as the UART SCC2691, UART SCC2692, the CS8900 ethernet controller, Real Time Clock, ADCs, DACs, and the LCD controller are options on the ST. If any of these devies are not installed on the ST, the /PCSx line(s) will then become free for application use, otherwise the /PCSx line must be reserved for the corresponding chip select.

I/O space	Select	Location	Usage
0x0000-0x00ff	/PCS0	J1 pin 19=P16	User
0x0100-0x01ff	/PCS1	U4 pin 9=P17	PAL16V8CE (U4) input line
0x0100-0x0106	/PPI	U5 pin 7	PPI82C55(U5)
0x0110-0x0116	/DA	U11 pin 23	U11 DAC7625
0x0120-0x0126	/AD	U12 pin 32	U12 AD7655
0x0140-0x0146	/SC	U27 pin 39	U27 SCC2692
0x01c0-0x01df	/ET	U35 pin 63	U35 CS8900
0x01e0-0x01ef	/CF	U26 pin 7, 32	U26 CompactFlash
0x0200-0x02ff	/PCS2	U06 pin 4=CTS1	74HC138
0x0200-0x206	/CNT	U37 pin 24	71054 counters
0x0220	/CV	U12 pin 35	AD7655 start convert
0x0280	/HP1	U05 pin 4	HTLC2020 Quadrature decoder
0x02A0	/HP2	U04 pin 4	HTLC2020 Quadrature decoder
0x02C0	/RT1	U05 pin 7	RESET quadrature decoder

0x02E0	/RT2	U04 pin 7	RESET quadrature decoder
0x0300-0x03ff	/PCS3	U28 pin 7 = RTS1	PPI82C55 (U28)
0x0400-0x04ff	/PCS4		Reserved
0x0500-0x05ff	/PCS5	J2 pin 15 = P3	UART, SCC2691
0x0600-0x06ff	/PCS6	J2 pin 24 = P2	RTC 72423

3.5 S1D13075

The SID13075 is a color/monochrome LCD graphics controller with an embedded 80K Byte SRAM display buffer. The LCD controller can achieve up to 20 frames per second and support 256 colors. A unique aspect of the ***ST*** is the CompactFlash storage working in concert with this LCD controller. With this deisgn, the 186 can DMA images directly from the CompactFlash into the image buffer to achieve higher performance. Because of the 80KB image buffer, the function call *slc_init();* re-defines the upper half of the memory map to accommodate this buffer. After calling *slc_init();* the memory map will be as in the following diagram: **(diagram not to scale)**

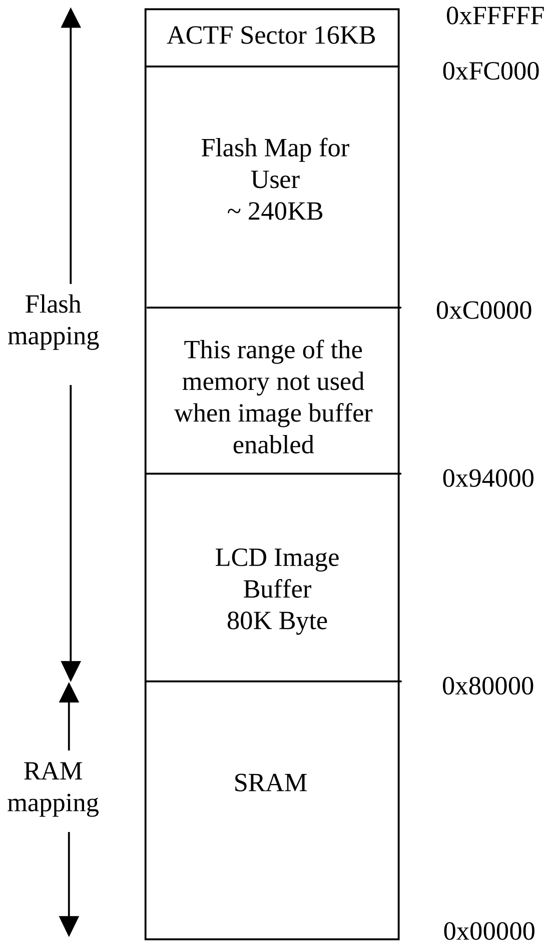

ACTF Sector 16KB
Flash Map for User ~ 240KB
This range of the memory not used when image buffer enabled
LCD Image Buffer 80K Byte
SRAM
0xFFFFF
0xFC000
0xC0000
0x94000
0x80000
0x00000
Flash mapping
RAM mapping

Figure 3.2 Memory Map after *slc_init();* to accommodate Image Buffer

Sample code has been provided in the **tern\186\samples\st** directory to illustrate the process of tranfering an image from the Compact Flash into the Image Buffer. It is mandatory that slc_init(); be called so as to re-initialize the memory map to accommodate the image buffer. Note that the entire ROM/Flash mapping is 512KB, but the re-mapping of the ROM/Flash only allows to configure to certain sizes. Thus the map available for the user then becomes 256KB, starting from 0xC0000 and going to 0xFFFFF, yet the ACTF utility occupies the top 16KB sector, again reducing the usable Flash to the range 0xC0000 to 0xFC000, or about 240KB. It is possible to map the Image Buffer into the SRAM, leaving all 512KB of Flash space for the user. This would require the user to re-define the /LMCS line used to select the SRAM. Refer to the AM186ES manual chapter 5 for details on how to re-map the SRAM. This re-mapping of the Image Buffer into the SRAM would also require the DMA process from the Compact Flash to the LCD controller to be altered. Refer to sample code in the **\tern\186\samples\st** directory.

3.6 Touch Screen Controller

The ADS7846E is a 12-bit sampling ADC with a synchronous serial interface and low on-resistance switches for driving touch screens. The ADS7846E is routed to a 4-pin terminal at H6 which connects to a flax cable to drive the touch screen. This controller allows the user to specify the touch screen resolution needed a particular application. See sample program that allows for calibration of touch sreen in **\tern\186\samples\st** directory. The sample program is “st_grid.c”. This sample is already included in the pre-made project “st.ide” in the **\tern\186\samples\st** directory.

3.7 Compact Flash Interface

By utilizing the compact flash interface on the ***ST***, users can easily add widely used 50-pin CF standard mass data storage cards to their embedded application via RS232, TTL I2C, or parallel interface. TERN software supports Linear Block Address mode, 16-bit FAT flash file system, RS-232, TTL I2C or parallel communication. Users can write files to the CompactFlash card or read files from the CompactFlash card. Users can also transfer files to a PC via the CF reader port.

CF cards can also be used as a means to store images and data to be displayed onto the color LCD. This allows users to have access to unlimited images to be used in an application in conjunction with the color LCD. As dicussed above, the AM186ES suuports DMA to allow images/data to be transferred directly to the image buffer for increased speed.

Sample code and function prototypes are available to assist in creating applications which use the file system to access the CF. Refer to the target “slc_fs.axe” within the pre-built project “st.ide” found in the \tern\186\samples\st directory. This sample uses the source code \tern\186\samples\fn\fs_cmds1.c. Also, for a complete listing of file system function prototypes and data types, refer to the header files “fileio.h” and “filegeo.h” found the **\tern\186\include** directory.

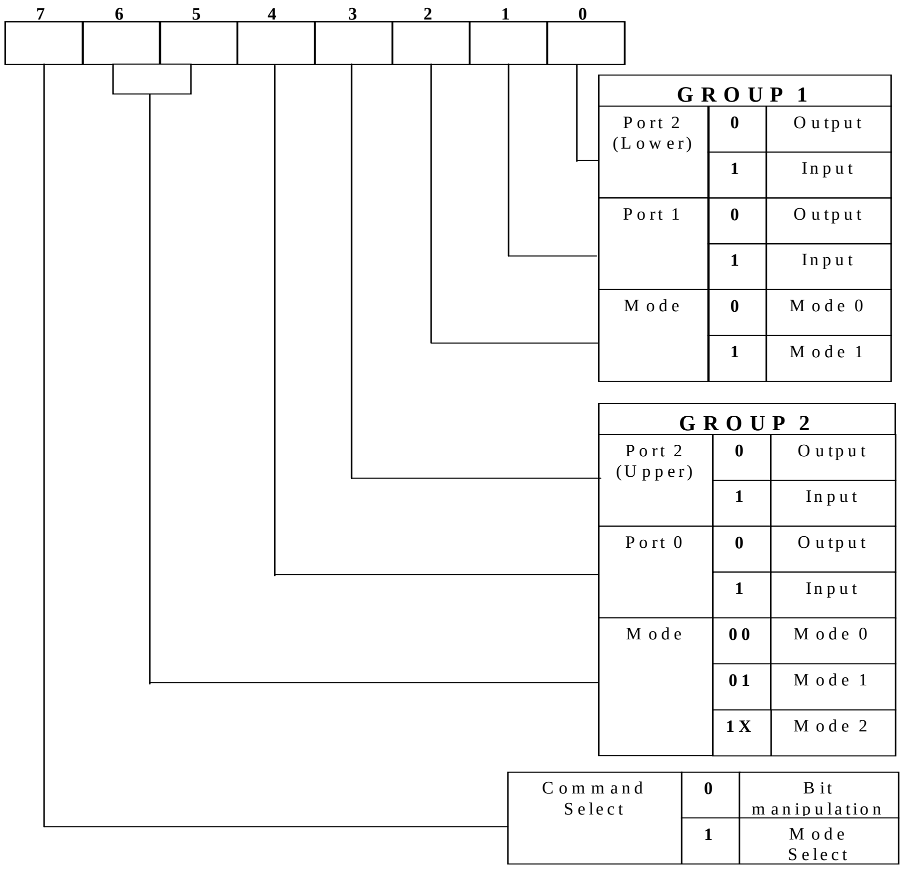
3.8 Ethernet

The Ethernet LAN Controller on the SmartTFT is the CS8900 from Crystal Semiconductor Corporation, which is now a subsidiary of Cirrus Logic (http://www.cirrus.com/en/). The CS8900 includes on-chip RAM and 10BASE-T transmit and receive filters. The CS8900 directly interfaces to the TERN controller’s data bus, providing a high-speed, full duplex operation. The ***ST*** interface to the Ethernet port is via a standard RJ45 8-pin connector (J3). The CS8900 offers a broad range of performance features and configuration options. Its unique PacketPage architecture automatically adapts to changing network traffic patterns and available system resources. The CS8900-based ST can increase system efficiency and minimize CPU overhead in a 10BASE-T network. The ***ST*** with CS8900 provides a true full-duplex Ethernet solution, incorporating all of the analog and digital circuitry needed for a complete C/C++ programmable Ethernet node controller.

3.9 Programmable Peripheral Interface (82C55A)

U5 and U28, PPIs (82C55) are low-power CMOS programmable parallel interface unit for use in microcomputer systems. It provides 24 I/O pins each, 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 4 remaining pins, 3 are used for handshaking and interrupt control signals. MODE 2 is a strobed bi-directional bus configuration.



7 6 5 4 3 2 1 0
GROUP 1
Port 2 (Lower) 0 Output 1 Input
Port 1 0 Output 1 Input
Mode 0 Mode 0 1 Mode 1
GROUP 2
Port 2 (Upper) 0 Output 1 Input
Port 0 0 Output 1 Input
Mode 00 Mode 0 01 Mode 1 1X Mode 2
Command Select 0 Bit manipulation 1 Mode Select

Figure 3.3 Mode Select Command Word

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

The SmartTFT maps U28, the 82C55/uPD71055, at base I/O address 0x300.

Note that registers increment by 2.

Use U5 PPI as a program example, the Command Register = 0x0106; Port 0 = 0x0100; Port 1 = 0x0102; and Port 2 = 0x0104.

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

```
    outportb(0x0106,0x80); /* Mode 0 all output selection. */
    outportb(0x0100,0x55); /* Sets port 0 to alternating high/low I/O pins. */
    outportb(0x0102,0x55); /* Sets port 1 to alternating high/low I/O pins. */
    outportb(0x0104,0x55); /* Sets port 2 to alternating high/low I/O pins. */
```

To set all ports to input mode:

```
    outportb(0x0106,0x9f);     /* Mode 0 all input selection. */
```

You may read the ports with:

```
inportb(0x0100); /* Port 0 */
inportb(0x0102); /* Port 1 */
inportb(0x0104); /* 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. The PPI data sheet can be found under the root of the TERN installation CD, \tern_docs\parts with filename “82c55a.pdf”. You may also see the sample code **\tern\186\samples\slc\slc_ppi.c**.

3.10 Real-time Clock RTC72423

If installed, the real-time clock RTC72423 (EPSON, U10) 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_rds()*** (see Appendix C and the Software chapter for details). A sample program is provided **\tern\186\samples\slc\slc_rtc.c**. Its data sheet can be found in the \tern_docs\parts directory, filename “rtc7242xam.pdf”.

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***.

3.11 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 SmartTFT may be used as a RS485 network 9-bit UART (for the TERN NT-Kit). Its data sheet and sample code are **\tern_docs\parts\scc2691.pdf** and **\tern\186\samples\slc\slc_scc.c**, respectively.

3.12 Reed Relays

Four Reed relays are installed on the ST at locations S1-S4. The relays offer high speed swithing compared to electromechanical relays, a specification of 200 V, maximum 1 Amp carry current, 0.5 Amp switching, and 100 million times operation. The relays are connected to I24-I27 of the PPI at location U5. See tern\186\samples\slc\slc_ppi.c and \tern_docs\parts\relay9007.pdf for details.

3.13 UART SCC2692

The dual UART (SC26C92, Philips, U27) is a 44-pin PLCC chip. This DUART includes two independent full-duplex asynchronous receiver/transmitters, 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.

A 3.6864 MHz external czrystal is installed as the default crystal for the dual UART.

For more detailed information, refer to the SC26C92 data sheets on the CD in the **tern_docs\parts** directory, filename “scc2692.pdf”.

By default, RS-232 drivers are provided for the DUART. The RS-232 signals are routed to the H8 (TXDA and RXDA) and H9 (TXDB and RXDB) headers. Optional RS422 or RS485 drivers can be installed for TXDA and RXDA.

Sample programs for the SC26C92 can be found in the **`c:\tern\186\samples\slc`** directory, “slc_scc.c”.

3.14 Programmable Timer/Counter (NEC PD71054)

The NEC PD71054 Programmable Timer/Counter (PTC) chip provides three high-performance, programmable counters for the SmartTFT. Three 16-bit counters, each with its own clock input, gate control, and output pins, can be clocked from DC to 10 MHz.

Under software control, they can generate accurate timer delays. There are six programmable count modes. All PTC related pins are routed to H10. Refer to TERN’s CD-ROM for ST schematics for more details. Sample programs are available under **\tern\186\samples\slc**, “slc_cn0.c” and “slc_cn3.c”. The samples have been included in the sample project “st.ide”, in the **\tern\186\samples\st** directory.

3.15 Quadrature Decoders (HTLC2020)

Two quadrature decoder/counter interface chips, (Hewlett Packard, HCTL2020, U04 and U05) can be installed on the ***ST***. The quadrature decoder is used to interface incremental motion encoders with the microprrocessor system or to improve system performance for digital closed-loop motion control systems. The HCTL2020 includes a quadrature decoder, a 16-bit counter, and an 8-bit bus interface. It features full 4x decoding, up to 14MHz clock operation, high noise immunity due to digital noise filters, quadrature decoder output signals, up/down signals, count signals, and cascade output signal. Many types of optical incremental encoder modules, such as the HEDS-9000, HEDS-9100, and HEDS-9200 from HP, can be directly interfaced the the HCTL2020.

The quadrature inputs are routed to the J5 pin header, pins 37-40, for access. Sample code is provided, “st_hp.c” in the **\tern\186\samples\st** directory. It is also included in the pre-built project “st.ide”.

3.16 Other Devices

A number of other devices are also available on the SmartTFT. 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.

3.16.1 On-board Supervisor with Watchdog Timer

The MAX691/LTC691 (U6) is a supervisor chip. With it installed, the SmartTFT 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 SmartTFT. 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 SmartTFT 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 Am186ES has an internal watchdog timer. This is disabled by default with **ae_init()**.

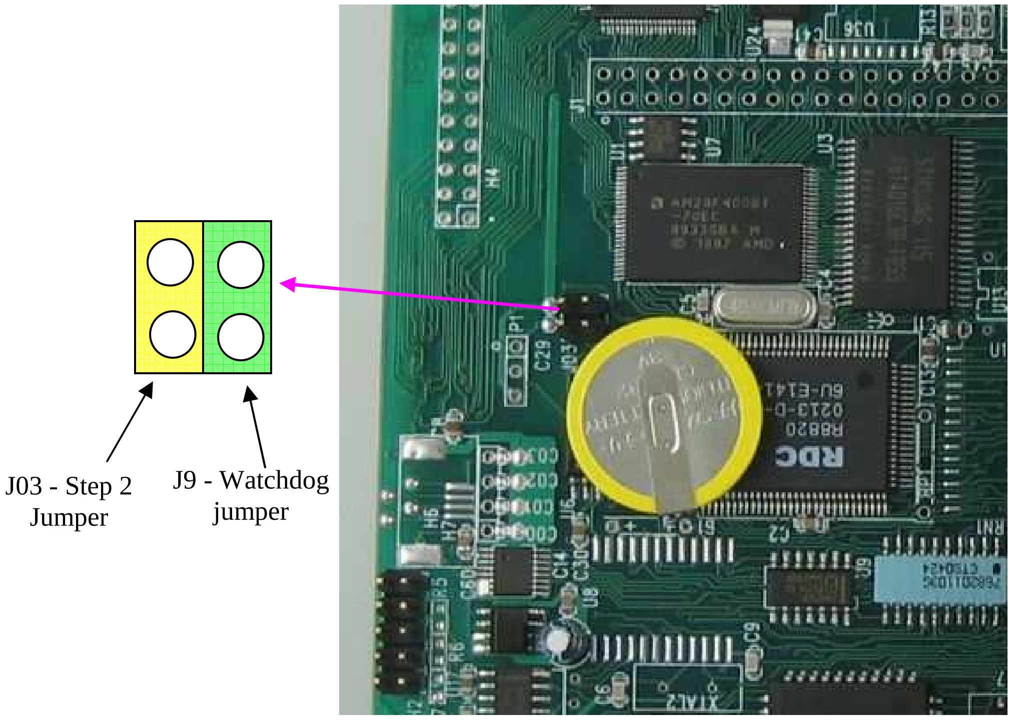


Figure 3.4 Location of STEP2 and watchdog timer enable jumpers

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 about 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.

3.17 EEPROM

A serial EEPROM of 512 bytes (24C04) is installed in U7. The SmartTFT 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 the functions **ee_rd()** and **ee_wr()**.

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.

3.18 Opto-couplers

Twelve opto-couplers are installed on the SmartTFT, providing high voltage opto-isolation capability. They can be used for digitals inputs, relay contact monitoring, or powerline monitoring. Typical ON time is 3µs, while typical OFF time is 5µs. The corresponding input pins are pulled high via 10kΩ resistor. As a result, an input low signal will turn the opto-couplers on. The opto-couplers are arranged in three packages, 4 opto-isolators per package. The three packages (PS2701-4) are located at U01, U02, and U03. The device at location U01 uses input lines IP2 – IP5 of the SCC2692 Dual UART. Thus to read the status of the opto-couplers at U01, the user must check the status of the input port on the SCC2692. Similarly, the devices at U02 and U03 are tied to signals, L27 – L20, port 3 of the U28 PPI chip. To monitor the status of these opto-couplers, configure port 3 of the U28 PPI chip to input mode and check that status of that port. More details are provided in Chatper 4: Software of this manual. The data sheet for the opto-coupler package can be found in **\tern_docs\parts\ps2701.pdf**, from the root of the TERN installation CD-ROM.

3.19 16-bit ADC (ADS8344)

The ADS8344 is an 8 channel, 16-bit sampling analog-to-digital converter with a synchronous serial interface. Input voltage range goes from 0V to 5V. The precision reference (LT1029) installed at location U07 provides the reference input to the ADS8344. Three control lines drive the ADS8344; /CS = I20, CLK = P12, and DIN = P25. DOUT is tied to the 186 CPU’s /RTS0, or P20.

It is necessary to initialize P20 = DOUT as input, I20, P12, and P25 as output to drive the ADS8834. Refer to **\tern\186\samples\st\st_ad16.c** for sample code. To inititalize I20 as output, you must program the control register of the U5 PPI.

The ADC digital data output communicates with a host through a serial tri-state output (DOUT). If /CS is low, the ADS8344 will have output on DOUT. If /CS is high, the ADS8344 is disabled and DOUT is free. The effective maximum sampling rate is 20KHz.

The ADS8344 can support 8 single-ended inputs or 4 differential inputs. By default TERN software drivers use 8 single ended inputs. This mode can be changed via the control byte written to DIN.

The ADS8344 also implements an output called BSY. When /CS is high, the BSY signal is in high-impedance. When /CS is low, BSY will be low while reading the control bits on DIN, and during conversion. This line is not connected to any external pin on the ***ST***, so a 10µs software delay must be implemented to achieve correct timing.

To operate the ADS8344, five I/O lines are used, as listed below:

/CS	Chip select = I20 (U5 PPI pin16) , high to low transition enables DOUT, DIN and CK. Low to high transition disables DOUT, DIN and CK.
DIN	P25=J2 pin 18, ADC serial data input
DOUT	Serial data output. Tied to CPU line /RTS0=P20. Needs to be initialized as input.
BSY	Output signal. BSY is low when the ADS8344 is reading the DIN control pins and during conversion. It is high impedance when /CS is high. This line is not connected to any external pin.
CLK	Clock = P12
REF+	Upper reference voltage (normally RE5=5V). J4 pin 12
COM	Ground Reference. Set to GND by default on J4 header, pin 9.
VCC	Power supply, +5 V input

GND	Ground

All analog inputs are routed to the J4 pin header. By defult, J4 pin 9 is tied to J4 pin 10 to supply COM to the ADC. Furthermore, the precision reference at location U07 provides +5V reference to REF+.

3.19.1 Temperature sensor and reference voltage

A 5V precision reference (U07 LT1029, TI) supports ADS8344 and LTC2600. A 2.5V precision voltage reference (U13, LT1019, 3 ppm/C°) is on board providing 2.5 V reference for AD7655 and DAC7625. The LT1019 includes an on-board temperature senser, which outputs temperature related voltage at J4 pin 30 (TMP). See the ***ST*** schematic at the end of the manual for additional details on pin locations.

3.20 Eight channel 16-bit DAC (LTC2600)

The TLC2600 is an eight channel 16-bit digital-to-analog converter (DAC) in an SO-8 package. It is complete with a rail-to-rail voltage output amplifier capable of driving up to 15mA. It uses a 3-wire SPI compatable serial interface and has an output range of 0-5 volts, making 1 LSB equal to 5/65535 V. On the ***ST***, the reference voltage input is tied to the precision reference +5V supplied by U07. The DAC outputs are routed to the J4 pin header, pins 13-20.

The LTC2600 DAC is installed on the ***ST*** at location U08 and uses I21 as the chip select. The synchronous serial interface is used to send data to the device. The signals SDI (synchronous data in) and SCK (synchronous clock) are tied to the CPU PIO lines P25 and P12, respectively. Refer to the sample code, **\tern\186\samples\st\st_da.c** for an example on driving the DAC. The sample is also included in the pre-built sample project \tern\186\samples\st\st.ide.

Refer to the DAC data sheet for additional specifications; \tern_docs\parts\ltc2600.pdf.

3.21 Four channel, 16-bit ADC

The unique 16-bit parallel ADC (AD7655, 0-5V) supports ultra high-speed (1 MHz conversion rate) analog signal acquisition. The AD7655 contains two low noise, high bandwidth track-and-hold amplifiers that allow *simultaneous* sampling on two channels. Each track-and hold amplifier has a multiplexer in front to provide a total of 4 channels analog inputs. The parallel ADC achieves very high throughput by requiring only two CPU I/O operations (one start, one read) to complete a 16-bit ADC reading. With a precision external 2.5V reference(U13, LT1019), the AD7655 accepts 0-5V analog inputs at 16-bit resolution, yielding 65536 counts / 5000mV = 13 lsb's / mV.

See sample program **\tern\186\samples\st\st_ad.c** for details on reading the ADC. Refer to the data sheet for additional specifications; **\tern_docs\parts\ad7655.pdf**, from the root of the TERN installation CD-ROM.

3.22 Dual Channel 12-bit DAC (DAC7612)

The DAC7612 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 DAC7612 outputs a full-scale of 4.096V, making 1 LSB equal to 1 mV.

The buffered outputs can source or sink 7 mA. The outputs swing to within a few millivolts of supply rail when unloaded. They have an equivalent output resistance of 402 Ω when driving a load to the rails. The buffer amplifiers can drive 500 pf without going into oscillation.

The DAC is installed in U15 on the SmartTFT, and the outputs are routed to J4 (pins 25=VA and pin 26=VB). The DAC uses P12 as CLK, P25 as DI, and P29 as LD/CS. Please refer to the DAC7612 technical data sheets from Texas Intruments for more information (data sheet can be found on the TERN CD-ROM in the **`\tern_docs\parts`** directory, filename **dac7612.pdf**). See also the sample program ***st_da.c*** in the **`\tern\186\samples\st`** directory.

3.23 DAC7625, 300KHz 12-bit DAC

The DAC7625 is a parallel 12-bit D/A converter. This device includes 4 voltage output channels with an output range of 0-2.5V. It accepts 12-bit parallel input data and has double-buffered DAC input logic.

The ***ST*** uses pins D15 to D4 to directly interface to the DAC's full 12-bit data bus for maximum data transfer rate. The DAC7625 has an average settling time of 5 µs, with a maximum settling time of 10µs. Refer to the sample program **slc_da.c** in the **\tern\186\samples\slc** directory for an example. The sample code is also incorporated into the pre-built project for the ***ST***, **st.ide**. Further details can be found in the data sheet, **da7625.pdf**, in the **\tern_docs\parts** directory.

3.24 4-20 mA current drivers

Two channels of 4-20 mA current drivers can be installed. Two analog voltage outputs (V7 and V8) from the 16-bit DAC (LTC2600) can be converted to 4-20 mA current. The current driver power FET (Q1, Q2) can be powered with up to 24V external positive source (J5 pin 31=V++) or on-board positive power via J01 (+12VI) header. The current driver outptuts are routed to the J5 pin header pins 33 and 34.

3.25 High-Voltage, High-Current Drivers

The ULN2003A has high voltage, high current Darlington transistor array, 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 500 mA sinking are allowed.
U25 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 J5 pin 2. The common substrate of the high-voltage driver at location U25 is tied to gournd by default. Locations U33 and U38 must have G supplied by the user. All currents sinking in must return to the J5 pin 2. A heavy gauge (20) wire must be used to connect the J5 G 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 a user provided voltage at J5 pin 3. **ULN2003 is a sinking driver, not a sourcing driver.** An example of typical application wiring is shown in Figure 3.5.

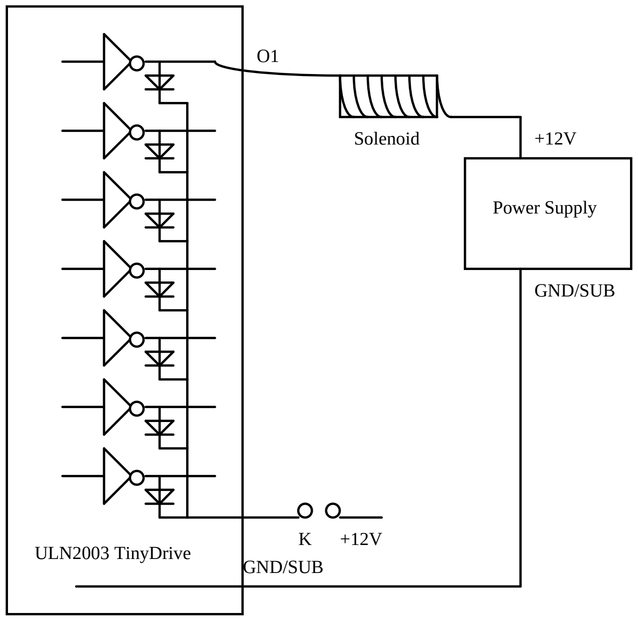


Figure 3.5 Drive inductive load with high voltage/current drivers.

By design the ULN2003 installed at U25 is configured for sinking only. Two additional 18 pin sockets are installed at U33 and U38. By design, these sockets can support sinking or sourcing drivers. By default sinking drivers are installed in these sockets. In order to correctly connect G and K to the sinking or sourcing dirvers at U33 and U38, they are routed to J5. For configuration supporting sinking drivers (ULN2003, default by factory), two jumpers must be installed to connect G to GND and to connect K to the user provided external voltage (J5 pin 1 = J5 pin 2, and J5 pin 3 = J5 pin 4). If sourcing drivers are desired, the ULN2003 must be replaced by UDS2982 (refer to tern_docs\parts from the CD for data sheet). The jumpers at J5 must then be reconfigured to route G and K to the appropriate places for sourcing (J5 pin 1 = J5 pin 3, and J5 pin 2 = J5 pin 4). U25 is driven by output pins from DUART (U27, SC26C92) and one line from the U5 PPI chip, U33 is driven by U28 PPI Port1(L00-L07) and U38 is driven by U28 PPI port1 (L10-L17).

3.26 Headers and Connectors

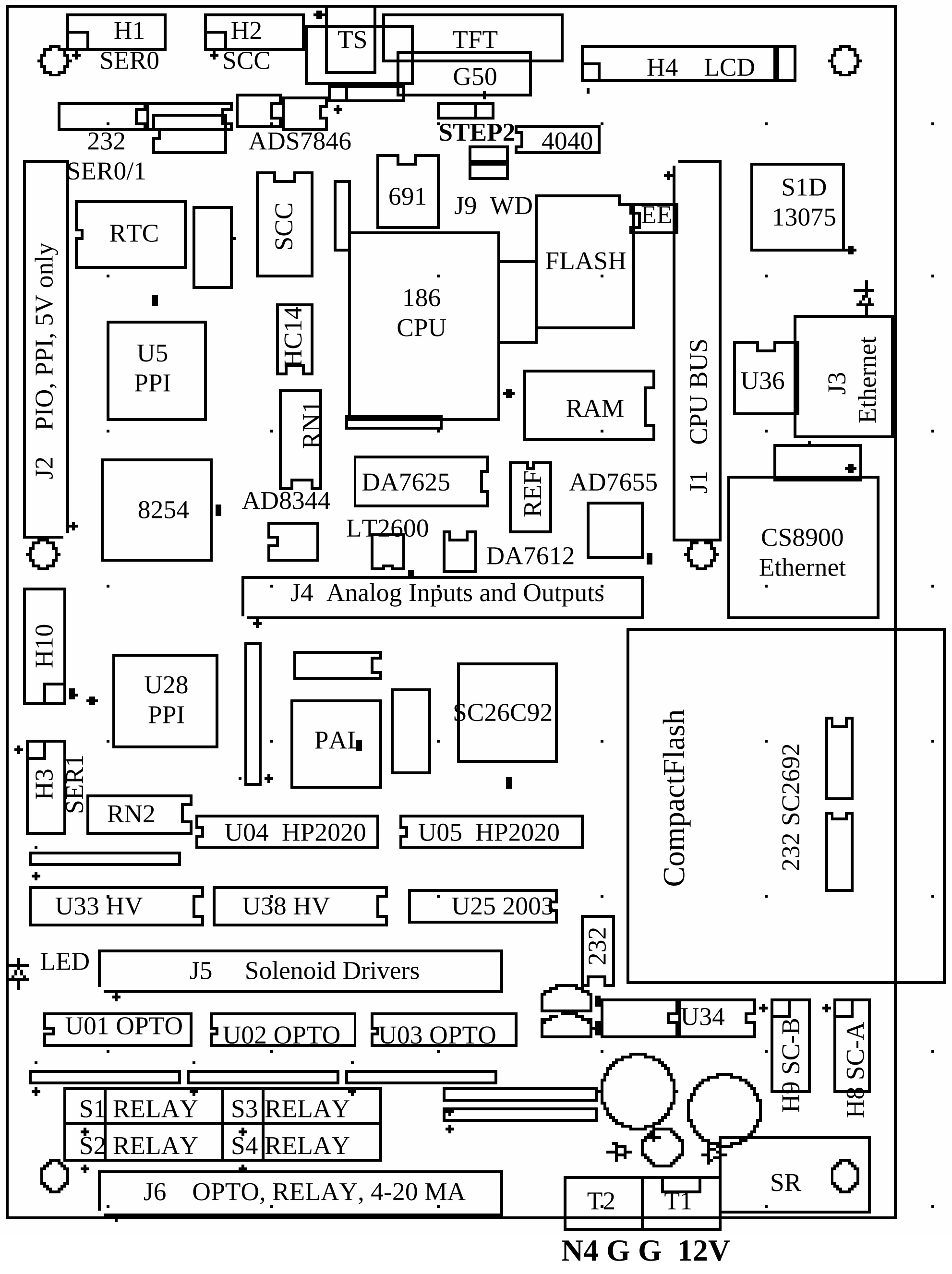


Figure 3.6 SmartTFT Headers and Connectors.

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 SmartTFT 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 SmartTFT Initialization

ae_init
This function should be called at the beginning of every program running on SmartTFT 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 AMD Am186ES Microcontroller User's manual.

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
```

tft_init vs. slc_init

The SmartTFT supports either Kyocera's QVGA TFT or Kyocera's QVGA STN LCD (G50). When using the TFT, use **tft_init()**, and if using the G50 STN LCD, use **slc_init()**. In the TERN software examples, it may be necessary to modify the source "C" code to select either tft_init(); or slc_init(); to match your hardware.

The appropriate initialization function should be called after ae_init() in every program for the SmartTFT. It will define certain I/O lines for selecting certain peripherals unique to the ST. In addition it will re-map the memory to accommodate the image buffer (see chapter 3, section 3.4.2). A function call to ae_init() is still mandatory in addition to tft_init()/slc_init().

The chip select lines are by default set to 15 wait states. 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 SmartTFT, 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 Am186ES 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 SmartTFT. 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 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 Am186ES 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 SmartTFT can be used for a variety of applications. All three timers run at 1/4 of the processor clock rate (10MHz based on 40MHz system clock, or one timer clock per 100ns), 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 Am186ES 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. It is important to note the the interrupt latency generated by the ISRs that handle a signal transition will define the time resolution the user will be able to achieve.

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 Am186ES 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

Two ADC chips can be installed on the ST.

The ADS8344 provides 8 channels of 16-bit analog conversion. It allows for an input range of 0-REF, where REF has been tied by hardware design to 5V (from the precision reference LT1029 at location U07). The negative reference, COM, is routed to J4 pin 9 and is user selectable. By deafult from the factory, COM is shorted to GND, making the input range 0-5V.

The ADS833 uses CPU PIO lines and PPI lines for serial interface. As a result, CPU PIO lines P12 and P25 must be configured as outputs, U5 PPI line I20 must be configured as output, and finally CPU PIO line /RTS0 (P20) must be configured as mode 1 input.

For a sample file demonstrating the use of the ADS8344, please see **st_ad16.c** in **`tern\186\samples\st`**.

unsigned int ad16
Arguments: unsigned char k
Return values: unsigned int ad_value

The argument **k** contains 4 types of information, including channel for conversion, power-down options, and signle-ended or differential inputs. The TERN software example uses signle-ended inputs. The resulting values for **k** become:

0x86	Channel AN0 (J4 pin 2)
0xc6	Channel AN1 (J4 pin 1)
0x96	Channel AN2 (J4 pin 4)
0xd6	Channel AN3 (J4 pin 3)
0xa6	Channel AN4 (J4 pin 6)
0xe6	Channel AN5 (J4 pin 5)
0xbf	Channel AN6 (J4 pin 8)
0xf6	Channel AN7 (J4 pin 7)

The AD7655 provides 4 high-speed analog inputs. The interface to the AD7655 uses the signals, P26, A2, /CV, and /AD. P26 and A2 are used to determine which of the four input channels is being selected, /CV is used to start the conversion, and /AD is used to select the device for a conversion read over the CPU data bus. The /AD chip select signal is generated from the PAL located at U4. As a result, the /AD signal is mapped to I/O location 0x120. The following table summarizes the channel selection:

Channel	Pin location	P26	A2	Read command
AA1	J4 pin 40	Low	High	inport(0x124)
AB1	J4 pin 34	Low	Low	inport(0x120)
AA2	J4 pin 38	High	High	inport(0x124)
AB2	J4 pin 36	High	Low	inport(0x120)

Refer to the sample code, **st_ad.c**, in the **\tern\186\samples\st** directory. It shows necessary steps to read channels on the AD7655. The sample code is also incorporated into the sample project, “st.ide”.

4.2.6 Digital-to-Analog Conversion

There are three DAC chips on the ST.

A DAC7612U chip is available on the SmartTFT in position **U15**. The chip offers two channels, A and B, for digital-to-analog conversion. Details regarding hardware, such as pin-outs and performance specifications, can be found in the Hardware chapter.

A sample program demonstrating the DAC can be found in **slc_da.c** in the directory **`tern\186\samples\slc`**.

void slc_da
Arguments: dat
Return value: none

Since the DAC6712U is a 12-bit device, the value **dat** must be ANDed with 0x0fff to select the lower 12 bits of the 16-bit value **dat**. Then **dat** must bo ORed with either 0x2000 or 0x3000 to select the channel you wish to write to. Use 0x2000 for channel A, and 0x3000 for channel B

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.

The DAC7625 is a 12-bit device at location U11. It is mapped into I/O space 0x110 – 0x116. Only one *outport(addr, data)* instruction is needed per channel, where the upper 12-bits of the 16-bit value *data* will be the value written to the device, and *addr* 0x110 corresponds to channel DA1, 0x112 to DA2, 0x114 to DA3, and 0x116 to DA4. See the sample code, **“slc_da.c”**, in the **\tern\186\samples\slc** directory.

An LTC2600 is also available on the ST. It is located at U08 and provides 8 16-bit analog outputs. It uses a serial interface with the CPU and uses signals P25, P12, and I21 for communication. Therefore, all three signals must be configured as outputs. See the sample code **“st_da.c”** in **\tern\186\samples\st** for details.

void da_16
Arguments: char c, unsigned int dat
Return value: none

The argument **c** will determine the channel to write to and **dat** will provide the data to write. Argument **c** can be defined as follows:

0x30 = V1 (J4 pin 17)
0x31 = V2 (J4 pin 18)
0x32 = V3 (J4 pin 19)
0x33 = V4 (J4 pin 20)
0x34 = V5 (J4 pin 16)
0x35 = V6 (J4 pin 15)
0x36 = V7 (J4 pin 14)
0x37 = V8 (J4 pin 13)

0x3f = Update all channels

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-P/DV-P 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 Am186ES 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 Am186ES Microprocessor User's Manual and the schematic of the SmartTFT provided on the CD in the **tern_docs\schs** directory.

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.

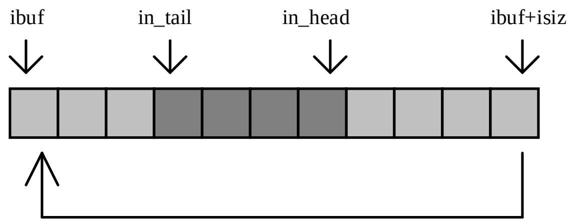


Figure 4.1 Circular ring input buffer

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 Am186ES 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 without overwrite.

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, and 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 Am186ES 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 Am186ES 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 AM186ES 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
5	1200
6	2400
7	4800
8	9600 (default)
9	19,200

Function Argument	Baud Rate
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

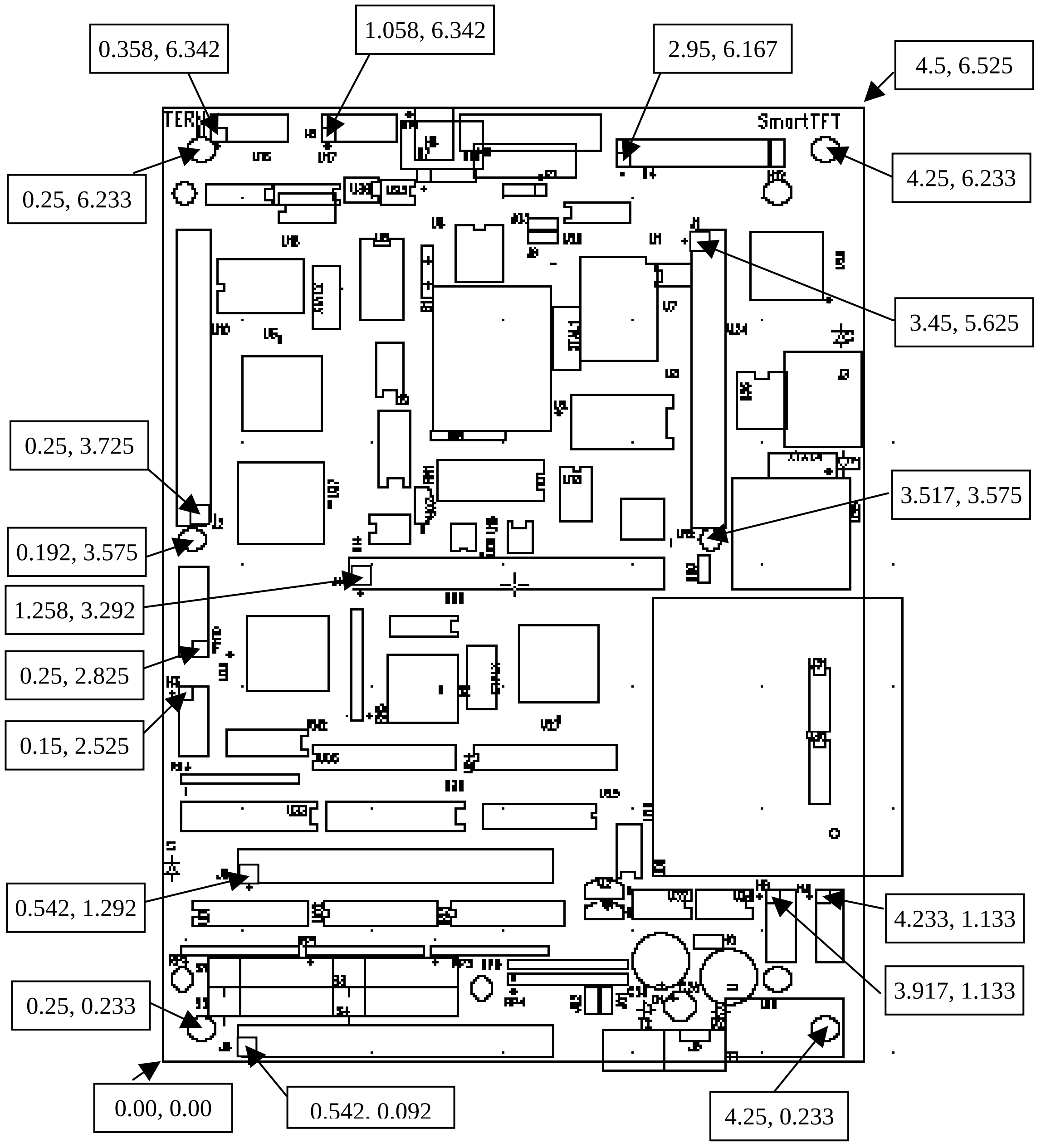
This function is used to write the passed in **dat** to the specified **addr**. The return value is 0 in success.

ee_rd
Arguments: int addr
Return value: int data

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

Appendix A: SmartTFT(ST) Mechanical Dimensions

All dimensions are in inches.



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

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 locations are used by system software. Application programs must not use these locations. The 512 bytes have hexadecimal address range of 0x00 to 0x1FF. TERN reserves the range 0x00 to 0x1F for system use. This leaves the range 0x20 to 0x1FF free for applications.

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™
0x18	MM page register 0
0x19	MM page register 1
0x1a	MM page register 2
0x1b	MM page register 3
0x20	Free for application use
.	
.	
.	
.	
.	
0x1FF	

Appendix E: Software Glossary

The following is a glossary of library functions for the SmartLCD-Color.

---

void ae_init(void) ae.h

Initializes the Am186ES 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 Am186ES 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

---

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

b	baud (40MHz)	baud (20MHz)
8	9600	4800
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. Table 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

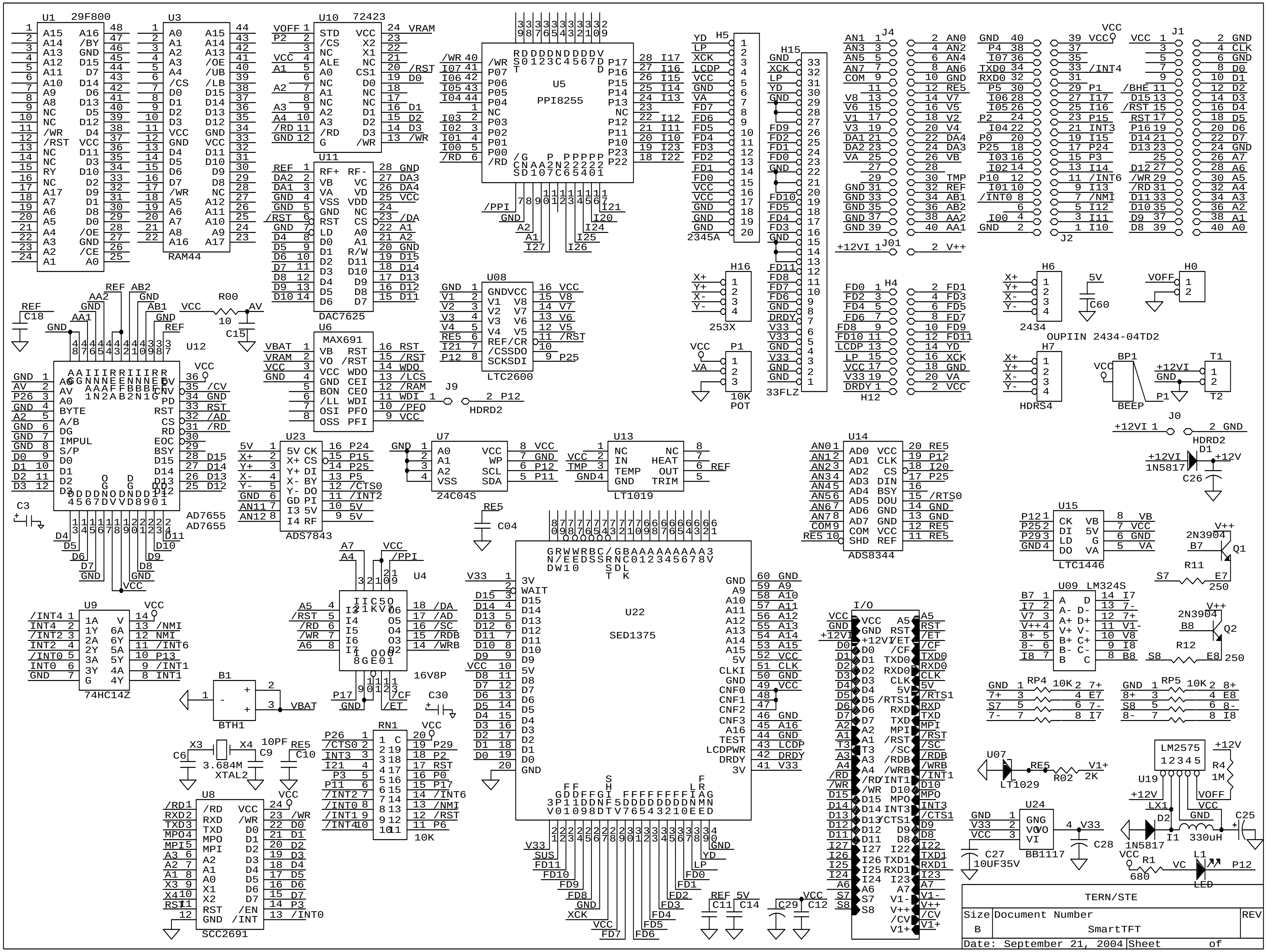
---

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

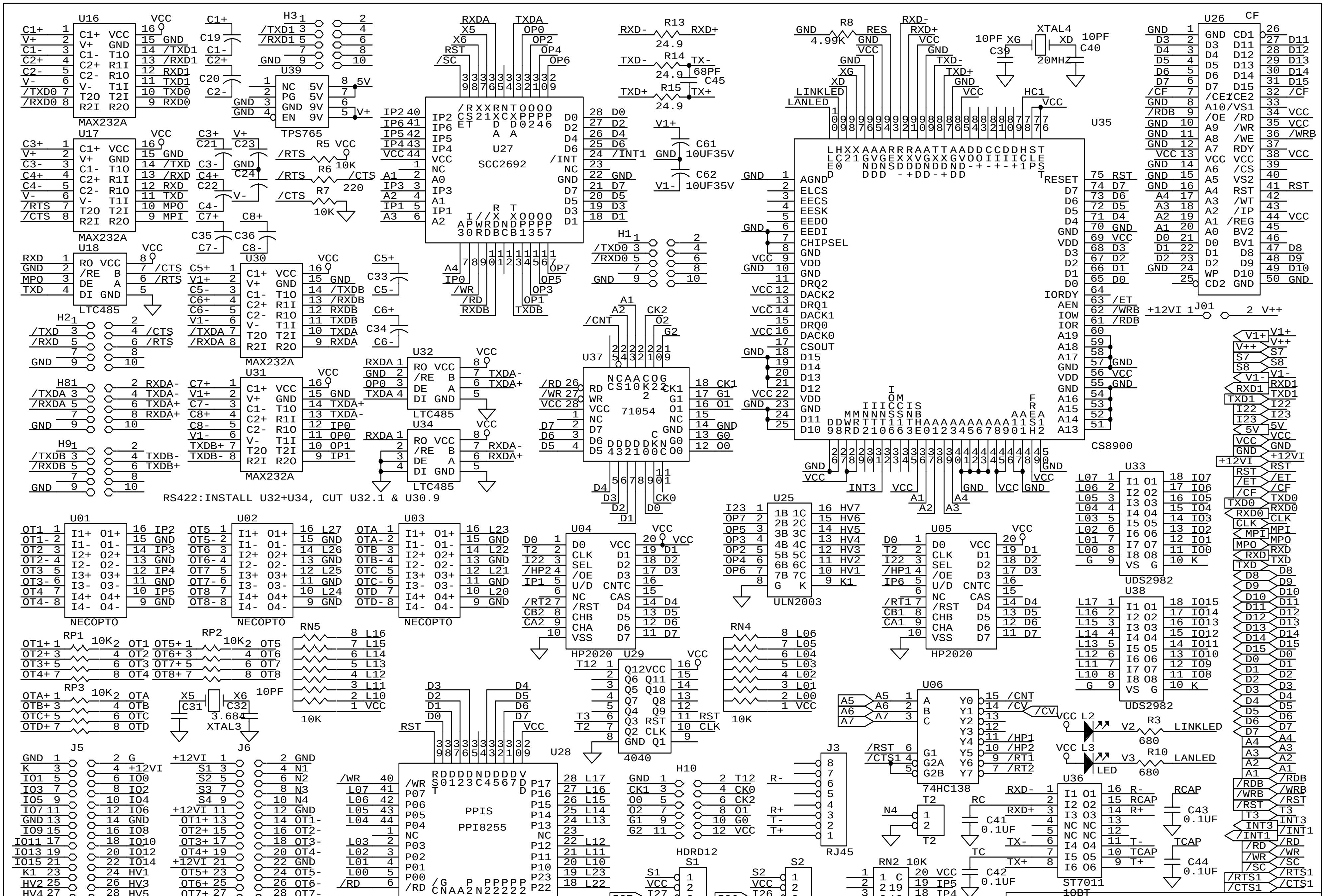
Retrieves a fixed length character string from the input buffer. If the buffer contains less characters than the length requested, *str* will contain only the remaining characters from the buffer. Appends a '\0' character to the end of *str*. Returns the retrieved string length.

```
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.



TERN/STE		
Size B	Document Number SmartTFT	REV
Date: September 21, 2004	Sheet of	



TERN/STE

Title
SmartTFT I/O

Size	Document Number	REV
B	ST.2	

Date: September 28, 2004 Sheet 2 of 2