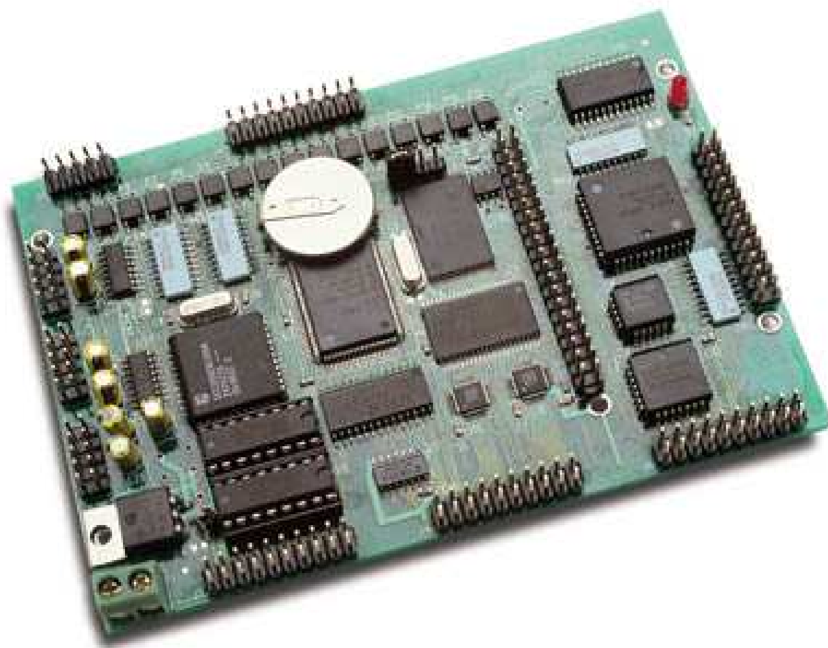
TD86™

High Performance Industrial Controller

With

16 300KHz ADCs, 4 DACs, 4 Serial Ports, 6 16-bit Timers/counters, 16 Opto-couplers, 14 Solenoid Drivers, 16-bit Flash and SRAM

Technical Manual



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

COPYRIGHT

TD86, A-Engine86, A-Engine, A-Core86, A-Core, i386-Engine, V25-Engine, MemCard, MotionC, MotionC2140, VE232, NT-Kit, and ACTF are trademarks of TERN, Inc.
Paradigm C++ is a trademark of Paradigm Systems
Am188ES and Am186ES are trademarks of Advanced Micro Devices, Inc.
Borland C/C++ is a trademark of Borland International.
Microsoft, MS-DOS, Windows, Windows95, and Windows98 are trademarks of Microsoft Corporation.

Version 2.00

October 25, 2010

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

© 1994-2010
1950 5th Street, Davis, CA 95616, USA
Tel: 530-758-0180 Fax: 530-758-0181
Email: ***sales@tern.com*** ***http://www.tern.com***

Important Notice

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

Chapter 1: Introduction

1.1 Functional Description

The ***TD86***™ is designed for rugged industrial control applications that require compact size, superior performance, and reliability.

SUPERIOR PERFORMANCE

The ***TD86*** is a complete C/C++ programmable industrial controller based on a 16-bit, 40 MHz CPU (Am186ES, AMD) with 16-bit external data bus, supporting 16 channels of high speed parallel ADCs (AD7852, 12-bit, 300KHz, 0-5V) and 4 channels parallel DACs (DA7625, 12-bit, 5µs, 0-2.5V).

With the wider external data bus, code executes directly out of 16-bit SRAM or 16-bit flash. The CPU can access the parallel 12-bit ADCs, or parallel DAC with zero wait state in single instruction.

Three 16-bit programmable high-performance counters (71054, NEC), each with its own clock input, gate input, and output, can be clocked up to 10 MHz. Three 16-bit CPU internal timer/counters can support PWM, or pulse-width demodulation.

INDUSTRIAL CONTROL INPUT/OUTPUTS

The ***TD86*** also has 16 opto-isolated digital inputs, 14 solenoid drivers, and 24 TTL I/O pins. The 16 opto-couplers (PS2701/5, NEC) can take up to 50V DC (PS2701, default) or AC (PS2705) inputs. These opto-couplers has 3 µs ON time and 5 µs OFF time. Five opto-inputs are also external interrupts. The 14 solenoid drivers are capable of sinking 350 mA at 50V per line, and they can drive solenoids or relays.

VERSATILE APPLICATIONS

Four serial ports (two CPU internal, two from SCC2692) are configured to RS232 by default. One CPU internal UART(SER1) can be configured to RS485 (half-duplex) or RS422 (full-duplex). The SCC2692 UART can be configure to RS485.

The ***TD86*** also supports a real-time clock (RTC72423) with battery, a watchdog timer, and a 512-byte EEPROM. A 64KW or 256 KW 16-bit SRAM can be installed. Using the DV-P kit, user applications can be easily field-programmed into the 16-bit ACTF flash over the serial link.

A MMA card (from TERN) can be installed to support up to 500 MB PCMCIA ATA flash card, 10 BaseT ethernet, and an additional 39 channels 12-bit/24-bit ADC inputs.

A 82C55 PPI chip provides 24 I/Os, which also can be used to interface to an LCD. An optional switching regulator can be installed to reduce power consumption and heat.

The ***TD86*** also supports power-off mode, allowing µA-level power consumption. In this mode, the real-time clock or an external signal can turn ON or off the TD86 via the VOFF pin of the switching regulator.

A optional TD-Pack including a 16x2 LCD, 8x2 keys, and an enclosure is available.

Measuring 4.8 x 3.4 x 0.3 inches, the ***TD86*** offers high performance, reliability, and versatile features for wide range of applications including high speed data acquisition, motion control, process automation, and instrumentation.

The ***TD86*** is a new design based on the popular original ***TinyDrive***-V25 and the ***TD40***. The original ***TD40*** is based on the Am188ES processor with internal 16-bit data path and external 8-bit data bus. The new ***TD86*** is based on the 40 MHz Am186ES processor, which has both internal and external 16-bit data path. The ***TD86*** uses 16 channels parallel ADCs and DAC, replacing the serial ADC and DACs used on TD40. There are additional three 16-bit hardware Programmable Interval Counters support high speed external events counting without software overhead. Overall, the ***TD86*** is much faster than the 40 MHz TD40. The *TD86* also provides a true 16-bit data bus at J1 20x2 header supporting expansion.

The ***TD86*** is an ideal upgrade for the TinyDrive-V25, or TD40, providing increased reliability, functionality, and performance. They have the similar mechanical dimensions, pin outs, software drivers, and both are programmed using the C/C++ Evaluation Kit (EV-P) or Development Kit (DV-P).

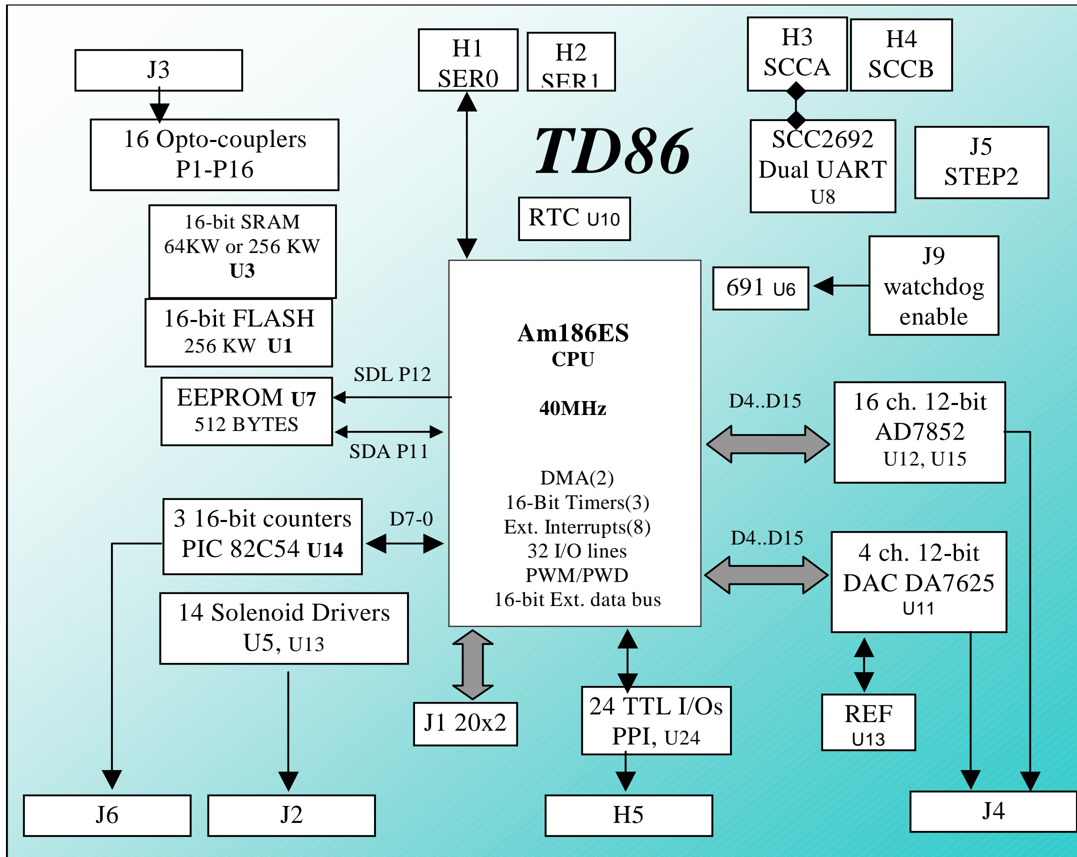


Figure 1.1 Functional block diagram of the TD86

Two DMA-driven serial ports from the Am186ES support high-speed, reliable serial communication at a rate of up to 115,200 baud. An dual UART SCC2692 provides two more serial ports. All four serial ports support 8-bit and 9-bit communication.

There are three AM186ES internal 16-bit programmable timers/counters. Timer0,1 and 2. The Timer1 can be used to count external events, at a rate of up to 10 MHz. Timer0 and timer1 can generate non-repetitive or variable-duty-cycle waveforms as PWM outputs. Timer2 can be used as a pre-scale clock source or used as a timer. 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.

A supervisor chip with power failure detection, a watchdog timer, an LED, and expansion ports are on-board. A MMA board can be installed to provide PCMCIA, 39 more ADCs, and 10-baseT Ethernet port.

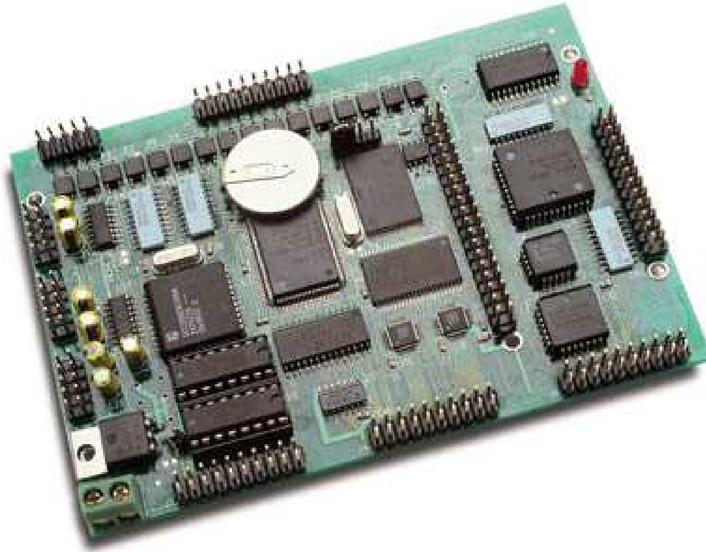


Figure 1.2 TD86

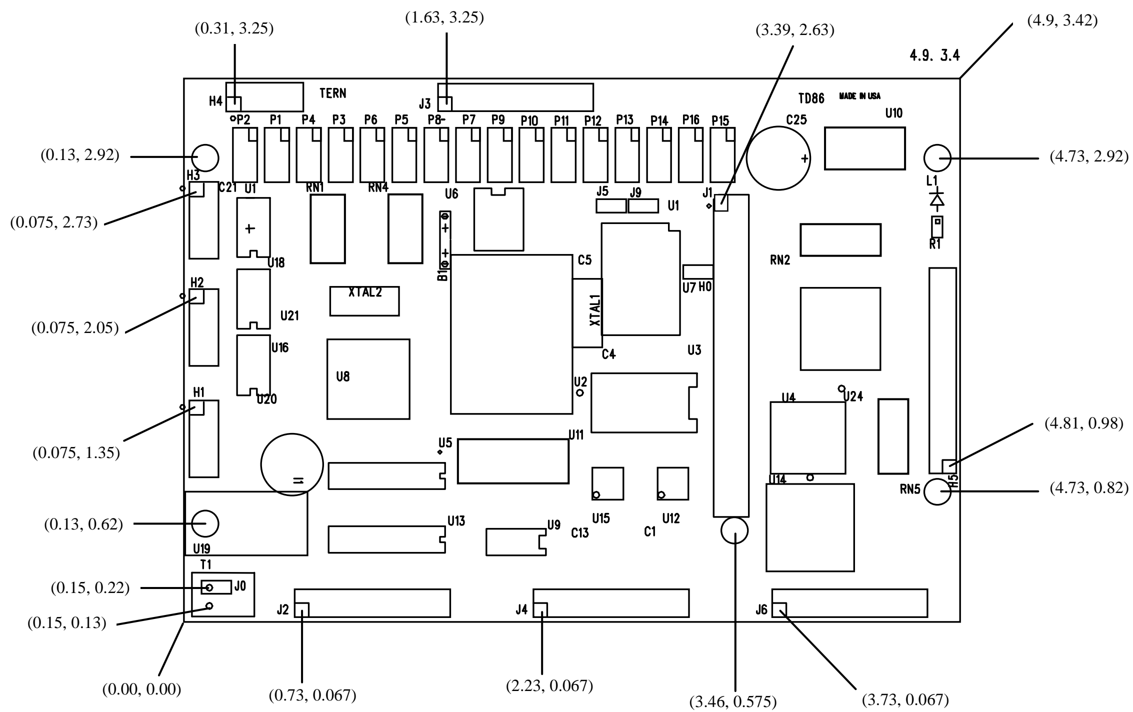
1.2 Features

- 4.8 x 3.4 x 0.5 inches
- 16-bit CPU (Am186ES), 40 MHz, program in C/C++
- 190 mA at 40 MHz, 30 mA in power-save mode
- Up-to 256KW flash, and 64/256 KW SRAM on-board
- 16 ch. 300 KHz 12-bit ADC (AD7852), 0-5V analog input*
- 4 ch. 200 KHz 12-bit DAC (DAC7625), 0-2.5V analog output*
- 16-bit external data bus, 512-byte EE, 4 serial ports
- PWM outputs and Pulse Width Demodulation
- 6 16-bit timer/counters, 24 TTL I/Os, and 14 solenoid drivers
- 16 opto-coupler inputs including 5 external interrupts
- Real-time clock, battery, switching regulator*
- 16-bit external data bus expansion port
- Up to 1GB memory expansion via **MemCard-A™**
- 4 serial ports (2 from Am186ES, 2 from SCC2692 UART)
- 2 high-speed PWM outputs and Pulse Width Demodulation
- 512-byte serial EEPROM
- Supervisor chip (691) for power failure, reset and watchdog
- Real-time clock (RTC72423), lithium coin battery*

  * optional

1.3 Physical Description

The physical layout of the A-Engine86 is shown in Figure 1.3.



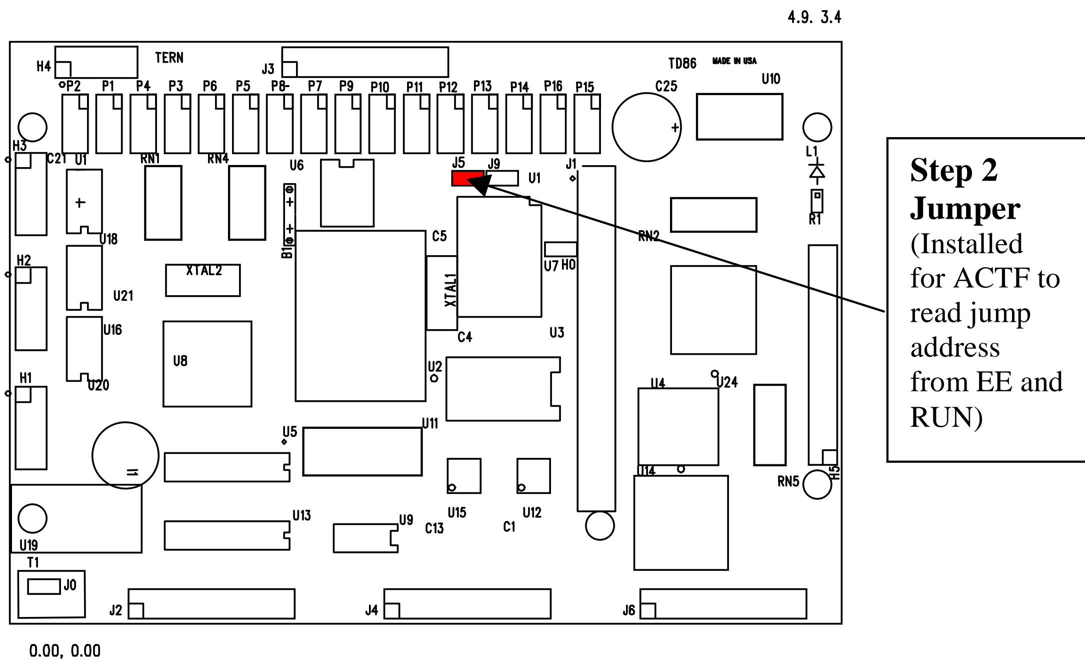


Figure 1.3 Physical layout and STEP2 Jumper Location of the TD86

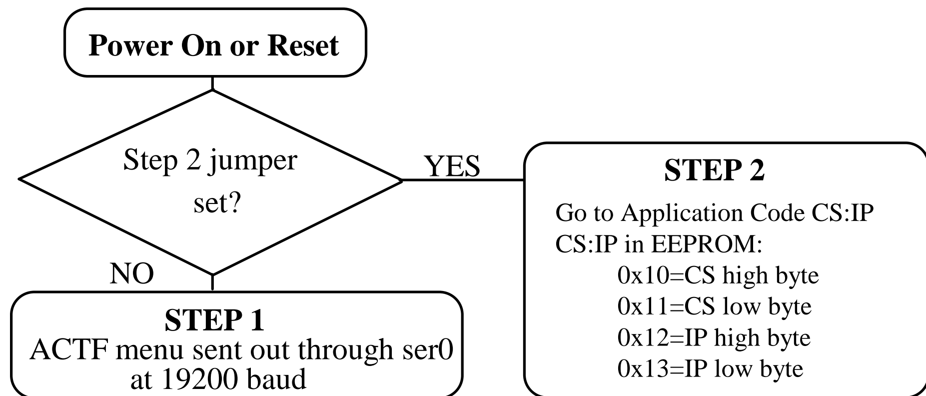


Figure 1.4 Flow chart for ACTF operation

A “ACTF boot loader” resides in the top protected sector of the 256KW on-board Flash chip (29F400). At power-on or RESET, the “ACTF” will check the STEP 2 jumper. If STEP 2 jumper is not installed, the ACTF menu will be sent out from serial port0 at 19200 baud. If STEP 2 jumper is installed, the “jump address” located in the on-board serial EE (see App. E) will be read out and then jump tothat address. A DEBUG kernel “c:\tern\186\rom\ae86\AE86_115.HEX” can be downloaded and programmed into the Flash starting at address 0xFA000 . Using the ACTF menu with “HyperTerminal, 19,200 baud”, use “GFA000” command will setup a “Jump Address of 0xFA000” into the on-board EE (U7), and run the kernel, ready for talking to the PC side Paradigm C++ TERN Edition via RS232 serial link at 115,200 baud.

1.4 Minimum Requirements for TD86 System Development

1.4.1 Minimum Hardware Requirements

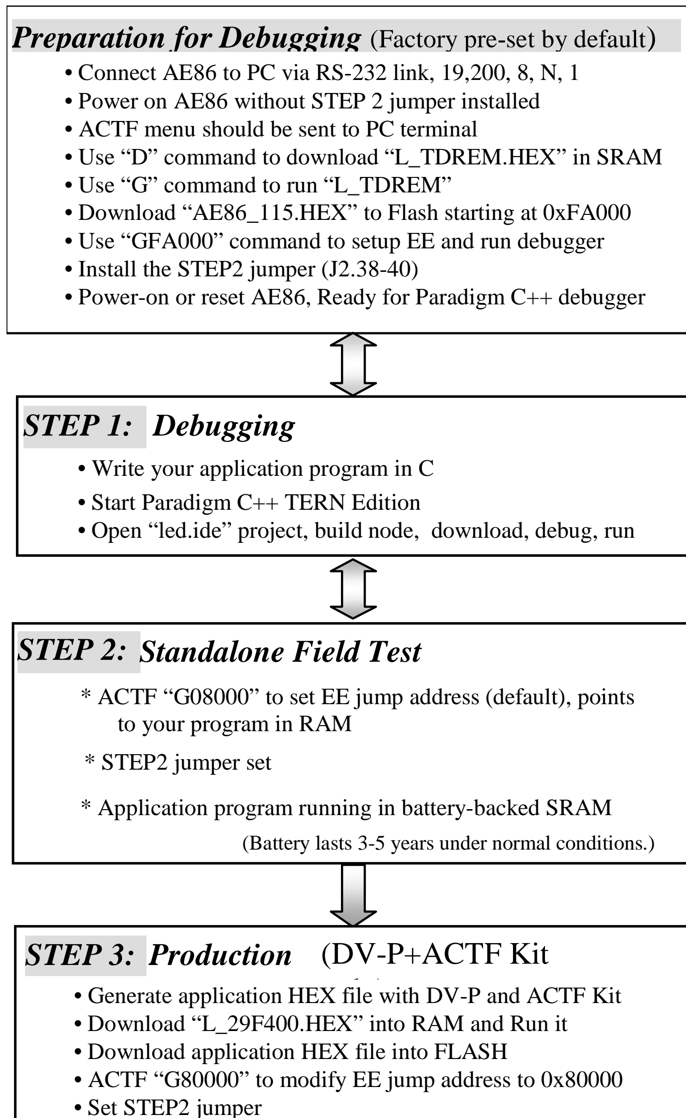
- PC or PC-compatible computer with serial COMx port that supports 115,200 baud
- A TD86 controller
- Debug serial cable (RS-232; DB9 connector for PC COM port and IDE 2x5 connector for controller)
- center negative wall transformer (+9V, 500 mA)

1.4.2 Minimum Software Requirements

- TERN EV-P or DV-P Kit CD and a PC running Windows 95/98/2000

1.5 TD86 Programming Overview

Steps for AE86-based product development:



There is no ROM socket on the TD86. The user's application program must reside in SRAM (Starting at address of 0x08000 by default based on the c:\tern\186\config\186.cfg) for debugging in STEP1, reside in battery-backed SRAM for the standalone field test in STEP2, and finally be programmed into Flash for a complete product. For production, the user must produce an ACTF-downloadable HEX file for the application, based on the DV-P+ACTF Kit. Use "G80000"to point to the user code in Flash". The "STEP2" jumper (J2 pins 38-40) must be installed for every production-version board.

Chapter 2: Installation

2.1 Software Installation

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

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
- Connect wall transformer:
  Connect 9V wall transformer to power and plug into power jack

Hardware installation for the TD86 consists primarily of connecting the microcontroller to your PC

2.2.1 Connecting the TD86 to the PC

The following diagram (Figure 2.1) illustrates the connection between the TD86 and the PC. The TD86 is linked to the PC via a serial cable (DB9-IDE).

The TD86 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).

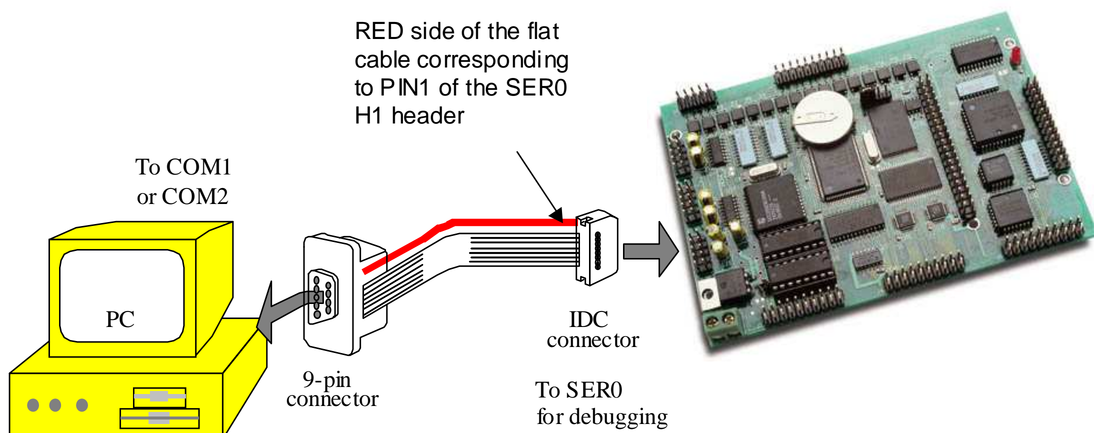


Figure 2.1 Connecting the TD86 to the PC via RS232 serial link

2.2.2 Powering-on the TD86

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

Connect +9V to +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.

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

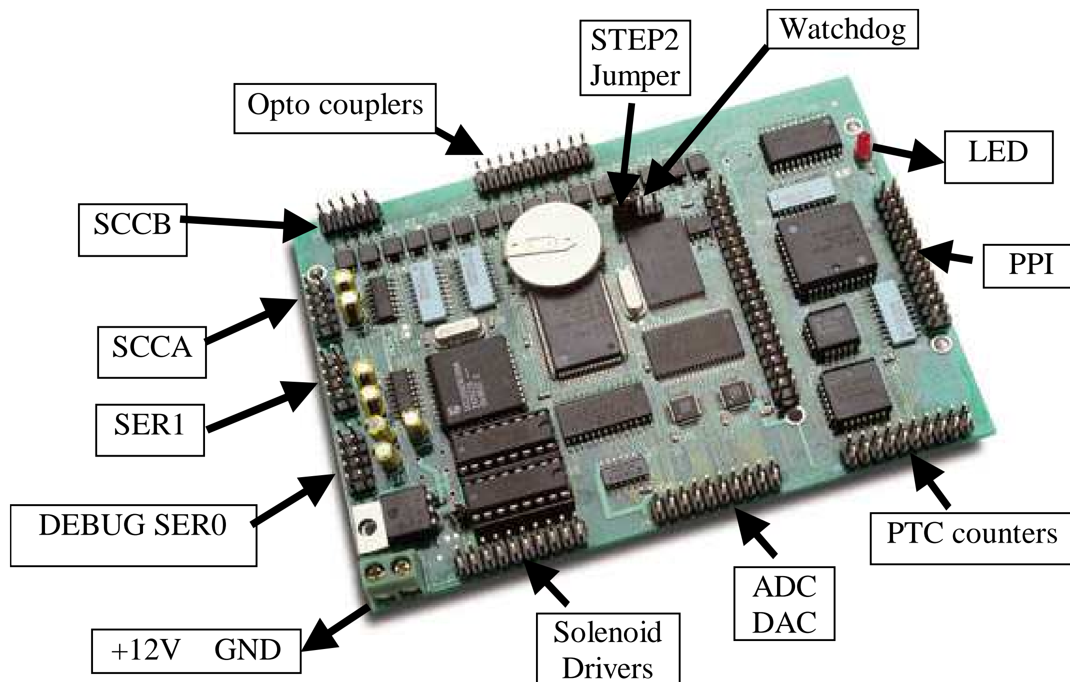


Fig 2.2 Locations of STEP2 Jumper, LED, Power input and DEBUG port

The LED blinks twice indicating ready for DEBUG, while STEP2 Jumper installed, DEBUG kernel resides in flash (0xFA000), and EE Jump address = 0xFA000 (Set by “GFA000”), after powered-on or reset

Chapter 3: Hardware

3.1 Am186ES - Introduction

The Am186ES is based on industry-standard x86 architecture. The Am186ES controllers uses 16-bit external data bus, are higher-performance, more integrated versions of the 80C188 microprocessors which uses 8-bit external data bus. In addition, the Am186ES has new peripherals. The on-chip system interface logic can minimize total system cost. The Am186ES has two asynchronous serial ports, 32 PIOs, a watchdog timer, additional interrupt pins, a pulse width demodulation option, DMA to and from serial ports, a 16-bit reset configuration register, and enhanced chip-select functionality.

3.2 Am186ES – Features

Clock

Due to its integrated clock generation circuitry, the Am186ES microcontroller allows the use of a times-one crystal frequency. The design achieves 40 MHz CPU operation, while using a 40 MHz crystal.

The system CLKOUTA signal is routed to J1 pin 4, default 40 MHz. The CLKOUTB signal is not connected in the TD86.

CLKOUTA remains active during reset and bus hold conditions. The TD86 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=CLK=J1 pin 4.

External Interrupts and Schmitt Trigger Input Buffer

There are eight external interrupts: INT0-INT6 and NMI provided by the Am186ES.

/INT0 is used by the on-board SCC2692 Dual UART.
/INT1, buffered by opto-coupler P15 at J3 pin 17=IN15, edge triggered interrupt only
INT2=P31, buffered by opto-coupler P14 at J3 pin 16=IN14, input, or interrupt, or PWD
INT3, buffered by opto-coupler P10 at J3 pin 12=IN10, edge triggered interrupt only
INT4=P30, buffered by opto-coupler P13 at J3 pin 15=IN13, input or interrupt
INT5=P12, internal used only by TD86 as output for LED/EE/HWD
INT6=P13, buffered by opto-coupler P16 at J3 pin 18=IN16, input or interrupt
/NMI, Supervisor 691 U6 pin 10 power-fail output (PFO).

External interrupt inputs (/INT0, /INT1, /NMI) and three counter clock inputs (CNT0-2, U14, 82C54), are buffered by Schmitt-trigger inverters (U9, 74HC14), in order to increase noise immunity and transform slowly changing input signals to fast changing and jitter-free signals. As a result of this buffering, these pins are capable of only acting as input.

The buffered external interrupt input IN15=/INT1 requires a falling edge (HIGH-to-LOW) to generate an interrupt. The IN10=INT3 requires a rising edge (Low-to-high) to generate an interrupt. The IN14=INT2=P31, IN13=INT4=P30, and IN16=INT6=P13 are used as inputs by default. User can re-program these pins to rising edge trigger interrupts.

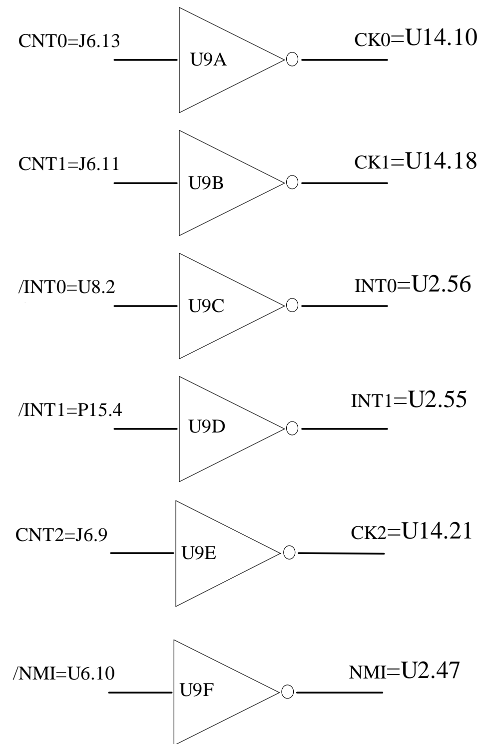


Figure 3.1 External interrupt inputs

The TD86 uses vector interrupt functions to respond to external interrupts. Refer to the Am186ES User's manual for information about interrupt vectors.

Asynchronous Serial Ports

The Am186ES CPU has two asynchronous serial channels: SER0 and SER1. Both asynchronous serial ports support the following:

- Full-duplex operation; 7-bit, 8-bit, and 9-bit data; odd, even, and no parity; one stop bit
- Error detection and hardware flow control
- DMA transfers to and from serial ports; Transmit and receive interrupts for each port
- Multidrop 9-bit protocol support; 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 samples files ***s1_echo.c*** and ***s0_echo.c***.

An external SCC2692 Dual UART is located in position U8. For more information about the external UART SCC2692, please refer to SCC2692.pdf file included in the CD.

Timer Control Unit

Am186ES includes 3 internal 16-bit timer/counters. Additional 3 external 16-bit timer/counters (82C54, U14) are supported on the TD86.

The Am186Es internal timer/counter unit has three 16-bit programmable timers: Timer0, Timer1, and Timer2. Timer0 and Timer1 are connected to external pins:

Timer0 output = P10 = J6 pin 8
Timer0 input = P11 = U7 EE pin 5 = J6.2

Timer1 output = P1 = J6 pin 6
Timer1 input = P0 = J6 pin 4

Timer0 input P11 is used and shared by on-board EE. At power-on or reset, if the STEP2 Jumper is installed, the TD86 will read the EE. If an external source is using the P11 as Timer0 external counter input, then the EE operation will run in problem. So the P11is not recommended for other external use, unless user can disable the external source at power-on, or reset.

The timer can be used to count or time external events, or can generate non-repetitive or variable-duty-cycle waveforms.

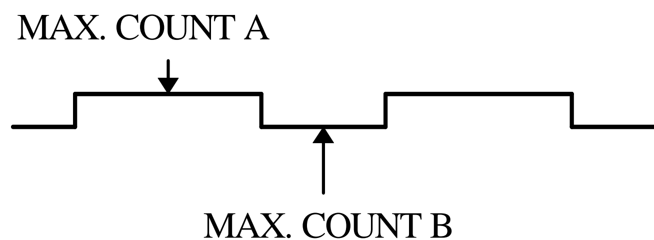
Timer2 is not connected to any external pin. It can be used as an internal timer for real-time coding or time-delay applications. It can also prescale timer 0 and timer 1 or be used as a DMA request source.

The maximum rate at which each timer can operate is 10 MHz, since each timer is serviced once every fourth clock cycle. Timer output takes up to six clock cycles to respond to clock or gate events. See the sample programs ***timer02.c*** and ***ae_cnt1.c*** in the **`tern\186\samples\ae`** directory.

PWM outputs and PWD

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

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



Pulse Width Demodulation can be used to measure the input signal's high and low phases on the INT2 pin (INT2=P31, buffered by opto-coupler P14 at J3 pin 16=IN14, input, or interrupt, or PWD).

Power-save and Power-off Mode

The TD86 is an ideal high performance controller for low power consumption applications. The power-save mode of the Am186ES 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 TD86 has a VOFF signal routed to RTC72423 (U10 pin 1), J1 pin 9, and 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 optional on-board switching power supply, and can turn the whole board power supply off. In the power-off mode, only micro amp current power consumption. More details are available in the sample file ***poweroff.c*** in the **`186\samples\ae`** sub-directory.

3.3 Am186ES PIO lines

The Am186ES has 32 pins available as user-programmable I/O lines. Each of these pins can be used as a user-programmable input or output signal, if the normal shared function is not needed. A PIO line can be configured to operate as an input or output with or without a weak pull-up or pull-down, or as an open-drain output. A pin's behavior, either pull-up or pull-down, is pre-determined and shown in the table below.

After power-on/reset, PIO pins default to various configurations. The initialization routine provided by TERN libraries reconfigures some of these pins as needed for specific on-board usage, as well. These configurations, as well as the processor-internal peripheral usage configurations, are listed below in Table 3.1.

PIO	*Function*	*Power-On/Reset status*	*TD86 Pin No.*	*TD86 Initial*
P0	Timer1 in	Input with pull-up	J6 pin 4	Input with pull-up
P1	Timer1 out	Input with pull-down	J6 pin 6	Input with pull-down
P2	/PCS6/A2	Input with pull-up	U10 pin 2	RTC72423 select
P3	/PCS5/A1	Input with pull-up	U8 pin 39	SCC2692 select
P4	DT/R	Normal	J5 pin 2	Input with pull-up Step 2
P5	/DEN/DS	Normal	U13 pin 1	Input with pull-up (HV14)
P6	SRDY	Normal	J1 pin 25	Input with pull-down
P7	A17	Normal	U3 pin 22	A17
P8	A18	Normal	U3 pin 23	A18
P9	A19	Normal	U13 pin 6	A19 (HV9)
P10	Timer0 out	Input with pull-down	J6 pin 8	Input with pull-down
P11	Timer0 in	Input with pull-up	U7 EE pin 5, J6.2	Input with pull-up
P12	DRQ0/INT5	Input with pull-up	J9 pin 2	Output for LED/EE/HWD
P13	DRQ1/INT6	Input with pull-up	Opto P16 pin 4	Input with pull-up
P14	/MCS0	Input with pull-up	J13 pin 7	Input with pull-up (HV8)
P15	/MCS1	Input with pull-up	Opto P8 pin 4	Input with pull-up
P16	/PCS0	Input with pull-up	J1 pin 19	/PCS0
P17	/PCS1	Input with pull-up	PAL U4 pin 9	PPI, ADC, DAC select
P18	CTS1/PCS2	Input with pull-up	Opto P9 pin 4	Input with pull-up
P19	RTS1/PCS3	Input with pull-up	RS485 U20 pin 3	Input with pull-up
P20	RTS0	Input with pull-up	Opto P12 pin 4	Input with pull-up
P21	CTS0	Input with pull-up	Opto P11 pin 4	Input with pull-up
P22	TxD0	Input with pull-up	RS232 U16 pin 10	TxD0
P23	RxD0	Input with pull-up	RS232 U16 pin 9	RxD0
P24	/MCS2	Input with pull-up	U13 pin 2	Input with pull-up (HV13)
P25	/MCS3	Input with pull-up	U13 pin 3	Input with pull-up (HV12)
P26	UZI	Input with pull-up	U13 pin 4	Input with pull-up (HV11)
P27	TxD1	Input with pull-up	RS232 U16 pin 11	TxD1
P28	RxD1	Input with pull-up	RS232 U16 pin 12	RxD1
P29	/CLKDIV2	Input with pull-up	U13 pin 5	Input with pull-up (HV10)
P30	INT4	Input with pull-up	Opto P13 pin 4	Input with pull-up
P31	INT2	Input with pull-up	Opto P14 pin 4	Input with pull-up

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

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

Three external interrupt lines are not shared with PIO pins:

INT0 = Used by SCC2692 interrupt
/INT1 = P15 pin 4
INT3 = P10 pin 4

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

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 initialize 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,

Some of the I/O lines are used by the TD86 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 and INT1 are not available for use as output because of the 74HC14 inverters attached. The input values of these PIO interrupt lines will also be inverted for the same reason. As a result, calling ***pio_rd*** to read the value of P31 (**INT2**) will return 1 when Opto-coupler P14 pin 1 on header J3 pin 16=IN14 is pulled high, with the result reversed if the pin is short to ground.

Signal	**Pin**	**Function**
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	STEP2 jumper
P11	Timer0 input	Shared with U7 24C04 EE data input, tri-state, while disabled
P12	DRQ0/INT5	Output for LED or U7 serial EE clock or Hit watchdog

Signal	Pin	Function
P17	/PCS1	U4 PAL select for PPI, DAC, ADC, CNT(82C54) select signals
P22	TxD0	Default SER0 debug
P23	RxD0	Default SER0 debug
/INT0	J2 pin 8	U8 SCC2692 UART interrupt, if U8 is installed

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

3.4 I/O Mapped Devices

I/O Space

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

The default I/O access time is 15 wait states. You may use the function void ***io_wait***(char wait) to define the I/O wait states from 0 to 15. The system clock is 25 ns (or 50 ns), giving a clock speed of 40 MHz (or 20 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.

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

I/O space	Select	Location	Usage
0x0000-0x00ff	/PCS0	J1 pin 19=P16	USER expansion, MM
0x0100-0x0103	/PCS1	J2 pin 13=P17	U4 PAL
0x0130	/CNT	U14 pin 24	82C54, /CNT(PIT)
0x0110	/DA	U11 pin 23	DAC7625 select
0x0180	/AD	U12 pin 31	AD7852
0x0120	/AD1	U15 pin 31	AD7852
0x0100	/PPI	U24 pin 7	82C55, PPI
0x0400-0x04ff	/PCS4		Reserved
0x0500-0x05ff	/PCS5	U8 pin 15=39	UART, SCC2692
0x0600-0x06ff	/PCS6	U10 pin 2=P2	RTC 72423

*PCS0 may be used for other TERN peripheral boards such as MMA.

To illustrate how to interface the TD86 with external I/O boards via J1 header, a simple decoding circuit for interfacing to an 82C55 parallel I/O chip is shown in Figure 3.2.

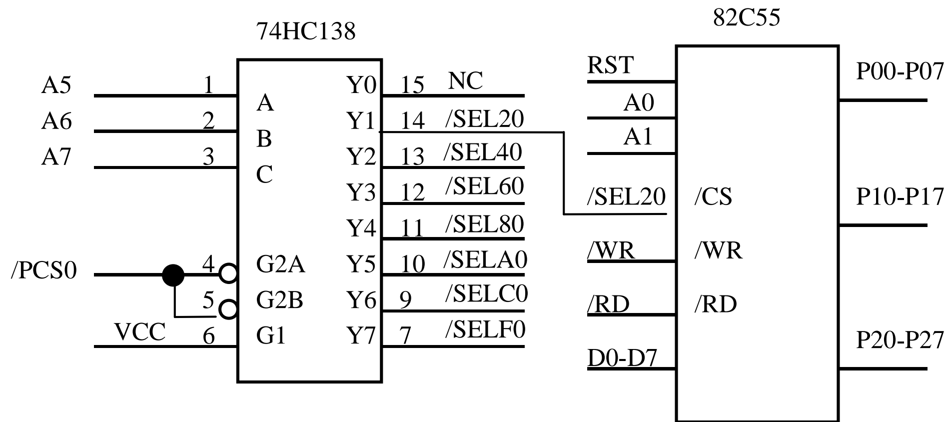


Figure 3.2 Interface to external I/O devices via J1

The function **`ae_init()`** by default initializes the /PCS0 line at base I/O address starting at 0x00. You can read from the 82C55 with ***inportb(0x020)*** or write to the 82C55 with ***outportb(0x020,dat)***. The call to ***inportb(0x020)*** will activate /PCS0, as well as putting the address 0x00 over the address bus. The decoder will select the 82C55 based on address lines A5-7, and the data bus will be used to read the appropriate data from the off-board component.

Programmable Peripheral Interface (82C55A)

U5 PPI (82C55) is a low-power CMOS programmable parallel interface unit for use in microcomputer systems. It provides 24 I/O pins that may be individually programmed in two groups of 12 and used in three major modes of operation.

In MODE 0, the two groups of 12 pins can be programmed in sets of 4 and 8 pins to be inputs or outputs. In MODE 1, each of the two groups of 12 pins can be programmed to have 8 lines of input or output. Of the 4 remaining pins, 3 are used for handshaking and interrupt control signals. MODE 2 is a strobed bi-directional bus configuration.

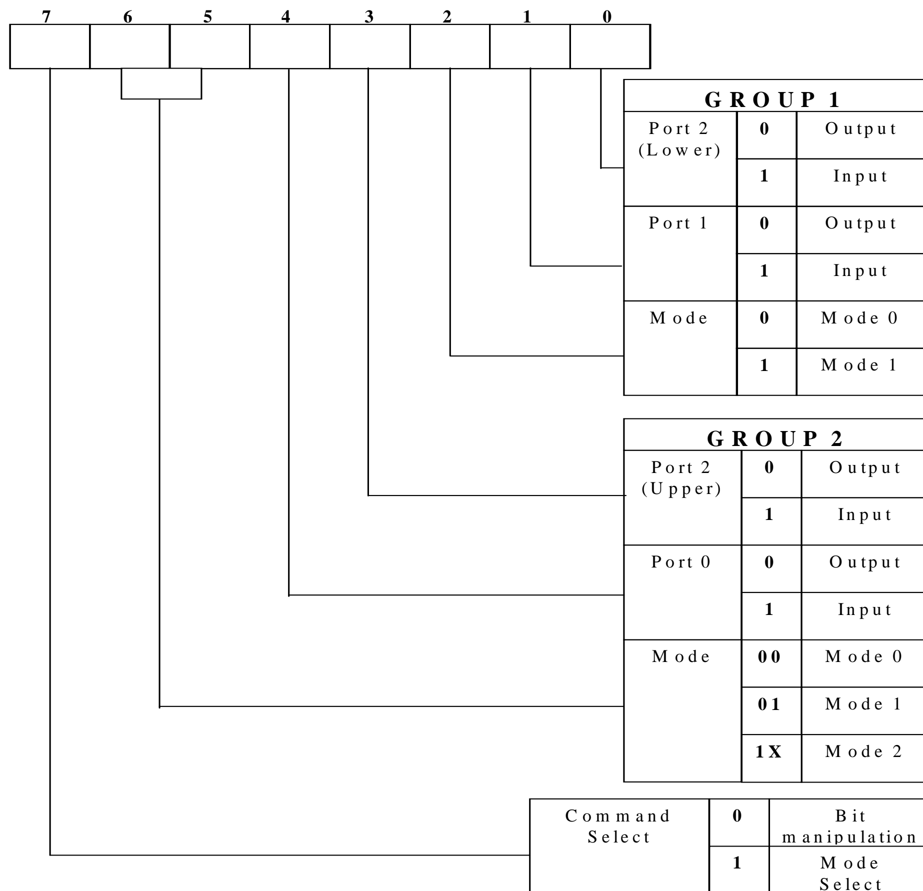


Figure 3.3 Mode Select Command Word

The TD86 maps U5, the 82C55/uPD71055, at base I/O address 0x0100.

The ports/registers are offsets of this I/O base address.

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 can 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.

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_rd()***.

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\v25\samples\ve\poweroff.c`**.

UART SCC2692

The UART SCC2692 (Signetics, U8) is mapped into the I/O address space at 0x0500. The SCC2692 has dual 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 SCC2692.PDF file included in TERN CD.

3.5 Other Devices

A number of other devices are also available on the TD86. 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.

On-board Supervisor with Watchdog Timer

The MAX691/LTC691 (U6) is a supervisor chip. With it installed, the TD86 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 TD86. 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 TD86 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**().

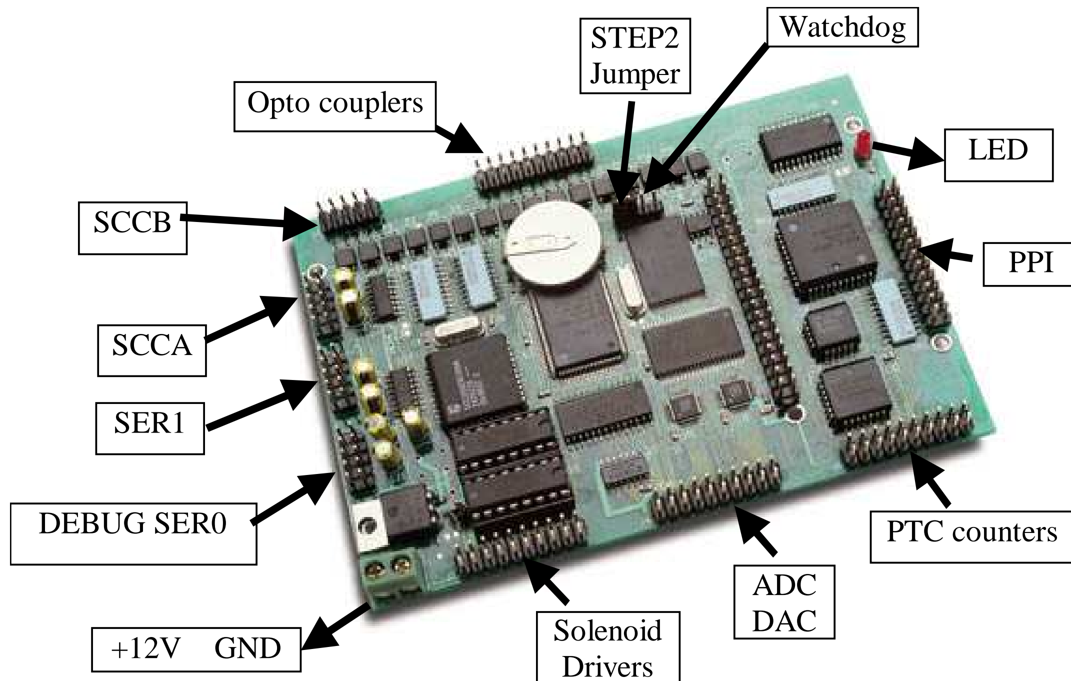


Figure 3.4 Location of watchdog timer enable jumper

Power-failure Warning

The supervisor supports power-failure warning and backup battery protection. When power failure is sensed by the PFI, pin 9 of the MAX691 (lower than 1.3 V), the PFO is low. The PFI pin 9 of 691 is connected to the 10K ohm RN1 pin 10 and is pulled high to VCC. In order to use PFI externally, add a wire to bring the PFI signal out via any empty header pin, such as H4 pin 10. You may design an NMI service routine to take protect actions before the +5V drops and processor dies. The following circuit shows how you might use the power-failure detection logic within your application.

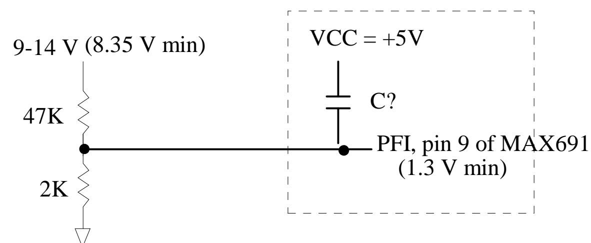


Figure 3.5 Using the supervisor chip for power failure detection

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.

EEPROM

A serial EEPROM of 512 bytes (24C04), or an optional 2K bytes (24C16) can be installed in U7. The TD86 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()**.

The EEPROM uses P11 as data input signal line. P11 at J6 pin 2 is a multifunctional PIO pin, which can be Timer0 input. Be aware of the ACTF flash firmware uses EE with P11 signal at power-on, or reset, if STEP2 jumper is installed. Not recommended to use P11 for Timer0 external counter input, unless the external clock source can be disabled during power-on.

AD7852, 300KHz 12-bit ADC

Two AD7852 chips can be installed on the TD86. The AD7852 is a 100 ksps, sampling parallel 12-bit A/D converter that draws only 55 mW from a single 5V supply. This device includes 8 channels with sample-and-hold, precision 2.5V internal reference, switched capacitor successive-approximation A/D, and needs an external clock.

The input range of the AD7852 is 0-5V. Maximum DC specs include ±2.0 LSB INL and 12-bit no missing codes over temperature. The ADC has a 12-bit data parallel output port that directly interfaces to the full 12-bit data bus D15-D4 for maximum data transfer rate.

The AD7852 requires 16 ADC clocks (or 3 µs) conversion time to complete one conversion, based on a 5 MHz ADC clock. The busy signal has an 3 µs low period indicating that conversion is in progress. In order to achieve the 300 KHz sample rate, the AE86 must use polling method, not interrupt operation, to acquire data. A sample program **td86_ad.c** can be found in the **`c:\tern\186\samples\td86`** directory.

DA7625, 300KHz 12-bit DAC

The DA7625 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, has double-buffered DAC input logic, and has a settling time of 10 µs.

The TD86 uses pins D15 to D4 to directly interface to the DAC's full 12-bit data bus for maximum data transfer rate.

The DA7625 has a settling time of 5 µs. A sample program **td86_da.c** may be found in the **`c:\tern\186\samples\td86`** directory.

Programmable Timer/Counter (Intel 82C54, or NEC PD71054)

The 82C54/PD71054 Programmable Timer/Counter (PTC) chip provides three high-performance, programmable counters for the TD86. 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 J6. See the CD included TD86 schematics and the 82C54.PDF file for more details. Sample programs are available under c:\tern

186\samples\td86.

Opto-couplers

There are 16 opto-couplers at the J3 header. These opto-couplers provide optical isolation and can used for digital inputs, relay contact monitor, or powerline monitor. These optos has a 3 micro-second ON time and 5 micro-second OFF time. The input pins on-board has a 1 K ohm pullup, so ground the pin will turn ON the coupler.

3.6 Headers and Connectors

J1 20x2 pin Header, Expansion port with address bus, data bus, and control lines.
J2 10x2 pin Header, 14 Solenoid Drivers.
J3 10x2 pin Header, 16 Opto-coupler inputs.
J4 10x2 pin Header, 16 high speed, 300K Hz, 12-bit ADC inputs and 4 12-bit DACs.
J6 10x2 pin Header, 6 16-bit Timer/Counter I/Os.
H1 5x2 pin Header, SER0, default DEBUG port, or RS232 for application.
H2 5x2 pin Header, SER1, RS232/RS422/RS485 for application.
H3 5x2 pin Header, SCC2692 portA, RS232/RS485 for application.
H4 5x2 pin Header, SCC2692 portB, RS232 for application.
H5 13x2 pin Header, 24 PPI I/Os.
J5 2x1 pin Header, STEP2 Jumper.
J9 2x1 pin Header, Watchdog enable jumper.
A detailed signal definition is available in the attached td86 schematics (td86-man.sch).

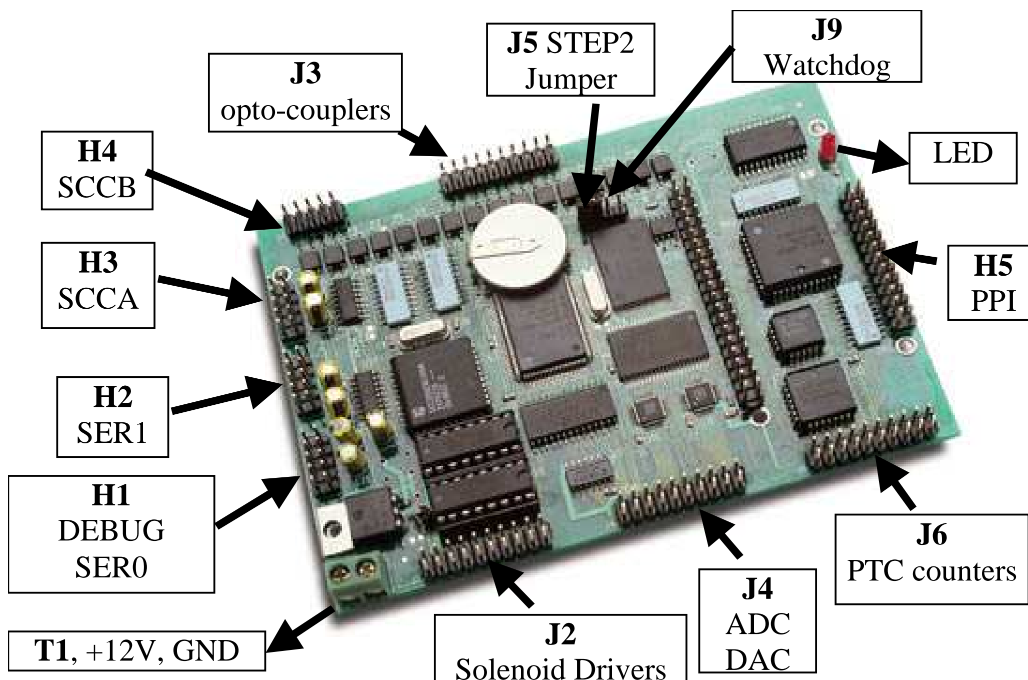


Figure 3.6 Header locations of the TD86

J1 Signal			
VCC	1	2	GND
MPO	3	4	CLK
RxD	5	6	GND
TxD	7	8	D0
VOFF	9	10	D1
/BHE	11	12	D2
D15	13	14	D3
/RST	15	16	D4
RST	17	18	D5
P16	19	20	D6
D14	21	22	D7
D13	23	24	GND
	25	26	A7
D12	27	28	A6
/WR	29	30	A5
/RD	31	32	A4
D11	33	34	A3
D10	35	36	A2
D9	37	38	A1
D8	39	40	A0

J1 20x2 pin Header, Expansion port with address bus, data bus, and control lines.

+12V	1	2	K
GND	3	4	GND
GND	5	6	GND
HV8	7	8	HV1
HV9	9	10	HV2
HV10	11	12	HV3
HV11	13	14	HV4
HV12	15	16	HV5
HV13	17	18	HV6
HV14	19	20	HV7

J2 10x2 pin Header, 14 Solenoid Drivers.

GND	1	2	GND
IN1	3	4	IN2
IN3	5	6	IN4
IN5	7	8	IN6
IN7	9	10	IN8
IN9	11	12	IN10
IN11	13	14	IN12
IN13	15	16	IN14
IN15	17	18	IN16
P15	19	20	P16

J3 10x2 pin Header, 16 Opto-coupler inputs.

DA2	1	2	DA1
DA4	3	4	DA3
AD9	5	6	AD8
AD11	7	8	AD10
AD13	9	10	AD12
AD15	11	12	AD14
AD1	13	14	AD0
AD3	15	16	AD2
AD5	17	18	AD4
AD7	19	20	AD6

J4 10x2 pin Header, 16 high speed, 300K Hz, 12-bit ADC inputs and 4 12-bit DACs.

OUT0	1	2	P11
Gate2	3	4	P0 (TMRIN1)
Gate1	5	6	P1 (TMRO1)
Gate0	7	8	P10 (TMRO0)
CNT2	9	10	T1, 10MHz
CNT1	11	12	T2, 5MHz
CNT0	13	14	T2, 5MHz
OUT1	15	16	OUT2
GND	17	18	VCC
	19	20	

J6 10x2 pin Header, 6 16-bit Timer/Counter I/Os.

I10	26	25	I11
I12	24	23	P13
I14	22	21	P15
I16	20	19	P17
I20	18	17	I21
I22	16	15	I23
I24	14	13	I25
I26	12	11	I27
I00	10	9	I01
I02	8	7	I03
I04	6	5	I05
GND	4	3	VCC
I06	2	1	I07

H5 13x2 pin Header, 24 PPI I/Os.

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 F.

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

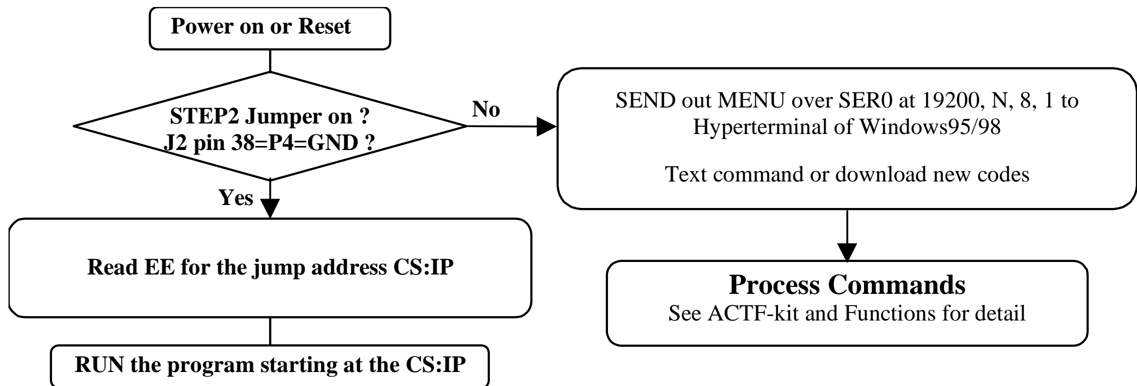
This function can be used to retrieve data from components in I/O space. You will find that most hardware options added to TERN controllers are mapped into I/O space, since memory space is valuable and is reserved for uses related to the code and data. Using I/O mappings, the address is output over the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

The ACTF loader in the TD86 512KB Flash will perform the system initialization and prepare for new application code download or immediately run the pre-loaded code. A remote debugger kernel can be loaded into the Flash located starting 0xfa000. Debugging at baud rate of 115,200 (AE86_115.HEX) and 38,400 (AE86_384.HEX) are available. A loader file L_TDREM.HEX and both debugger files AE86_115.HEX and AE86_384.HEX, are included in the EV-P or DV-P CD under the **`c:\tern\186\rom\AE86`** directory.

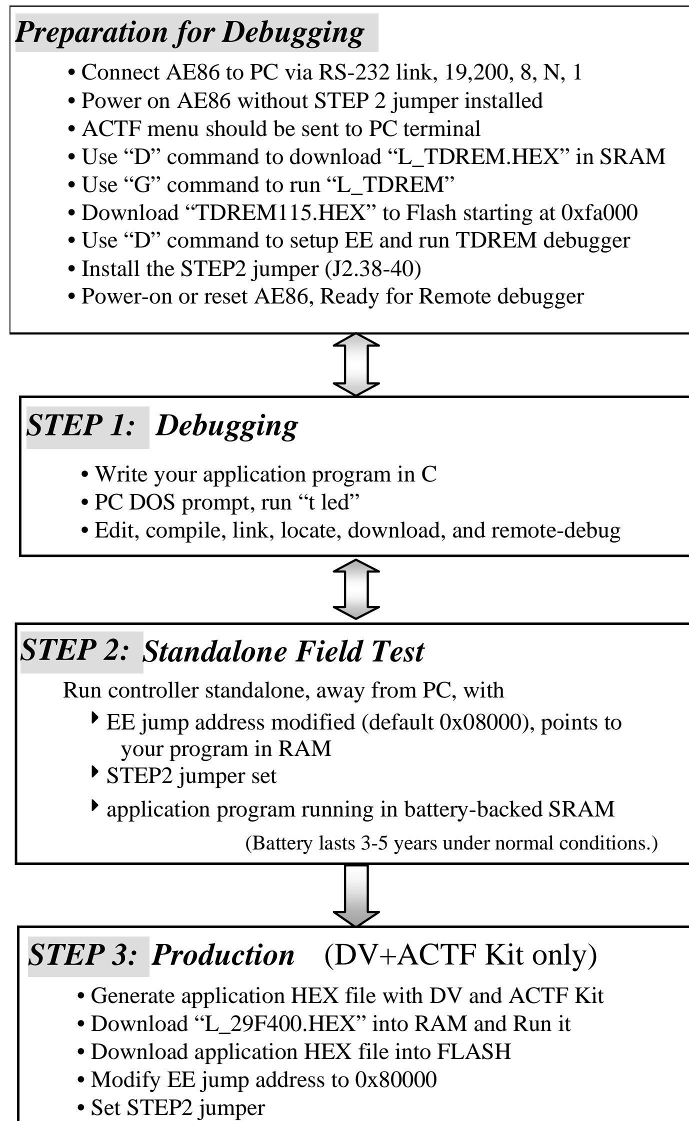
A functional diagram of the ACTF (embedded in the AE86) is shown below:



The C function prototypes supporting Am188/186ES hardware can be found in header file "**ae.h**", in the **`c:\tern\186\include`** directory.

Sample programs can be found in the **`c:\tern\186\samples\ae`** and **`c:\tern\186\samples\td86`** directories.

4.1.1 Steps for AE86-based product development



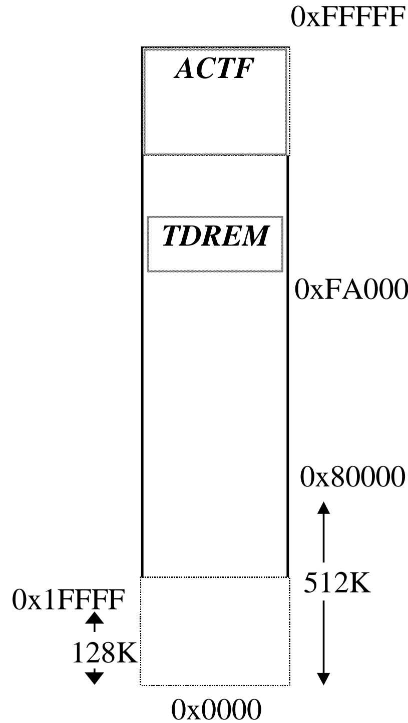
There is no ROM socket on the AE86. The user's application program must reside in SRAM for debugging in STEP1, reside in battery-backed SRAM for the standalone field test in STEP2, and finally be programmed into Flash for a complete product.

The on-board Flash 29F400BT has 256K words of 16-bits each. It is divided into 11 sectors, comprised of one 16KB, two 8KB, one 32KB, and seven 64KB sectors. The top one 16KB sector is pre-loaded with ACTF boot strip, the one 8KB sector starting 0xfa000 is for loading remote debugger kernel, and the reset all sectors are free for application use.

The top 16KB ACTF boot strip is protected.

Two utility HEX files, "L_TDREM.HEX" and "L_29F400.HEX", are designed for downloading into SRAM starting at 0x04000 with ACTF-PC-HyperTerminal. Use the "D" command to download, and use the "G" command to run.

"L_TDREM.HEX" will erase the 8KB sector and load a "TDREM115.HEX" or "TDREM384.HEX". "L_29F400.HEX" will erase the remaining sectors for downloading your application HEX file.



For production, the user must produce an ACTF-downloadable HEX file for the application, based on the DV+ACTF Kit. The application HEX file can be loaded into the on-board Flash starting address at 0x80000.

The on-board EE must be modified with a "G80000" command while in the ACTF-PC-HyperTerminal Environment.

The "STEP2" jumper (J2 pins 38-40) must be installed for every production-version board.

Step 1 settings

In order to correctly download a program in STEP1 with PC Turbo Remote Debugger, the AE86 must meet these requirements:

1) TDREM115.HEX must be pre-loaded into Flash starting address 0xfa000.

2) The SRAM installed must be large enough to hold your program.

For a 32K SRAM, the physical address is 0x00000-0x07fff
For a 128K SRAM, the physical address is 0x00000-0x01ffff
For a 512K SRAM, the physical address is 0x00000-0x07ffff

3) The on-board EE must have a correct jump address for the TDREM115.HEX with starting address of 0xfa000.

4) The STEP2 jumper must be installed on J2 pins 38-40.

4.2 AE.LIB

AE.LIB is a C library for basic A-Engine86 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

4.3 Functions in AE.OBJ

4.3.1 A-Engine86 Initialization

ae_init

This function should be called at the beginning of every program running on A-Engine86 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. You can reset these to inputs.

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

The chip select lines are set to 15 wait states, by default. 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
```

4.3.2 External Interrupt Initialization

There are up to eight external interrupt sources on the A-Engine86, 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 8 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)());
```

4.3.3 I/O Initialization

Two ports of 16 I/O pins each are available on the A-Engine86. Hardware details regarding these PIO lines can be found in the Hardware chapter.

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

You should also confirm the PIO usage that is described above within **ae_init()**. During initialization, several lines are reserved for TERN usage and you should understand that these are not available for your application. There are several PIO lines that are used for other on-board purposes. These are all described in some detail in the Hardware chapter of this technical manual. For a detailed discussion toward the I/O ports, please refer to Chapter 11 of the AMD 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 us. 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 us 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, normal operation
- 1, input with pullup/down
- 2, output
- 3, input without pull

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.

4.3.4 Timer Units

The three timers present on the A-Engine86 can be used for a variety of applications. All three timers run at ¼ of the processor clock rate, which determines the maximum resolution that can be obtained. Be aware that if you enter power save mode, the timers will operate at a reduced speed as well.

These timers are controlled and configured through a mode register that is specified using the software interfaces. The mode register is described in detail in chapter 8 of the AMD AM188ES User's Manual.

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

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

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

U12 AD7852 uses Timer1 output (P1=J2.29) as ADC clock, up to 5MHz.

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 AM188ES 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.

4.3.5 Analog-to-Digital Conversion

Parallel ADC AD7852

The high-speed AD7852 ADC unit (U12) is mapped in 0x118-0x11f. To start a ADC conversion on channel 0, A I/O write, outportb(0x118+?,0); will start a new ADC conversion on the ADC channel ?. The ADC busy signal is routed to J2 pin 11=P13. It goes low for 16 ADC clocks indicating busy. A 16-bit I/O read, inport(0x118); will return the previous ADC conversion result, with only upper 12-bit data D15-D4 valid. A sample program ***ae86_ad.c*** demonstrating the use of the AD7852 is included in `tern\186\samples\ae86`.

Serial ADC P2543

The P2543 ADC unit (U14) provides 11 channels of analog inputs based on the reference voltage supplied to **REF+**. For details regarding the hardware configuration, see the Hardware chapter.

In order to operate the ADC, lines T0, T1, and T2 from the PAL must be used. P11 of the AM186ES must also be configured to be input. This line is also shared with the EEPROM, and left high at power-on/reset. You should be sure not to re-program these pins for your own use. Be careful when using the EEPROM concurrently with the ADC. If the ADC is enabled, the line P11 will be reserved for its use and any attempt to access the EEPROM will time-out after some time.

For a sample file demonstrating the use of the ADC, please see **ae86_ad12.c** in `tern\186\samples\ae86`.

int ae86_ad
Arguments: char c
Return values: int ad_value

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

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

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

```
Ae86_ad(0); // Read from channel 0
chn_0_data = ae86_ad(0)>>4; //Start the next conversion, retrieve value.
```

4.3.6 Digital-to-Analog Conversion

Parallel DAC7625

The high-speed DAC DA7625 (U11) is mapped in 0x110-0x116.

Use outport(0x110, dac); to write upper 12-bit D15-D4 data into DAC channel 1, J4 pin 40

Use outport(0x112, dac); to write upper 12-bit D15-D4 data into DAC channel 2, J4 pin 42

Use outport(0x114, dac); to write upper 12-bit D15-D4 data into DAC channel 3, J4 pin 44

Use outport(0x116, dac); to write upper 12-bit D15-D4 data into DAC channel 4, J4 pin 46

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 **ae86_da.c** in the directory **`tern\186\samples\ae86`**.

Serial DAC LT1446

A LTC 1446 chip is available on the A-Engine86 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 **ae86_da.c** in the directory **`tern\186\samples\ae86`**.

void ae86_da
Arguments: int dat1, int dat2
Return value: none

Argument **dat1** is the current value to drive to channel A of the chip, while argument **dat2** is the value to drive channel B of the chip.

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

4.3.7 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 rollover 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.

int rtc_rds
Arguments: char* realTime
Return value: int error_code

This function places a string of the current value of the real time clock in the char* realTime.
The text string has a format of “year1000 year100 year10 year1 month10 month1 day10 day1 hour10 hour1 min10 min1 second10 second1”. For example” 19991220081020” presents year1999, december, 20th, eight clock 10 minutes, and 20 second.
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.

4.4 Functions in SER0.OBJ/SER1.OBJ

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

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

Two asynchronous serial ports are integrated in the 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 1 and STEP 2. We will use SER1 as the example in the following discussion; any of the interface functions that 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 A-Engine86 provided at the end of this manual.

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

The following table shows the function arguments that express each baud rate, to be used in TERN functions. These are based on a 40 MHz system clock; a 20 MHz system clock would have the baud rates halved.

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.

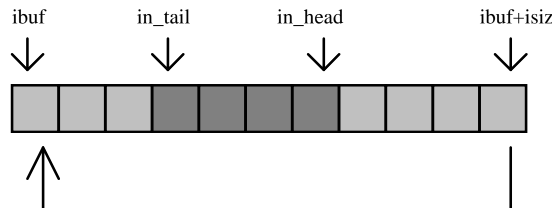


Figure 4.1 Circular ring input buffer

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.

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

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

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

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

Software Interface

Before using the serial ports, they must be initialized.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Miscellaneous Serial Communication Functions

One thing to be aware of in both transmission and receiving of data through the serial port is that TERN drivers only use the basic serial-port communication lines for transmitting and receiving data. Hardware flow control in the form of **CTS** (Clear-To-Send) and **RTS** (Ready-To-Send) is not implemented. There are, however, functions available that allow you to check and set the value of these I/O pins appropriate for whatever form of flow control you wish to implement. Before using these functions, you should once again be aware that the peripheral pin function you are using might not be selected as needed. For details, please refer to the 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, you can re-initialize the serial port with s1_init(); 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.

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 manual for a detailed discussion of other features available to you.

4.5 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 a 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

Function Argument	Baud Rate
4	600
5	1200
6	2400
7	4800
8	9600 (default)
9	19,200
10	31,250
11	62,500
12	125,000
13	250,000

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

Special control registers are used to define how the SCC operates. For a detailed description of registers **MR1** and **MR2**, please see Appendix C 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: **putsern**

putsers_scc
See: **putsersn**

getser_scc
See: **getsern**

getsers_scc
See: **getsersn**

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: ***sn*_cts**

scc_rts
See: ***sn*_rts**

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

scc_close
See: ***sn*_close**

serhit_scc
See: ***sn*_hit**

clean_ser_scc
See: **clean_*sn***

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.

4.6 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.

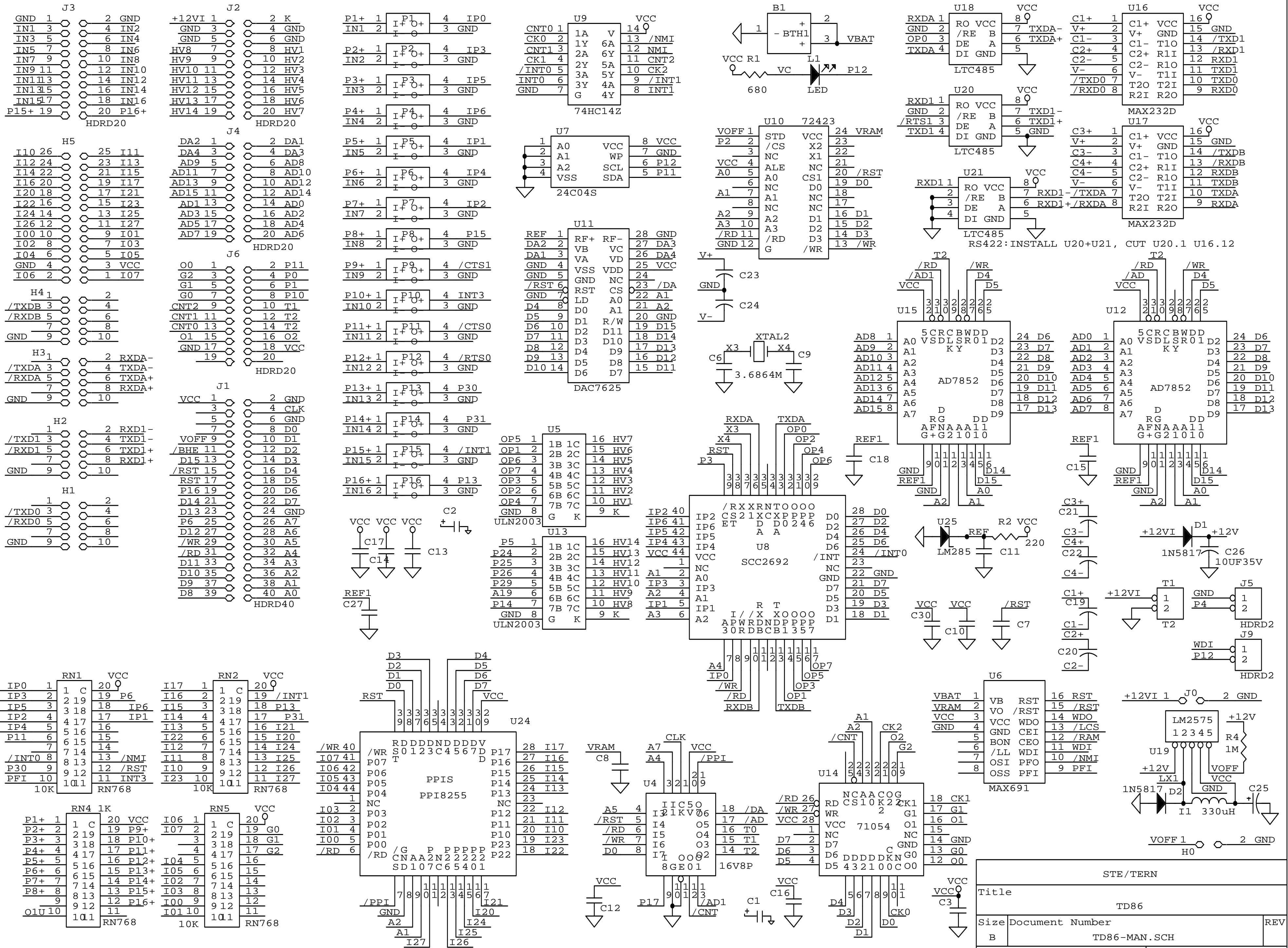
The EEPROM shares line P11 with the ADC. If the ADC is enabled, it can interfere with the EEPROM. The ADC is enabled if I20 is low. In the init function, it is brought high so that you can access the EEPROM. Be aware that if you modify the PPI control register by calling outportb(0x0103, xx); then all of the output lines on the PPI are brought low, including I20, which enables the ADC and disables the EEPROM. If you need to use the EEPROM, be sure to bring I20 high again to disable the ADC (refer to section 3.4.2).

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.



STE/TERN
Title
TD86
Size
B
Document Number
TD86-MAN.SCH
REV
Date: December 12, 2000
Sheet 1 of 1
RS422:INSTALL U20+U21, CUT U20.1 U16.12
74HC14Z
24C04S
DAC7625
72423
LTC485
MAX232D
AD7852
SCC2692
ULN2003
PPI8255
16V8P
71054
MAX691
LM2575
LM285
1N5817
3.6864M
HDRD20
HDRD40
HDRD2
RN768