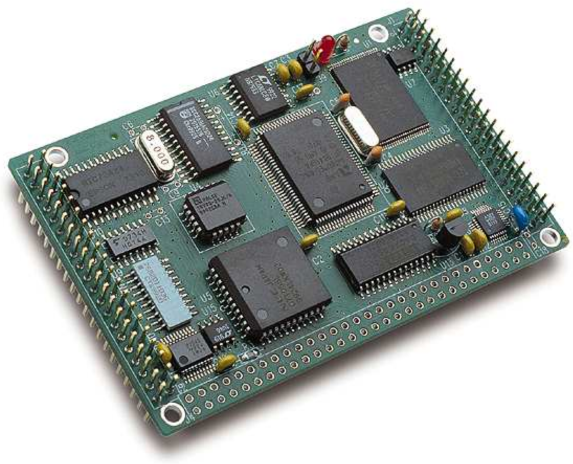
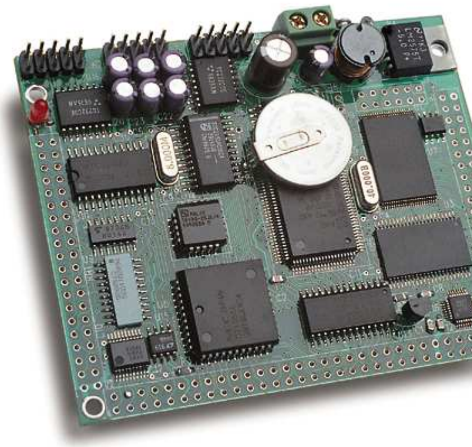
A-Engine86™
&
A-Engine86-P™

16-bit Controller with 16-bit SRAM & Flash, 300 KHz ADC & DAC
Based on the Am186ES



Technical Manual



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

Version 2.00

June 24, 2010

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

Measuring 3.6 x 2.3 x 0.3 inches, the ***A-Engine*86** (*AE86*) is a C/C++ programmable microprocessor module based on a 40 MHz, 16-bit CPU (Am186ES, AMD). Features such as its low cost, compact size, surface-mount flash, high performance, and reliability make the *AE86* ideal for industrial process control and high-speed data acquisition. It is designed for embedded applications that require compactness, low power consumption, and high reliability.

The **A-Engine86-P (AE86-P)** is a new version of the ***A-Engine*86** with on-board regulated 5-volt power and either RS232 or 485 drivers. These new additions eliminate the need for the VE232. Measuring 3.6 x 2.9 x 0.3 inches, the **AE86-P** is ideal for applications in which space is very limited, where the VE232's extra height cannot be accommodated.

The *AE86* is a new design based on the popular original A-Engine (AE-20/AE-40). The original A-Engine is based on the Am188ES processor with internal 16-bit data path and external 8-bit data bus. The new *AE86* is based on the 40 MHz Am186EX processor, which has both internal and external 16-bit data path. Overall, the *AE86* is about 100% faster than the 40MHz A-Engine, in terms of both I/O and computation applications. The *AE86* provides a true 16-bit data bus at J1 20x2 header The *A-Engine*86 is an ideal upgrade for the A-Engine, V25-Engine, or C-Engine, 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) or Development Kit (DV).

The *AE86* can be integrated into an OEM product as a processor core component. It also can be used to build a smart sensor, or can act as a node in a distributed microprocessor system.

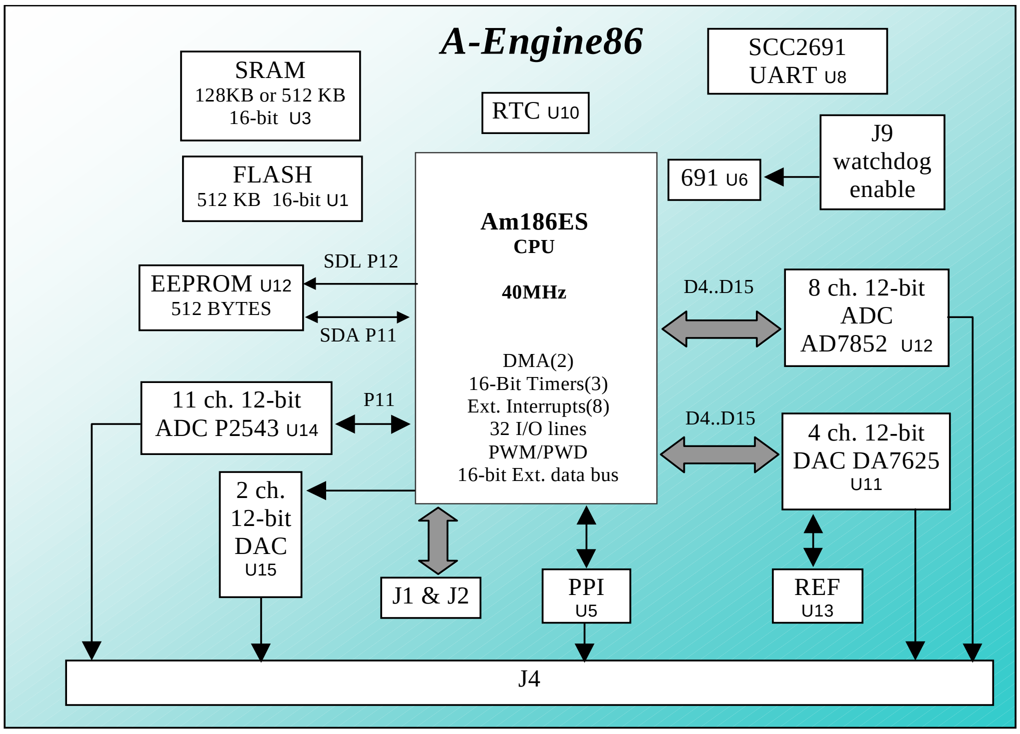


Figure 1.1 Functional block diagram of the A-Engine86

In addition to offering a 16-bit external data bus, the *AE86* supports on-board 512 KB 16-bit Flash and up to 512 KB 16-bit battery-backed SRAM. All chips are surface-mounted, without any sockets. The on-board Flash has a protected boot loader and can be easily programmed in the field via serial link. Users can download a kernel into the Flash for remote debugging. With the DV and ACTF Flash Kit support, user application codes can be easily field-programmed into and run out of the Flash.

A real-time clock* (RTC72423) provides information on the year, month, date, hour, minute, second, and 1/64 second. A 512-byte serial EEPROM is included on-board.

Two DMA-driven serial ports from the Am186ES support high-speed, reliable serial communication at a rate of up to 115,200 baud. An optional UART SCC2691 may be added in order to have a third UART on-board. All three serial ports support 8-bit and 9-bit communication.

There are three 16-bit programmable timers/counters and a watchdog timer. Two timers can be used to count or time external events, at a rate of up to 10 MHz, or to generate non-repetitive or variable-duty-cycle waveforms as PWM outputs. Pulse Width Demodulation (PWD), a distinctive feature, 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 *AE86* provides 32 user-programmable, multifunctional I/O pins from the CPU, plus 24 bi-directional user-definable I/O lines from a PPI (82C55). Schmitt-trigger inverters are provided for six external interrupt inputs, to increase noise immunity and transform slowly-changing input signals into fast-changing and jitter-free signals. A supervisor chip with power failure detection, a watchdog timer, an LED, and expansion ports are on-board.

A high-speed, up to 300K samples per second, 8-channel, 12-bit parallel ADC* (AD7852) can be installed. This ADC includes sample-and-hold and precision internal reference, and has an input range of 0-5 V. The ***AE86*** also supports a 4-channel, high-speed parallel DAC* (DA7625, 0-2.5V).

An optional 12-bit serial ADC (P2543) has 11 channels of analog inputs with sample-and-hold and a 5V reference that facilitate ratiometric conversion, scaling, and isolation of analog circuitry from logic and supply noise, supporting conversion up to a sample rate of approximately 10 KHz. An optional 2-channel, 12-bit serial DAC (LT1446) that provides 0-4.095V analog voltage outputs capable of sinking or sourcing 5mA is also available. The PPI chip (82C55) providing 24 user programmable I/O lines can interface to another processor module, or to an LCD and keypad(s).

An optional ***VE232*** interface board can provide regulated 5V power and RS-232/RS-485 drivers for the *AE86*.

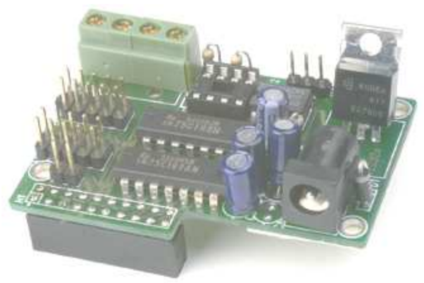


Figure 1.2 The VE232 interface board

The ***A-Engine*86** can be installed on TERN controllers, such as the ***P300, PowerDrive***, ***PC-Co***, ***LittleDrive***, or ***MotionC*** (see Figure 1.3). TERN also offers custom hardware and software design, based on the ***A-Engine*86** or other TERN controllers.

The **AE86-P** supports an optional LM2575 switching regulator (refer to U19 in Appendix A). The switching regulator consumes less power and generates less heat than the standard linear regulator. Furthermore, the switching regulator can be disabled to power-off the **AE86-P** via the VOFF pin of the real-time clock. The battery-backed real-time clock can be programmed to power-on the A-Enginre86-P (*see sample program poweroff.c*)

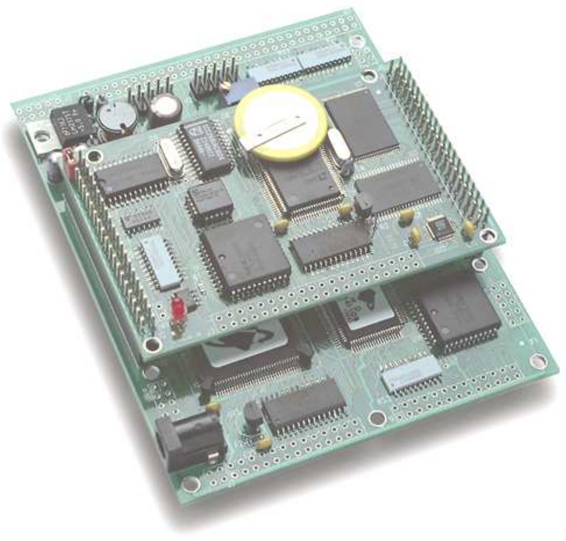


Figure 1.3 An A-Engine86 installed on the MotionC2140

1.2 Features

• Dimensions: 3.6 x 2.3 x 0.3 inches
• 40 MHz, 16-bit CPU (Am186ES), Intel 80x86 compatible
• Easy to program in C/C++
• Power consumption: 190 mA at 5V
• Power-save mode: 30 mA at 5V
• Power input: +5V regulated DC, or
+ 9V to +12V unregulated DC with VE232 interface board installed*
• Up to 512 KB 16-bit SRAM, 512 KB 16-bit Flash *
• 8-channel 300 KHz parallel 12-bit ADC (AD7852) with 0-5V analog input*
• 4-channel 200 KHz parallel 12-bit DAC (DA7625) with 0-2.5V analog output*
• 2 channels serial 12-bit DAC (LT1446), 10 KHz *
• 11 channels serial 12-bit ADC (P2543), 10 KHz *
• 16-bit external data bus expansion port
• Up to 420 MB memory expansion via **MemCard-A™**
• Up to 3 serial ports (2 from Am186ES, plus one optional SCC2691 UART) support 8-bit or 9-bit asynchronous communication *
• 2 high-speed PWM outputs and Pulse Width Demodulation
• 8 external interrupt inputs, 3 16-bit timer/counters
• 32 multifunctional I/O lines from Am186ES
• 24 bi-directional I/O lines from 82C55 PPI

- 512-byte serial EEPROM
- Supervisor chip (691) for power failure, reset and watchdog
- Real-time clock (RTC72423), lithium coin battery*
- VE232 add-on board for regulated 5V power & RS-232/485 drivers*

  * optional

1.3 Physical Description

The physical layout of the A-Engine86 is shown in Figure 1.4.

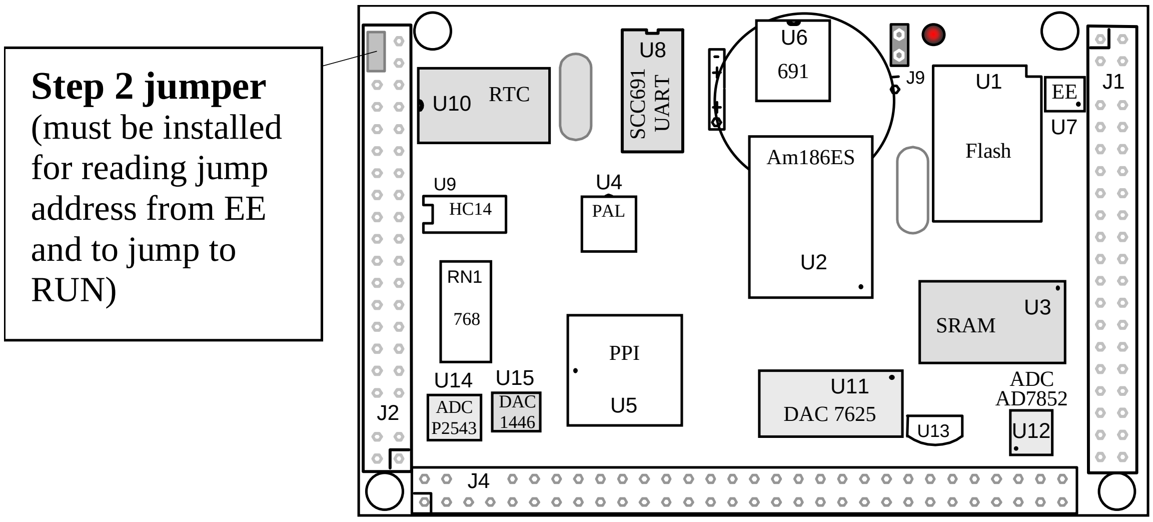


Figure 1.4 Physical layout of the A-Engine86

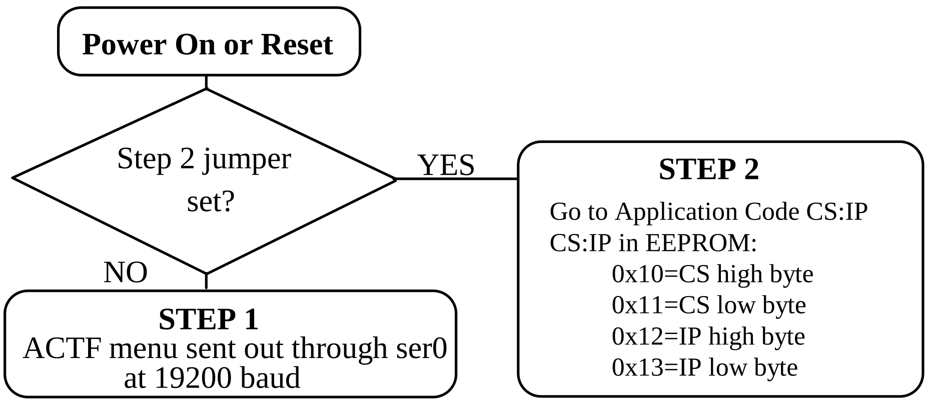


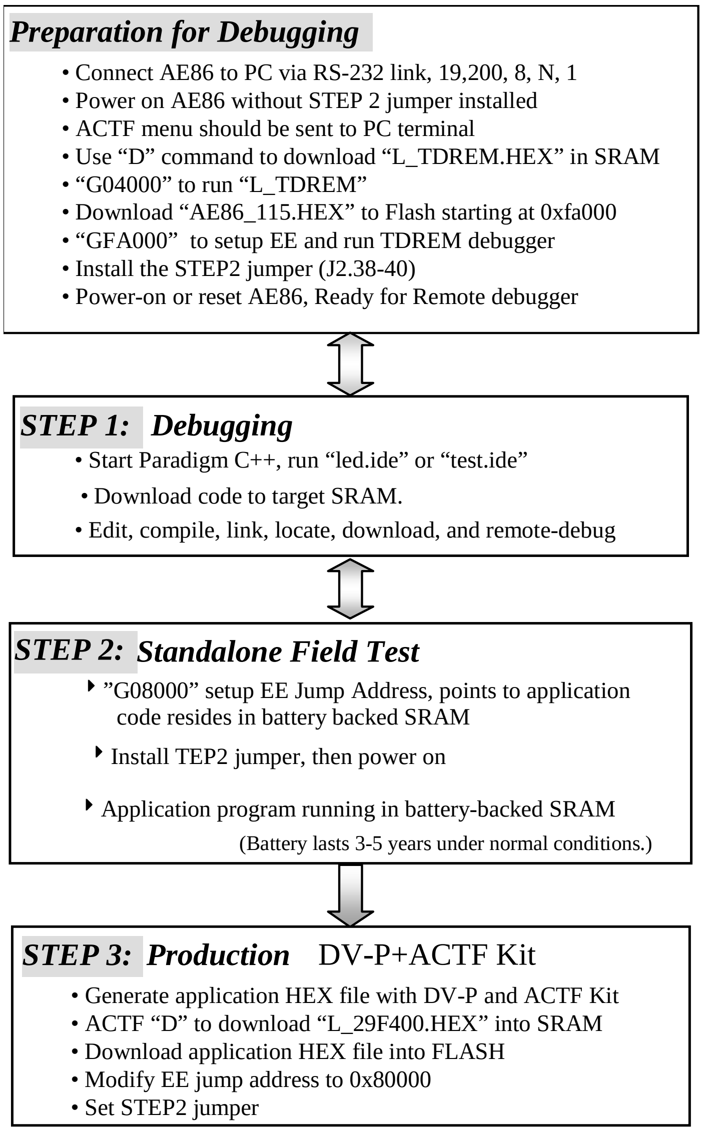
Figure 1.5 Flow chart for ACTF operation

The “ACTF boot loader” resides in the top protected sector of the 512KB 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 "TDREM_AE.HEX" can be downloaded, therefore residing in "0xFA000" of the 512KB on-board flash chip.

1.4 A-Engine86 Programming Overview

Steps for AE86-based product development:



Preparation for Debugging

- Connect AE86 to PC via RS-232 link, 19,200, 8, N, 1
- Power on AE86 without STEP 2 jumper installed
- ACTF menu should be sent to PC terminal
- Use "D" command to download "L_TDREM.HEX" in SRAM
- "G04000" to run "L_TDREM"
- Download "AE86_115.HEX" to Flash starting at 0xfa000
- "GFA000" to setup EE and run TDREM debugger
- Install the STEP2 jumper (J2.38-40)
- Power-on or reset AE86, Ready for Remote debugger

STEP 1: Debugging

- Start Paradigm C++, run "led.ide" or "test.ide"
- Download code to target SRAM.
- Edit, compile, link, locate, download, and remote-debug

STEP 2: Standalone Field Test

- "G08000" setup EE Jump Address, points to application code resides in battery backed SRAM
- Install TEP2 jumper, then power on
- Application program running in battery-backed SRAM

(Battery lasts 3-5 years under normal conditions.)

STEP 3: Production DV-P+ACTF Kit

- Generate application HEX file with DV-P and ACTF Kit
- ACTF "D" to download "L_29F400.HEX" into SRAM
- Download application HEX file into FLASH
- Modify EE jump address to 0x80000
- Set STEP2 jumper

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. For production, the user must produce an ACTF-downloadable HEX file for the application, based on the DV-P+ACTF Kit. The "STEP2" jumper (J2 pins 38-40) must be installed for every production-version board.

Step 1 settings

In order to talk to AE86 with Paradign C++, the AE86 must meet these requirements:

1) AE_0_115.HEX(-P version) 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 Jump Address for the AE_0_115.HEX with starting address of 0xfa000.

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

For further information on programming the A-Engine86, refer to the Software chapter.

1.5 VE232

The VE232 is an interface board for the A-Engine86 that provides regulated +5V DC power and RS-232/485 drivers. It converts TTL signals to and from RS-232 signals. You do not need the VE232 if you are using the A-Engine86 installed on another TERN controller such as the LittleDrive, MotionC, PowerDrive, or SensorWatch.

The VE232, shown in Figure 1.6, measures 2.3 x 1.57 inches. A wall transformer (9V, 300 mA) with a center negative DC plug (Ø=2.0 mm) should be used to power the A-Engine86 via the VE232. The VE232 connects to A-Engine86 via H1 (2x10 header). SER0 (J2) and SER1 (J3) on the VE232 are 2x5-pin headers for serial ports SER0 and SER1. SER0 is the default programming port.

While the VE232 is installed, J2 pins 38-40 of the A-Engine86 are connected to H2 of the V232. You may use H2 for the Step 2 jumper.

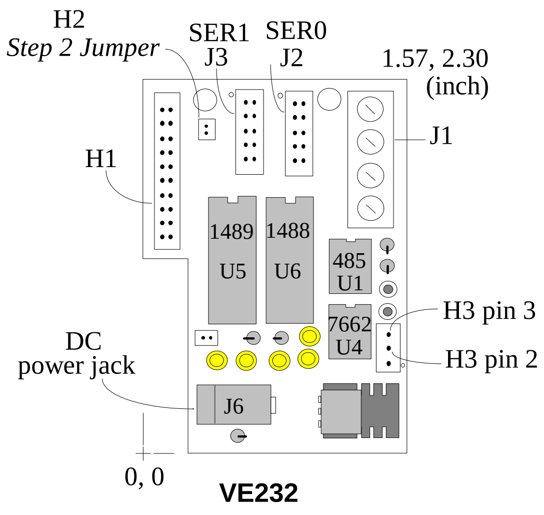


Figure 1.6 The VE232, an interface card for the A-Engine86

For further information on the VE232, please refer to Appendix B and to the VE232 schematic at the end of this manual.

1.6 Minimum Requirements for A-Engine86 System Development

1.6.1 Minimum Hardware Requirements

- PC or PC-compatible computer with serial COMx port that supports 115,200 baud
- A-Engine86 controller
- VE232 interface board*
- PC-V25 serial cable (RS-232; DB9 connector for PC COM port and IDE 2x5 connector for controller)
- center negative wall transformer (+9V, 500 mA)

* NOTE: the VE232 is not needed if you are using the AE86 installed on another controller

1.6.2 Minimum Software Requirements

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

With the EV Kit, you can program and debug the A-Engine86 in Step One and Step Two, but you cannot run Step Three. In order to generate an application Flash file and complete a project, you will need both the Development Kit (DV-P Kit) and the ACTF Flash Kit.

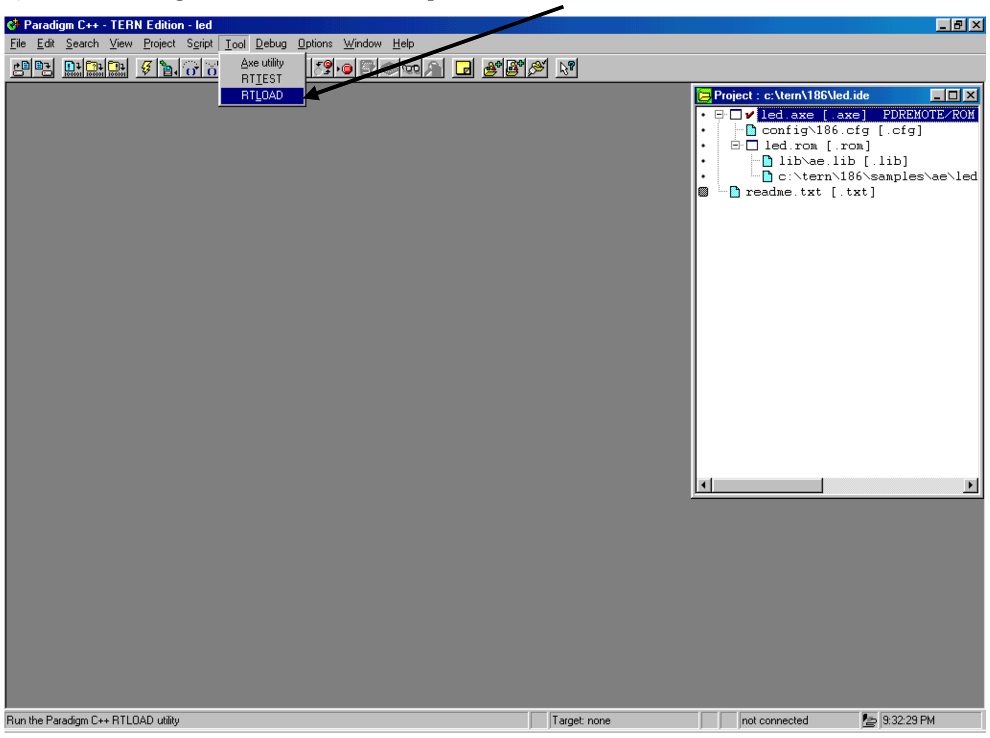
Chapter 2: Installation

2.1 Software Installation

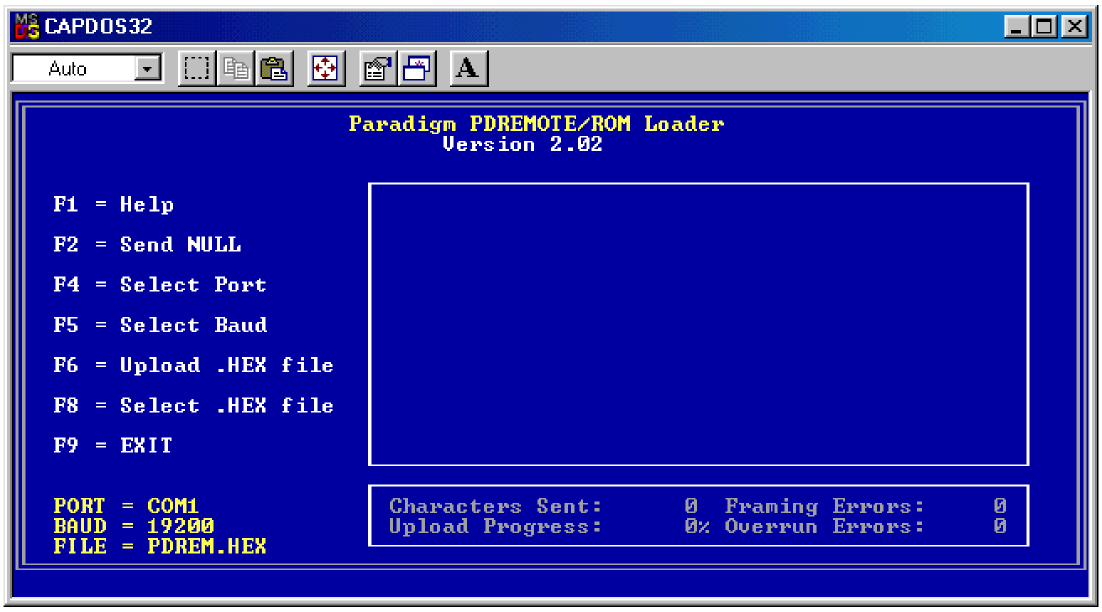
Please refer to the Technical manual for the "C/C++ Development Kit and Evaluation Kit for TERN Embedded Microcontrollers" for information on installing software.

Prepare AE86/AC86 for Paradigm C++ TERN Edition
By manufacture default, AE86/AC86 are ready for Paradigm C++ debug with STEP2 jumper installed. EE Jump Address setup for 0x80000, and 5860_115 debug kernel resides in Flash starting 0x80000. Power on, the on-board LED should blink twice indicating ready for debug. You DO NOT have to prepare and download debug kernel into flash again. You can SKIP the operation discussed below.

1) Start Paradigm C++. Select from Top manu: Tool, RTLOAD,


Paradigm C++ - TERN Edition - led
File Edit Search View Project Script Tool Debug Options Window Help
Axe utility
RTTEST
RTLOAD
Project : c:\tern\186\led.ide
led.axe [.axe] PDREMOTE/ROM
config\186.cfg [.cfg]
led.rom [.rom]
lib\ae.lib [.lib]
c:\tern\186\samples\ae\led
readme.txt [.txt]
Run the Paradigm C++ RTLOAD utility
Target: none
not connected
9:32:29 PM

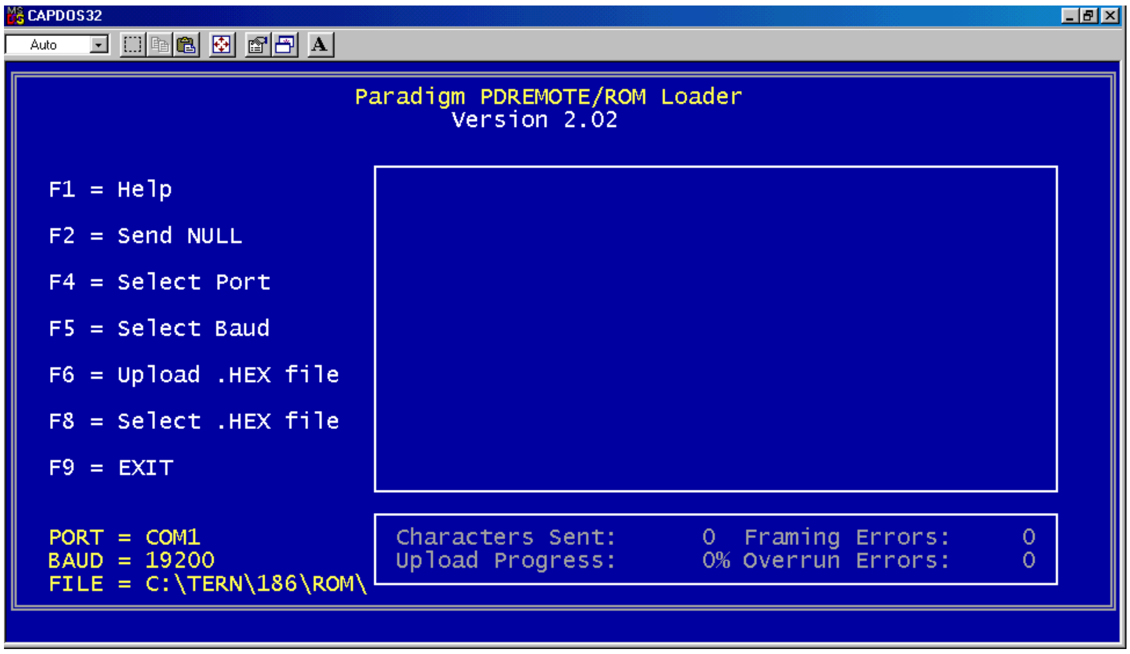
A HEX file Loader window is shown.



```
                    Paradigm PDREMOTE/ROM Loader
                           Version 2.02

F1  = Help
F2  = Send NULL
F4  = Select Port
F5  = Select Baud
F6  = Upload .HEX file
F8  = Select .HEX file
F9  = EXIT

PORT  = COM1           Characters Sent:        0  Framing Errors:     0
BAUD  = 19200          Upload Progress:       0%  Overrun Errors:     0
FILE  = PDREM.HEX
```

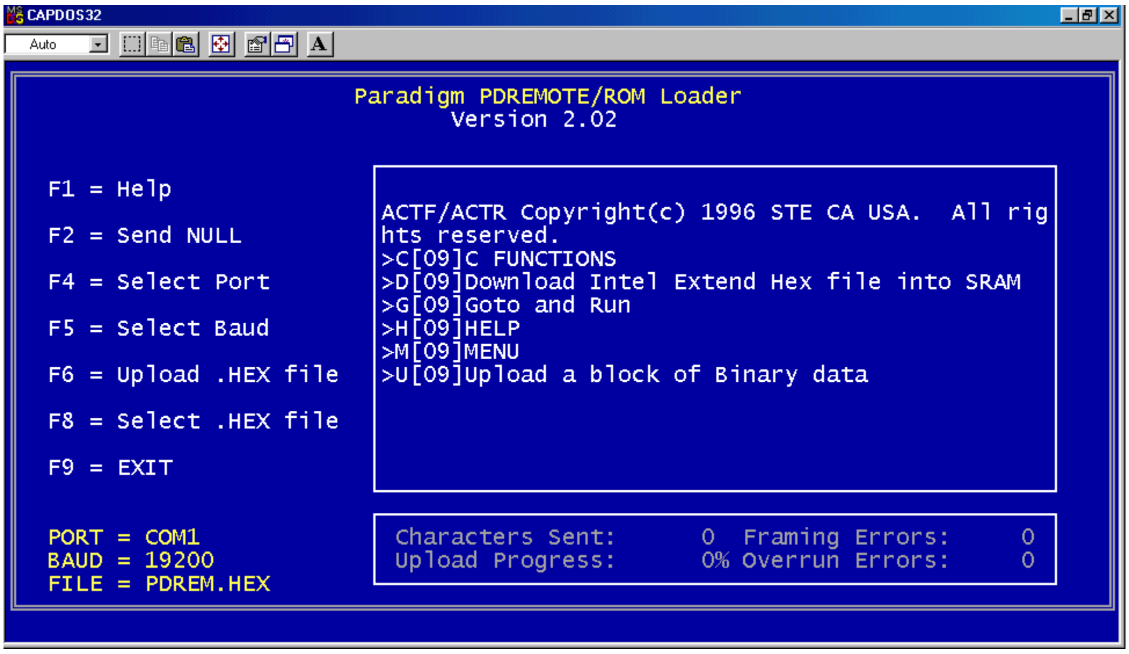
2) F5=Select Baud, to setup 19200.

3) F8=Select .HEX file from c:\tern\186\rom\ae86\L_TDREM.HEX



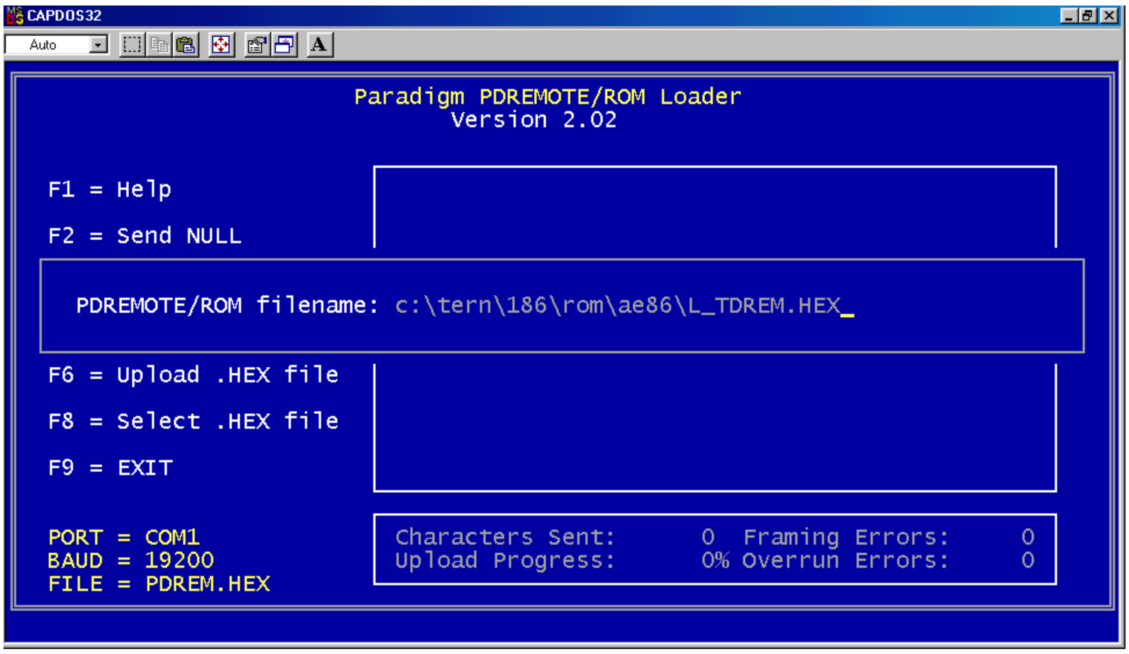
```
                    Paradigm PDREMOTE/ROM Loader
                           Version 2.02

F1  = Help
F2  = Send NULL
F4  = Select Port
F5  = Select Baud
F6  = Upload .HEX file
F8  = Select .HEX file
F9  = EXIT

PORT  = COM1             Characters Sent:        0  Framing Errors:     0
BAUD  = 19200            Upload Progress:       0%  Overrun Errors:     0
FILE  = C:\TERN\186\ROM\
```

3) Power on the AE86 with the STEP2 jumper off, the ACTF menu will show up.



```
                 Paradigm PDREMOTE/ROM Loader
                        Version 2.02

F1 = Help
                      ACTF/ACTR Copyright(c) 1996 STE CA USA.  All rig
F2 = Send NULL        hts reserved.
                      >C[09]C FUNCTIONS
F4 = Select Port      >D[09]Download Intel Extend Hex file into SRAM
                      >G[09]Goto and Run
F5 = Select Baud      >H[09]HELP
                      >M[09]MENU
F6 = Upload .HEX file >U[09]Upload a block of Binary data

F8 = Select .HEX file

F9 = EXIT

PORT = COM1            Characters Sent:       0  Framing Errors:     0
BAUD = 19200           Upload Progress:      0%  Overrun Errors:     0
FILE = PDREM.HEX
```

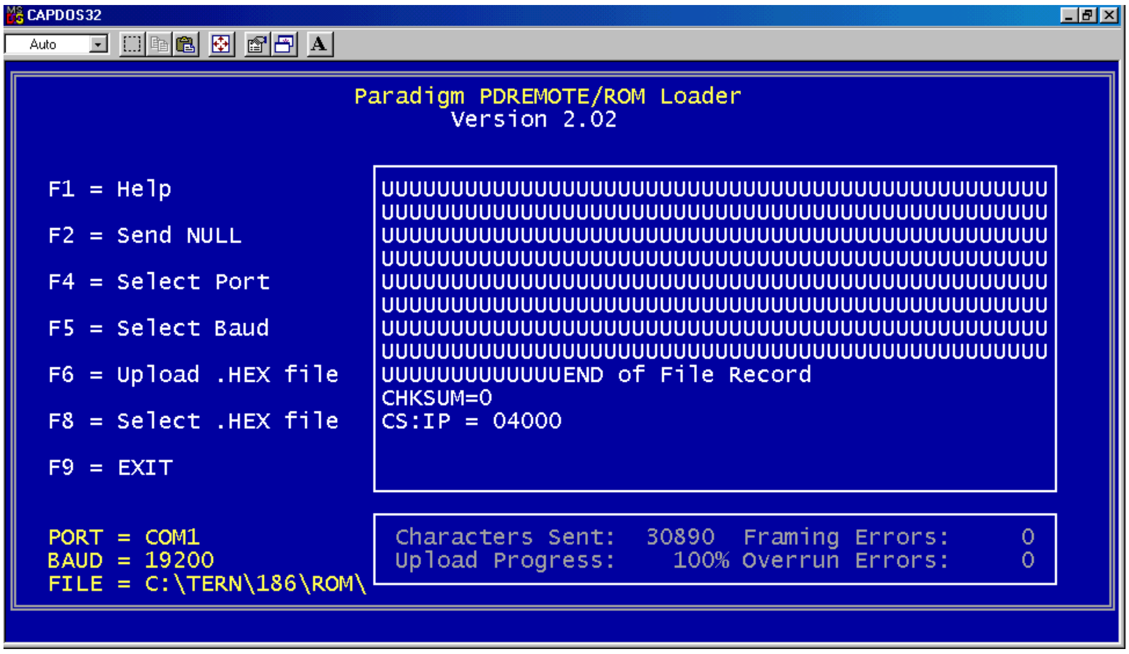
4) Caps Lock on your PC keyboard, Type “D” command



```
                 Paradigm PDREMOTE/ROM Loader
                        Version 2.02

F1 = Help

F2 = Send NULL

  PDREMOTE/ROM filename: c:\tern\186\rom\ae86\L_TDREM.HEX_

F6 = Upload .HEX file

F8 = Select .HEX file

F9 = EXIT

PORT = COM1            Characters Sent:       0  Framing Errors:     0
BAUD = 19200           Upload Progress:      0%  Overrun Errors:     0
FILE = PDREM.HEX
```

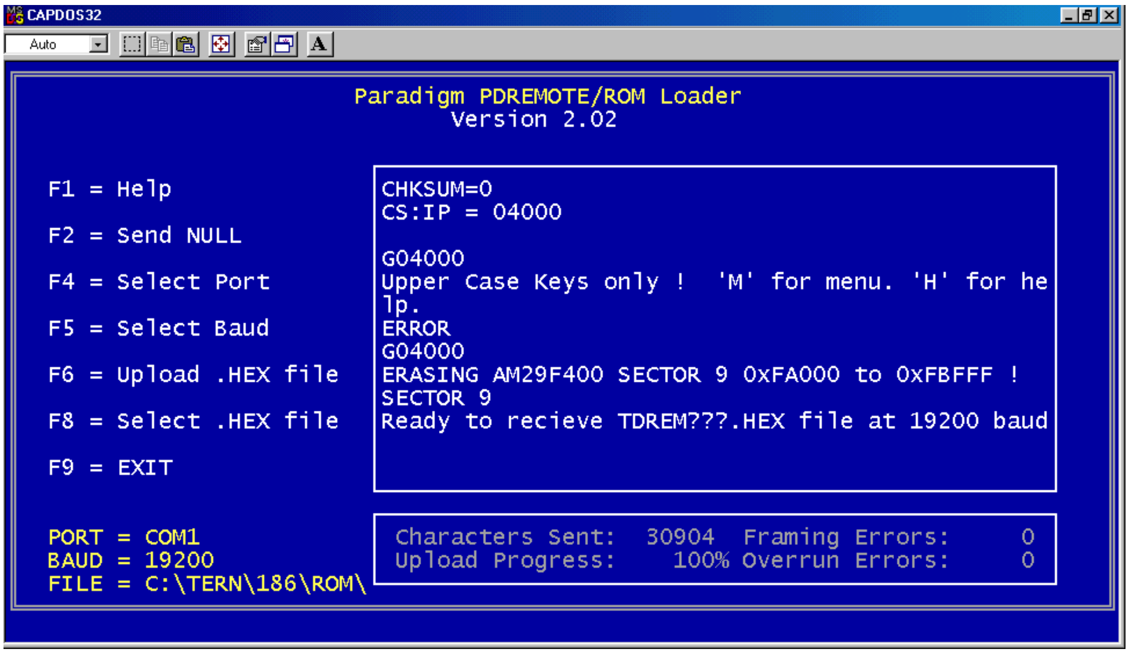
F8=Select .HEX file, from c:\tern\186\rom\ae86\L_TDREM.HEX.

F6 = Upload .HEX file, from PC to AE86 SRAM.



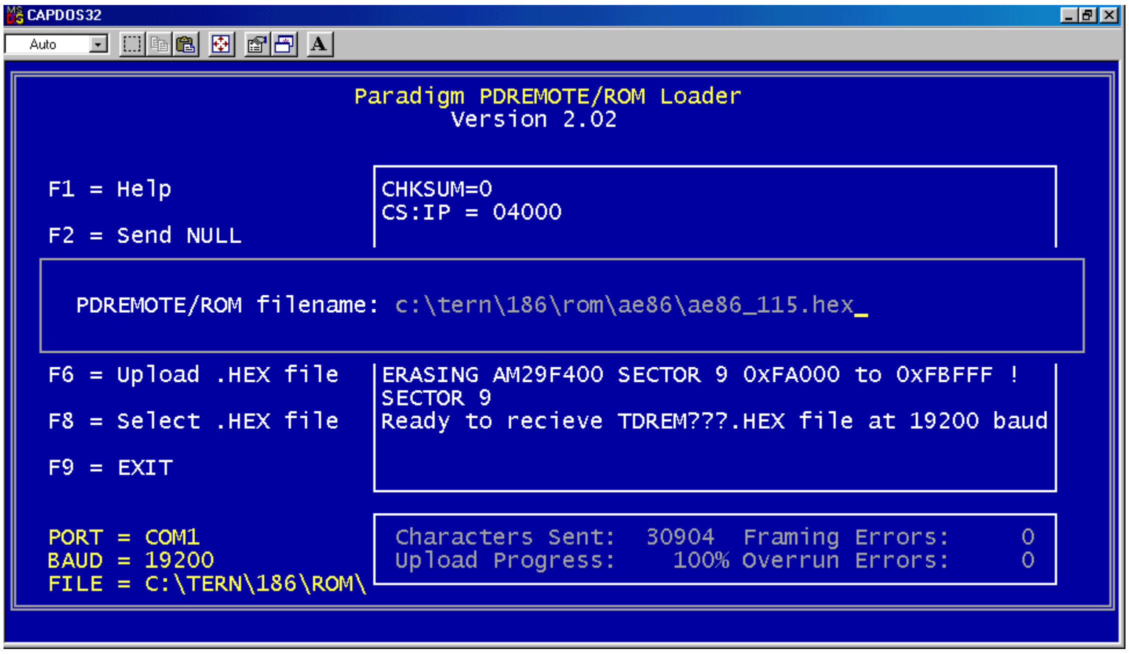
```
CAPDOS32
                    Paradigm PDREMOTE/ROM Loader
                           Version 2.02

F1 = Help            UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
                     UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
F2 = Send NULL       UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
                     UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
F4 = Select Port     UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
                     UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
F5 = Select Baud     UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
                     UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
F6 = Upload .HEX file UUUUUUUUUUUUUEND of File Record
                     CHKSUM=0
F8 = Select .HEX file CS:IP = 04000

F9 = EXIT

PORT = COM1           Characters Sent:  30890  Framing Errors:    0
BAUD = 19200          Upload Progress:   100% Overrun Errors:    0
FILE = C:\TERN\186\ROM\
```

"G04000" to run the "L_TDREM" in SRAM. For some reason, the first "G04000" always has error. Do the "G04000" again, then erase the AE86 Flash sector 0xFA000 to 0xFBFFF, and ready for programming Flash with the new DEBUG file (c:\tern\186\rom\ae86\ae86_115.hex).



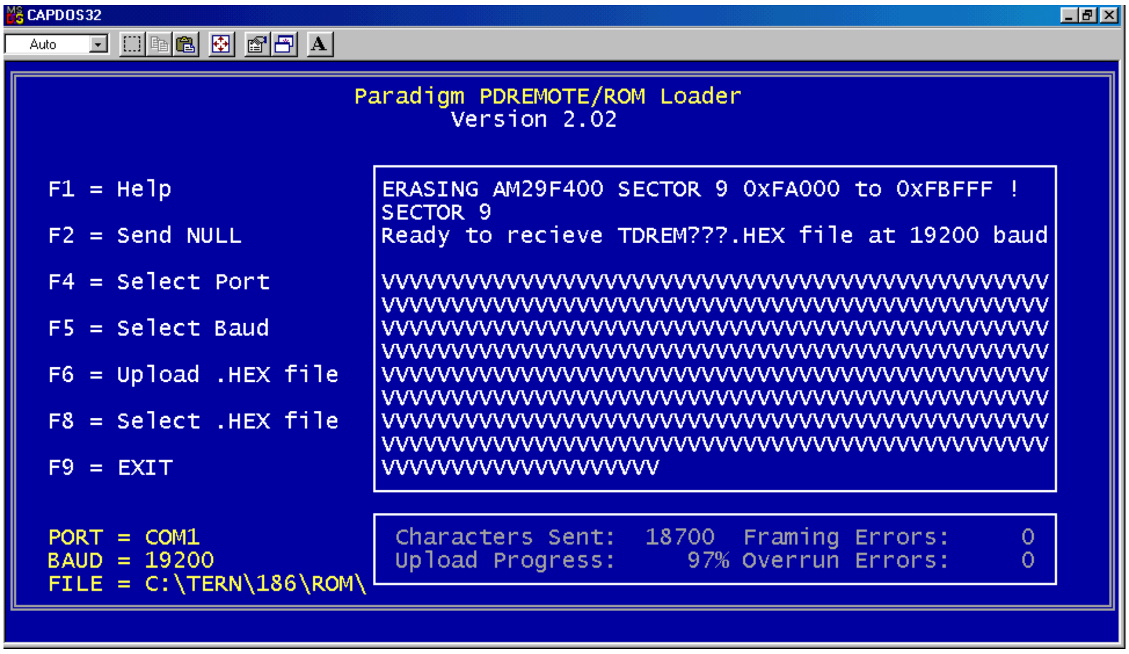
```
CAPDOS32
                    Paradigm PDREMOTE/ROM Loader
                           Version 2.02

F1 = Help            CHKSUM=0
                     CS:IP = 04000
F2 = Send NULL
                     G04000
F4 = Select Port     Upper Case Keys only !  'M' for menu. 'H' for he
                     lp.
F5 = Select Baud     ERROR
                     G04000
F6 = Upload .HEX file ERASING AM29F400 SECTOR 9 0xFA000 to 0xFBFFF !
                     SECTOR 9
F8 = Select .HEX file Ready to recieve TDREM???.HEX file at 19200 baud

F9 = EXIT

PORT = COM1           Characters Sent:  30904  Framing Errors:    0
BAUD = 19200          Upload Progress:   100% Overrun Errors:    0
FILE = C:\TERN\186\ROM\
```

F8= Select .HEX file, from c:\tern\186\rom\ae86\ae86_115.hex.



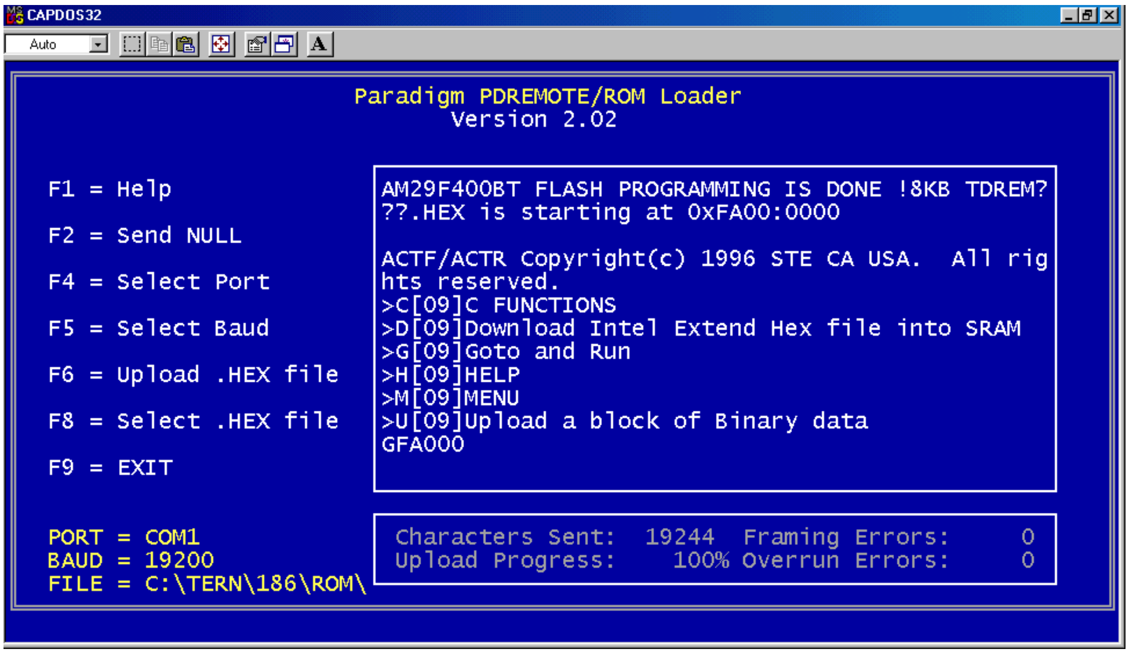
```
                    Paradigm PDREMOTE/ROM Loader
                           Version 2.02

 F1 = Help                CHKSUM=0
                          CS:IP = 04000
 F2 = Send NULL

   PDREMOTE/ROM filename: c:\tern\186\rom\ae86\ae86_115.hex_

 F6 = Upload .HEX file    ERASING AM29F400 SECTOR 9 0xFA000 to 0xFBFFF !
                          SECTOR 9
 F8 = Select .HEX file    Ready to recieve TDREM???.HEX file at 19200 baud

 F9 = EXIT

 PORT = COM1               Characters Sent:  30904  Framing Errors:    0
 BAUD = 19200              Upload Progress:   100% Overrun Errors:    0
 FILE = C:\TERN\186\ROM\
```

F6=Upload .HEX file to program the AE86_115 debug kernel into Flash starting address 0xFA000.



```
                    Paradigm PDREMOTE/ROM Loader
                           Version 2.02

 F1 = Help                ERASING AM29F400 SECTOR 9 0xFA000 to 0xFBFFF !
                          SECTOR 9
 F2 = Send NULL           Ready to recieve TDREM???.HEX file at 19200 baud

 F4 = Select Port         VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
                          VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
 F5 = Select Baud         VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
                          VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
 F6 = Upload .HEX file    VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
                          VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
 F8 = Select .HEX file    VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
                          VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
 F9 = EXIT                VVVVVVVVVVVVVVVVVVVV

 PORT = COM1               Characters Sent:  18700  Framing Errors:    0
 BAUD = 19200              Upload Progress:    97% Overrun Errors:    0
 FILE = C:\TERN\186\ROM\
```

After programming the Flash, AE86 automatically resets.



```
                    Paradigm PDREMOTE/ROM Loader
                           Version 2.02

F1 = Help              AM29F400BT FLASH PROGRAMMING IS DONE !8KB TDREM?
                       ??.HEX is starting at 0xFA00:0000
F2 = Send NULL
                       ACTF/ACTR Copyright(c) 1996 STE CA USA.  All rig
F4 = Select Port       hts reserved.
                       >C[09]C FUNCTIONS
F5 = Select Baud       >D[09]Download Intel Extend Hex file into SRAM
                       >G[09]Goto and Run
F6 = Upload .HEX file  >H[09]HELP
                       >M[09]MENU
F8 = Select .HEX file  >U[09]Upload a block of Binary data
                       GFA000
F9 = EXIT


PORT = COM1              Characters Sent:  19244  Framing Errors:    0
BAUD = 19200             Upload Progress:   100% Overrun Errors:     0
FILE = C:\TERN\186\ROM\
```

"GFA000" to setup the EE Jump Address, and runs the DEBUG kernel. The on-board LED should blink twice and then stay on, indicating AE86 is ready for DEBUG.

Power off the controller. Install the STEP2 Jumper, then power on, the LED blink twice.

Use F9 = Exit.

The AE86 is ready for using Paradigm C++ TERN Edition to download, debug, and run.

There are two sample project in the c:\tern\186 directory (default working directory):

led.ide and ***test.ide.***

Go to the File and open the sample project file, then build and download.

There are many sample programs under c:\tern\186\samples\directories for TERN controllers.

After debug your application code, you can setup the AE86 to run in Standalone Mode.

How to run the controller in Standalone Mode

By default, the Paradigm C++ TERN Edition will download your application code starting at 0x08000 in the battery backed SRAM.

Power off AE86. Remove STEP2 jumper.

PC side, start the TOOL, RTLOAD.

Power on AE86 again without STEP2 jumper, then the ACTF manu should show up.

"G08000" to setup the Jump Address and run your application.

Power off the AE86, Install the STEP2 Jumper. Then every time power on, the controller will run your application.

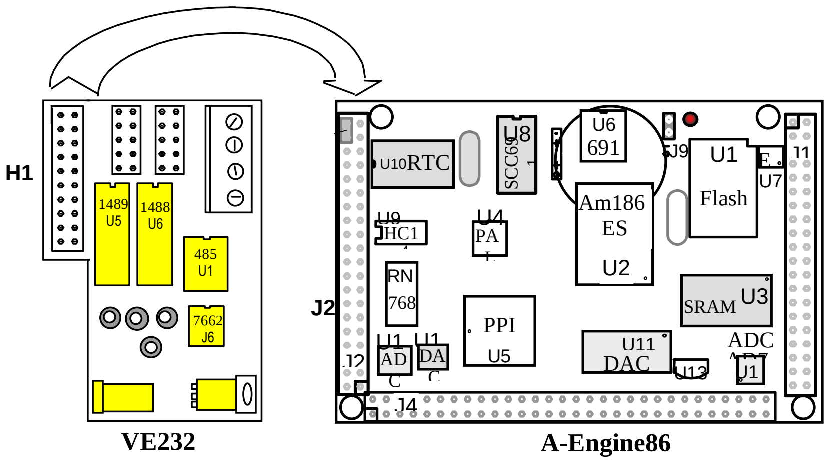
2.2 Hardware Installation

Overview

- Install VE232 (if applicable):
  H1 connector of VE232 installs on J2 of the A-Engine86
- Connect PC-V25 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 A-Engine86 consists primarily of connecting the microcontroller to your PC. For the A-Engine86, the VE232 must be used to supply regulated power and RS-232 drivers to the A-Engine86. If you are using the A-Engine86 installed on another controller, please refer to the technical manual for that controller for installation information.

2.2.1 Connecting the VE232 to the A-Engine86



Install the VE232 interface with the H1 (10x2) socket connector on the upper half of the J2 (dual row header) of the A-Engine86. Figure 2.1 and Figure 2.2 show the VE232 and the A-Engine86 before and after installation.

Figure 2.1 Before installing the VE232 on the A-Engine86

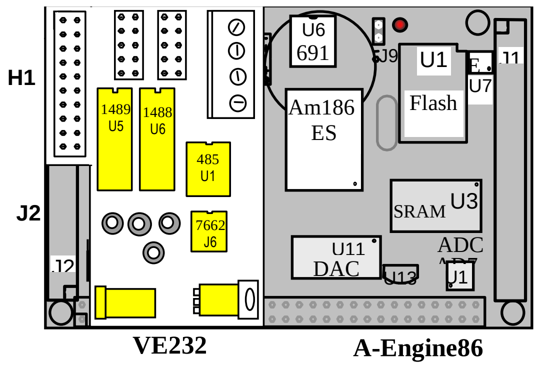


Figure 2.2 After installing the VE232 on the A-Engine86

2.2.2 Connecting the A-Engine86 to the PC

The following diagram (Figure 2.3) illustrates the connection between the A-Engine86, VE232, and the PC. The A-Engine86 is linked to the PC via a serial cable (PC-V25).

The A-Engine86 communicates through SER0 by default. Install the 5x2 IDC connector on the SER0 header of the VE232. ***IMPORTANT:*** *Note that the* ***red*** *side of the cable must point to pin 1 of the VE232 H1 header*. The DB9 connector should be connected to one of your PC's COM Ports (COM1 or COM2).

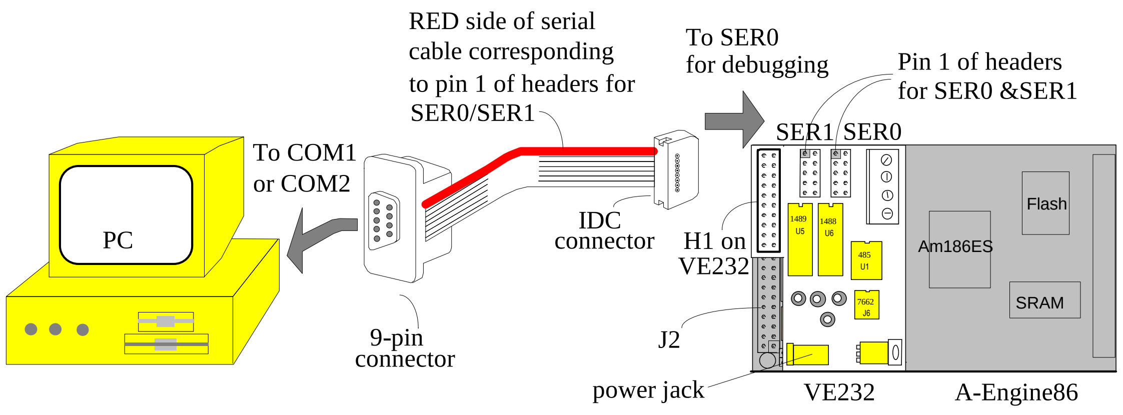


Figure 2.3 Connecting the A-Engine86 and VE232 to the PC

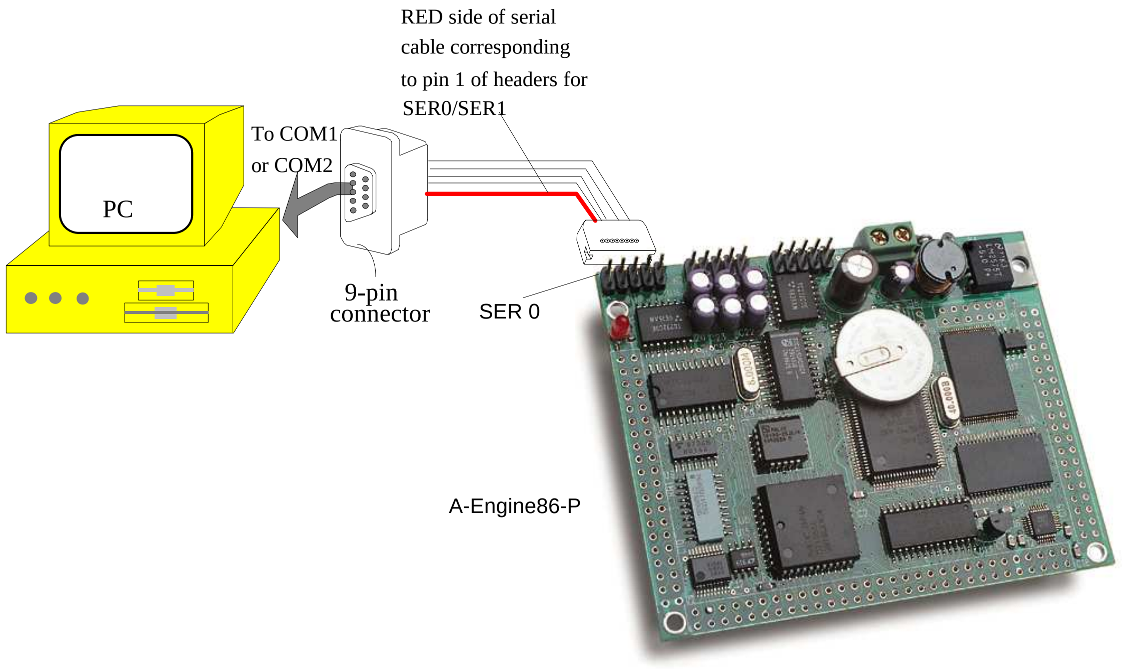

RED side of serial cable corresponding to pin 1 of headers for SER0/SER1
To COM1 or COM2
PC
9-pin connector
SER 0
A-Engine86-P

Figure 2.4 Connecting the A-Engine86-P to the PC

2.2.3 Powering-on the A-Engine86

Connect a wall transformer +9V DC output to the VE232 DC power jack.

The on-board LED should blink twice and remain on after the A-Engine86 is powered-on or reset, as shown in Figure 2.5.

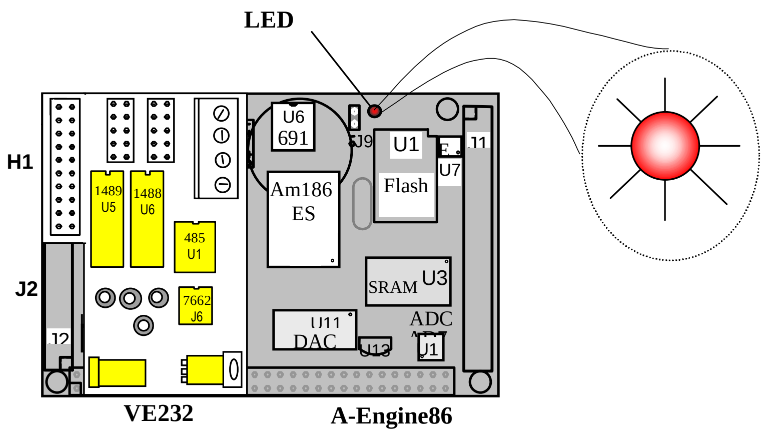


Figure 2.5 The LED blinks twice after the A-Engine86 is powered-on or reset

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

3.2.1 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 A-Engine86.

CLKOUTA remains active during reset and bus hold conditions. The A-Engine86 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.

3.2.2 External Interrupts and Schmitt Trigger Input Buffer

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

/INT0, J2 pin 8, is used by SCC2691 UART, if it is installed.
/INT1, J2 pin 6
/INT2, J2 pin 19
/INT3, J2 pin 21
/INT4, J2 pin 33
INT5=P12=DRQ0, J2 pin 5, used by A-Engine86 as output for LED/EE/HWD
INT6=P13=DRQ1, J2 pin 11, ADC U12 Busy
/NMI, J2 pin 7

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

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

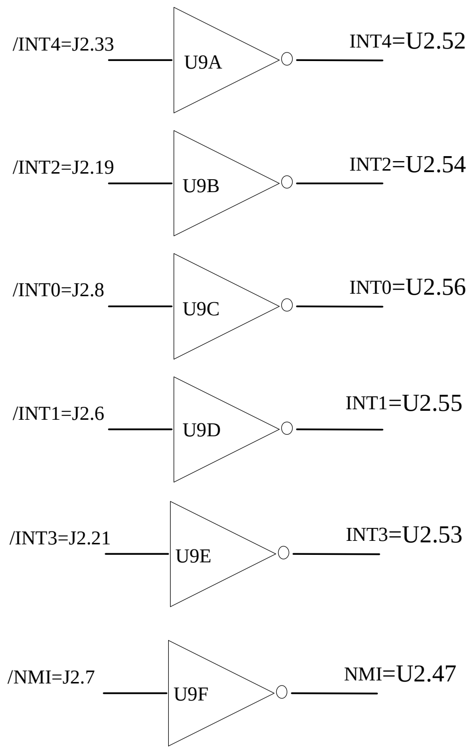


Figure 3.1 External interrupt inputs

The A-Engine86 uses vector interrupt functions to respond to external interrupts. Refer to the Am186ES User's manual for information about interrupt vectors.

3.2.3 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 transfers
- Odd, even, and no parity
- One stop bit
- Error detection
- 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 optional external SCC2691 UART is located in position U8. For more information about the external UART SCC2691, please refer to section 3.4.4 and Appendix C.

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 external pins:

Timer0 output = P10 = J2 pin 12
Timer0 input = P11 = U7 EE pin 5 = U14 ADC pin 16
Timer1 output = P1 = J2 pin 29
Timer1 input = P0 = J2 pin 20

Timer0 input P11 is used and shared by on-board EE and ADC, not recommended for other external use.

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

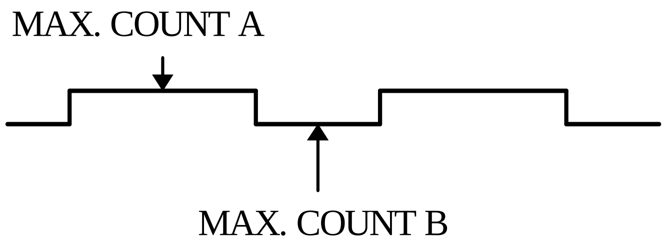
Timer2 is not connected to any external pin. It can be used as an internal timer for real-time coding or time-delay applications. It can also prescale timer 0 and timer 1 or be used as a DMA request source.

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.

3.2.5 PWM outputs and PWD

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

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



Pulse Width Demodulation can be used to measure the input signal's high and low phases on the /INT2=J2 pin 19.

3.2.6 Power-save Mode

The A-Engine86 is an ideal core module 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 A-Engine86 has a VOFF signal routed to J1 pin 9. 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 an external switching

power supply that turns the power supply on/off. 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***	***A-Engine86 Pin No.***	***A-Engine86 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	Use as Clock for AD7852
P2	/PCS6/A2	Input with pull-up	J2 pin 24	RTC select
P3	/PCS5/A1	Input with pull-up	J2 pin 15	SCC2691 select
P4	DT/R	Normal	J2 pin 38	Input with pull-up Step 2
P5	/DEN/DS	Normal	J2 pin 30	Input with pull-up
P6	SRDY	Normal	J2 pin 35	Input with pull-down
P7	A17	Normal	U3 pin 22	A17
P8	A18	Normal	U3 pin 23	A18
P9	A19	Normal	J2 pin 10	A19
P10	Timer0 out	Input with pull-down	J2 pin 12	Input with pull-down
P11	Timer0 in	Input with pull-up	U7 EE pin 5	Input with pull-up
P12	DRQ0/INT5	Input with pull-up	J2 pin 5	Output for LED/EE/HWD
P13	DRQ1/INT6	Input with pull-up	J2 pin 11	Input with pull-up
P14	/MCS0	Input with pull-up	J2 pin 37	Input with pull-up
P15	/MCS1	Input with pull-up	J2 pin 23	Input with pull-up
P16	/PCS0	Input with pull-up	J1 pin 19	/PCS0
P17	/PCS1	Input with pull-up	J2 pin 13	PPI, ADC, DAC select
P18	CTS1/PCS2	Input with pull-up	J2 pin 22	Input with pull-up
P19	RTS1/PCS3	Input with pull-up	J2 pin 31	Input with pull-up
P20	RTS0	Input with pull-up	J2 pin 27	Input with pull-up
P21	CTS0	Input with pull-up	J2 pin 36	Input with pull-up
P22	TxD0	Input with pull-up	J2 pin 34	TxD0
P23	RxD0	Input with pull-up	J2 pin 32	RxD0
P24	/MCS2	Input with pull-up	J2 pin 17	Input with pull-up
P25	/MCS3	Input with pull-up	J2 pin 18	Input with pull-up
P26	UZI	Input with pull-up	J2 pin 4	Input with pull-up*
P27	TxD1	Input with pull-up	J2 pin 28	TxD1
P28	RxD1	Input with pull-up	J2 pin 26	RxD1
P29	/CLKDIV2	Input with pull-up	J2 pin 3	Input with pull-up*
P30	INT4	Input with pull-up	J2 pin 33	Input with pull-up
P31	INT2	Input with pull-up	J2 pin 19	Input with pull-up

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

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

Three external interrupt lines are not shared with PIO pins:

INT0 = J2 pin 8
INT1 = J2 pin 6
INT3 = J2 pin 21

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

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

A-Engine86 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 A-Engine86 system for on-board components (Table 3.2). We suggest that you not use these lines unless you are sure that you are not interfering with the operation of such components (i.e., if the component is not installed).

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

Signal	**Pin**	**Function**
P1	Timer1 output	5MHz U12 ADC clock
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 U19 P2543 ADC and U7 24C04 EE data input The ADC and EE data output can be tri-state, while disabled

Signal	Pin	Function
P12	DRQ0/INT5	Output for LED or U7 serial EE clock or Hit watchdog
P13	/INT6	U12 ADC BUSY
P17	/PCS1	U5 PPI 0x100; U11 DAC 0x110; U12 ADC 0x0118;
P22	TxD0	Default SER0 debug
P23	RxD0	Default SER0 debug
/INT0	J2 pin 8	U8 SCC2691 UART interrupt, if U8 is installed

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

3.4 I/O Mapped Devices

3.4.1 I/O Space

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

The default I/O access time is 15 wait states. You may use the function void ***io_wait***(char wait) to define the I/O wait states from 0 to 15. The system clock is 25 ns (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*
0x0100-0x0103	/PCS1	J2 pin 13=P17	U5 PPI pin 7
0x0108	/RST1	U11 pin 6	/RST1 for DAC U11
0x0110-0x0113	/DA	U11 pin 23	DAC7625 select
0x0118-0x011f	/AD	U12 pin 31	AD7852
0x0120-0x0123	T0-3	U4 pin 13-16	U14 ADC, U15 DAC
0x0200-0x02ff	/PCS2	J2 pin 22=CTS1	USER
0x0300-0x03ff	/PCS3	J2 pin 31=RTS1	USER
0x0400-0x04ff	/PCS4		Reserved
0x0500-0x05ff	/PCS5	J2 pin 15=P3	UART, SCC2691
0x0600-0x06ff	/PCS6	J2 pin 24=P2	RTC 72423

*PCS0 may be used for other TERN peripheral boards.

To illustrate how to interface the A-Engine86 with external I/O boards, a simple decoding circuit for interfacing to an 82C55 parallel I/O chip is shown in Figure 3.2.

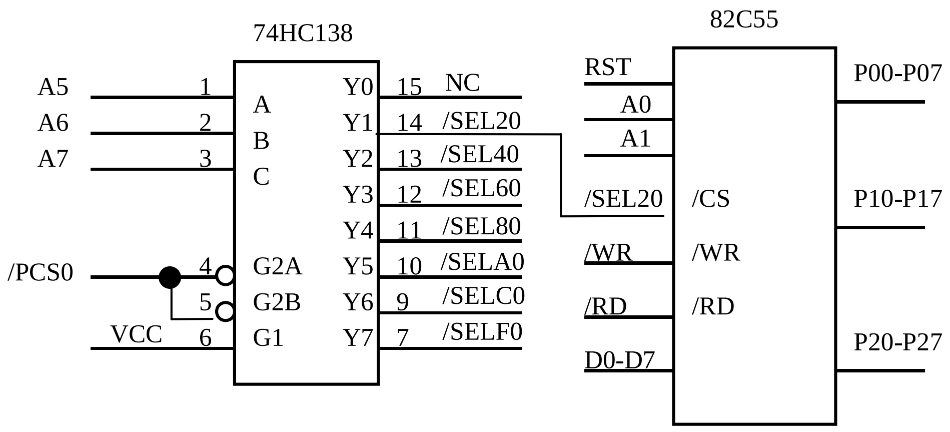


Figure 3.2 Interface the A-Engine86 to external I/O devices

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.

3.4.2 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.

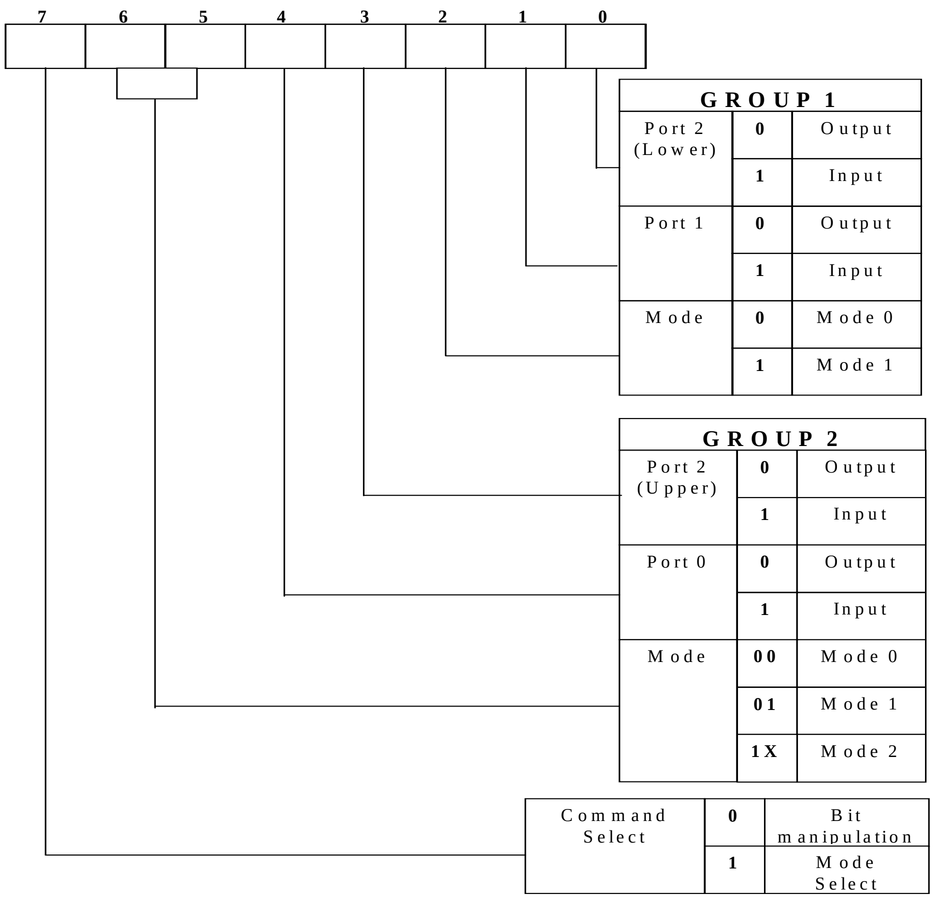


Figure 3.3 Mode Select Command Word

The A-Engine86 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 = 0x0103; Port 0 = 0x0100; Port 1 = 0x0101; and Port 2 = 0x0102.

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

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

To set all ports to input mode:

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

You can read the ports with:

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

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

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

See section 3.6.2 for PPI connection headers.

3.4.3 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`**.

3.4.4 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. MPO is routed to J1 pin 3, and MPI is not connected.

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

RxD (J1 pin 5), TxD (J1 pin 7), MPO (J1 pin 3), and MPI are TTL level signals. You must provide RS-232 or RS-485 drivers off of the A-Engine86 board if you choose to use either RS-232 or RS-485. This can be achieved by using either the VE232, or a number of other TERN peripheral controllers that are driven by the A-Engine86 and offer RS232/RS485 drivers.

The VE232 provides an RS-485 driver. You may connect the RxD and TxD signals on J1 of the A-Engine86 to H3 pins 3 and 2 on the VE232, and the 485+ and 485- signals at the screw terminal of the VE232 (J1) to join a multi-drop RS-485 twisted-pair network.

The RS-485 driver on the VE232 was designed to use J2 pin 24 to control the half-duplex RS-485 driver direction for the V25-Engine. However, the A-Engine86 J2 pin 24 (signal P2) is the RTC chip select signal. P2 should not be used as RS-485 driver direction control. You can use MPO of the SCC2691 (J1 pin 3) as RS-485 driver direction control and connect the AE86 J1 pin 3 to the VE232's H1 pin, which maps to AE86 J2 pin 24. You then should cut off pin 24 of the J2 on the A-Engine86 connection to the VE232. Please refer to Appendix B for the layout of the VE232 and to Chapter 1 for further information.

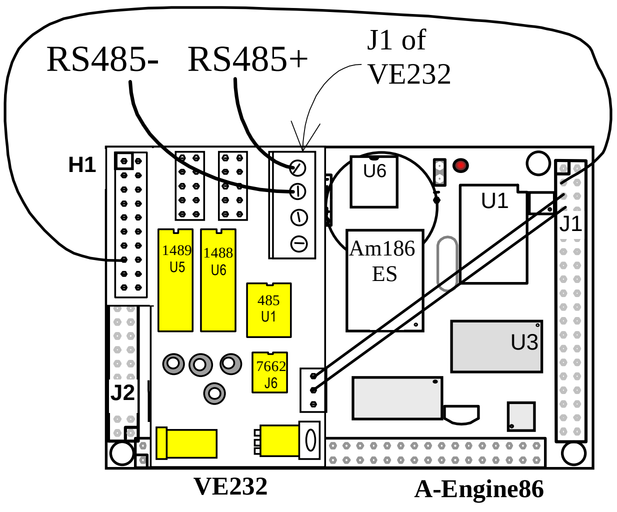


Figure 3.4 Settings and connections for networking with SCC2691 using the VE232

3.5 Other Devices

A number of other devices are also available on the A-Engine86. Some of these are optional, and might not be installed on the particular controller you are using. For a discussion regarding the software interface for these components, please see the Software chapter.

3.5.1 On-board Supervisor with Watchdog Timer

The MAX691/LTC691 (U6) is a supervisor chip. With it installed, the A-Engine86 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 A-Engine86. 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 A-Engine86 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()**.

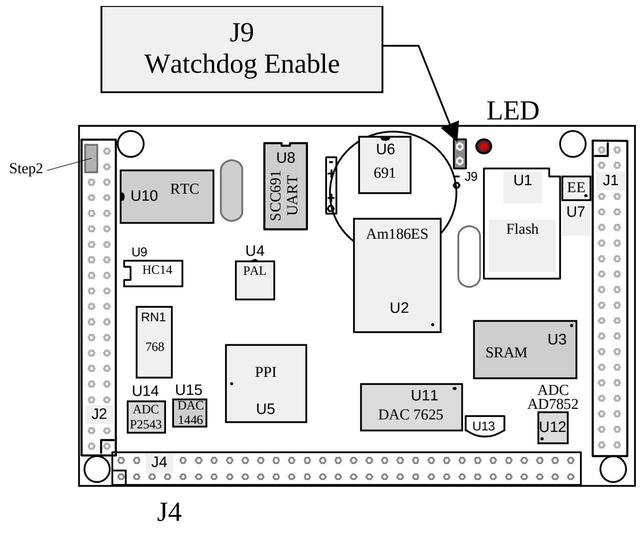


Figure 3.5 Location of watchdog timer enable jumper

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 directly shorted to VCC by default. In order to use PFI externally, cut the trace and bring the PFI signal out. 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.

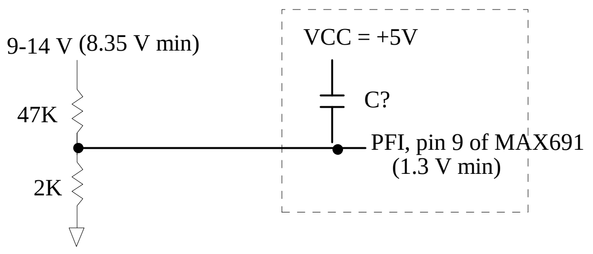


Figure 3.6 Using the supervisor chip for power failure detection

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.

3.5.2 EEPROM

A serial EEPROM of 128 bytes (24C01), 512 bytes (24C04), or 2K bytes (24C16) can be installed in U7. The A-Engine 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 and the 12-bit serial ADC (U14) share the same data input signal line, P11. The 12-bit ADC uses T2 (PAL U4) line as chip select. If the T2 line is low, the ADC will be enabled and holds the P11 data line, prohibiting EEPROM operation. The ae_init(); function sets T2 high. The ae_ad12(); function brings T2 low only when it needs to. 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 E of this manual.

3.5.3 P2543, 12-bit serial interface ADC

The P2543 is a 12-bit, switched-capacitor, successive-approximation, 11 channels, serial interface, analog-to-digital converter. Three I/O lines from PAL are used to handle the ADC, with /CS=T2; CLK=T0; and DIN=T1.

The ADC digital data output communicates with a host through a serial tri-state output (DOUT=P11). If T2=/CS is low, the P2543 will have output on P11. If T2=/CS is high, the P2543 is disabled and P11 is free. T2 and P11 are pulled high on board by default. The P2543 has an on-chip 14-channel multiplexer that can select any one of 11 inputs or any one of three internal self-test voltages. The sample-and-hold function is automatic. At the end of conversion, the end-of-conversion (EOC) output is not connected, although it goes high to indicate that conversion is complete.

P2543 features differential high-impedance inputs that facilitate ratiometric conversion, scaling, and isolation of analog circuitry from logic and supply noise. A switched-capacitor design allows low-error conversion over the full operating temperature range. The analog input signal source impedance should be less than 50Ω and capable of slewing the analog input voltage into a 60 pf capacitor.

The reference REF+ is connected to VQ (+5V). The VQ is a filtered VCC via a C-R-C filter, in order to reduce digital noise. The CLK signal to the ADC is toggled through an I/O pin, and serial access allows a conversion rate of up to approximately 10 KHz.

In order to operate the P2543, five I/O lines are used, as listed below:

/CS	Chip select = T2, high to low transition enables DOUT, DIN and CLK. Low to high transition disables DOUT, DIN and CLK.
DIN	T1, serial data input
DOUT	P11 of Am188ES, 3-state serial data output.
EOC	Not Connected, End of Conversion, high indicates conversion complete and data is ready
CLK	I/O clock = T0
REF+	Upper reference voltage (default VQ)
REF-	Lower reference voltage (default GND)
VCC	Power supply, +5 V input
GND	Ground

The analog inputs AN0 to AN10 are available at J4. You can use the C function ae86_ad12(char channel); to read the 12-bit ADC. See also the sample program ***ae86_ad.c*** in the **`c:\tern\186\samples\ae86`** directory.

3.5.4 Dual 12-bit DAC (LTC1446)

The LTC1446 is a dual 12-bit digital-to-analog converter (DAC) in an SO-8 package. It is complete with a rail-to-rail voltage output amplifier, an internal reference and a 3-wire serial interface. The LTC1446 outputs a full-scale of 4.096V, making 1 LSB equal to 1 mV. The buffered outputs can source or sink 5 mA. The outputs swing to within a few millivolts of supply rail when unloaded. They have an equivalent output resistance of 40 Ω when driving a load to the rails. The buffer amplifiers can drive 1000 pf without going into oscillation.

The DAC is installed in U15 on the AE86, and the outputs are routed to J4 pin 16 for DAC channel A, and J4 pin 18 for DAC channel B.

The DAC uses T0 as CLK, T1 as DI, and T3 as LD/CS. Please refer to the LT1446 technical data sheets from Linear Technology (1-408-432-1900) for further information. Use C function ae86_da(dat1, dat2); to drive the DAC. See also the sample program ***ae86_da.c*** in the **`c:\tern\186\samples\ae86`** directory.

3.5.5 AD7852, 300KHz 12-bit ADC

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 **ae86_ad.c** can be found in the **`c:\tern\186\samples\ae86`** directory. The 7852 A/D pins are located on pins AD0toAD7 on J4.

3.5.6 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 AE86 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 **ae86_da.c** may be found in the **`c:\tern\186\samples\ae86`** directory.

3.6 Headers and Connectors

3.6.1 Expansion Headers J1 and J2

There are two 20x2 0.1 spacing headers for A-Engine86 expansion. Most signals are directly routed to the Am186ES processor. These signals are 5V only, and any out-of-range voltages will most likely damage the board.

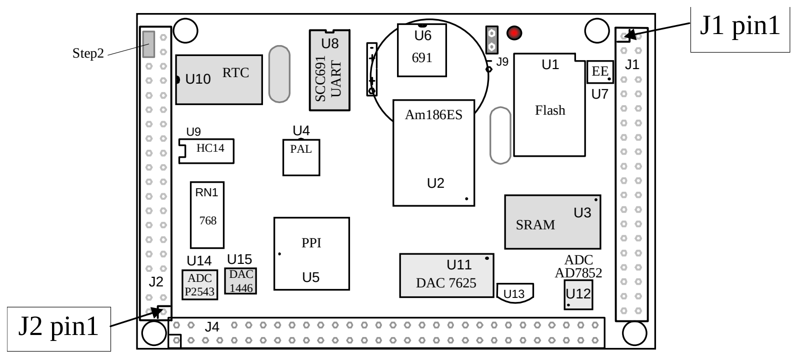


Figure 3.7 Pin 1 locations for J2 and J1

J2 Signal			
GND	40	39	VCC
P4	38	37	P14
/CTS0	36	35	P6
TxD0	34	33	/INT4
RxD0	32	31	/RTS1
P5	30	29	P1
TxD1	28	27	/RTS0
RxD1	26	25	ALE
P2	24	23	P15
/CTS1	22	21	/INT3
P0	20	19	/INT2
P25	18	17	P24
	16	15	P3
	14	13	P17
P10	12	11	P13
A19	10	9	
/INT0	8	7	/NMI
/INT1	6	5	P12
P26	4	3	P29
GND	2	1	GND

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

Table 3.3 Signals for J2 and J1, 20x2 expansion ports

Signal definitions for J2:

VCC	+5V power supply, < 200 mA
GND	ground
Pxx	Am186ES PIO pins
A19	Am186ES pin 19
TxD0	Am186ES pin 2, transmit data of serial channel 0
RxD0	Am186ES pin 1, receive data of serial channel 0
TxD1	Am186ES pin 98, transmit data of serial channel 1
RxD1	Am186ES pin 99, receive data of serial channel 1
/CTS0	Am186ES pin 100, Clear-to-Send signal for SER0
/CTS1	Am186ES pin 63, Clear-to-Send signal for SER1
/RTS0	Am186ES pin 3, Request-to-Send signal for SER0
/RTS1	Am186ES pin 62, Request-to-Send signal for SER1
/INT0-4	Schmitt-trigger external interrupt inputs
/NMI	Schmitt-trigger external NMI inputs

Signal definitions for J1:

VCC	+5V power supply
GND	ground
CLK	Am186ES pin 16, system clock, 40 MHz (25 ns)
RxD	data receive of UART SCC2691, U8
TxD	data transmit of UART SCC2691, U8
MPO	Multi-Purpose Output of SCC2691, U8
VOFF	real-time clock output of RTC72423 U10, open collector
D0-D15	Am186ES 16-bit external data lines
A0-A7	Am186ES address lines
/RST	reset signal, active low
RST	reset signal, active high
P16	/PCS0, Am186ES pin 66
/BHE	Am186ES pin 4
/WR	Am186ES pin 5
/RD	Am186ES pin 6

3.6.2 J4 Connector for PPI, ADC, DAC

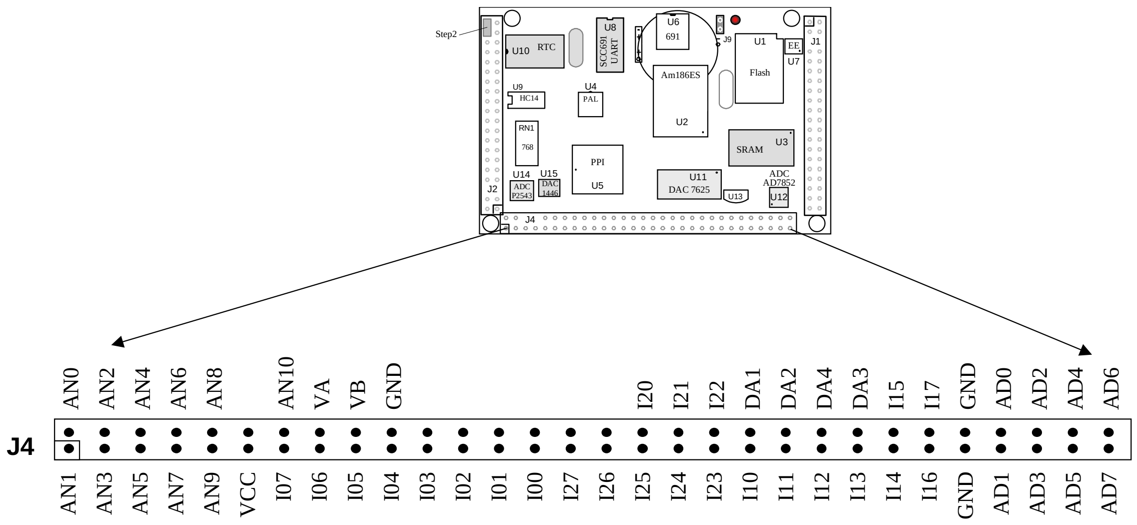


Figure 3.8 J4 connector

The pin layout for the J4 30x2 pin header on the A-Engine86 is as follows:

J4 Signal			
AN1	1	2	AN0
AN3	3	4	AN2
AN5	5	6	AN4
AN7	7	8	AN6
AN9	9	10	AN8
VCC	11	12	
I07	13	14	AN10
I06	15	16	VA
I05	17	18	VB
I04	19	20	GND
I03	21	22	GND
I02	23	24	GND
I01	25	26	GND
I00	27	28	GND
I27	29	30	GND
I26	31	32	GND
I25	33	34	I20
I24	35	36	I21
I23	37	38	I22
I10	39	40	DA1
I11	41	42	DA2
I12	43	44	DA4
I13	45	46	DA3
I14	47	48	I15
I16	49	50	I17
GND	51	52	GND
AD1	53	54	AD0
AD3	55	56	AD2
AD5	57	58	AD4
AD7	59	60	AD6

Table 3.4 Signals for J4, 30x2 header

3.6.3 Jumpers

The following table lists the jumpers and connectors on the A-Engine86.

Name	**Size**	**Function**	**Possible Configuration**
J1	20x2	main expansion port	MemCard1 interface
J2	20x2	main expansion port	Pins 38=40: Step2 jumper (must be installed for Step1 and Step2)
J4	30x2	ADC, DAC, PPI	
J9	2x1	Watchdog timer	Enabled if Jumper is on Disabled if jumper is off

3.6.4 20x4 line LCD Interface

The A-Engine86 CAN*NOT* directly interface to LCD modules with address/data bus. You may use 82C55 I/O pins to drive LCD modules. See the sample program **ae_lcd.c** in the TERN EV/DV diskette.

Pin connections are made as follows:

Use PPI on A-Engine86 to drive 20x4 line LCD.

J4 I07-I00 = D7-D0

I27=LCD, I26=R/W, I25=A0, I24=VLC, I23=VCC, I10=GND

LCD configuration:

Pin 3 = VLC disconnected to I24

VLC via 10K to pin2=VCC and via 200 ohm to pin1=GND

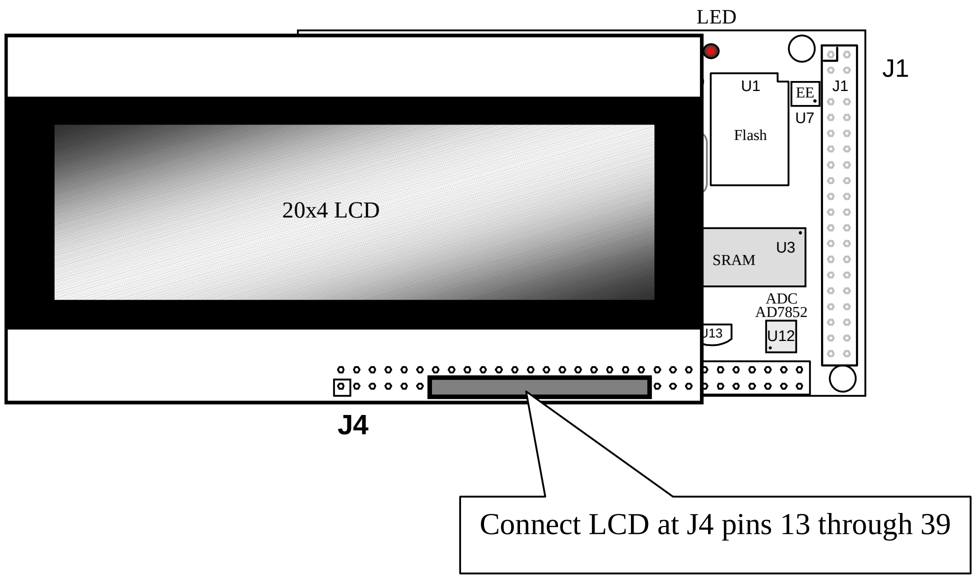


Figure 3.9 Installation of a 20x4 LCD on the A-Engine86

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

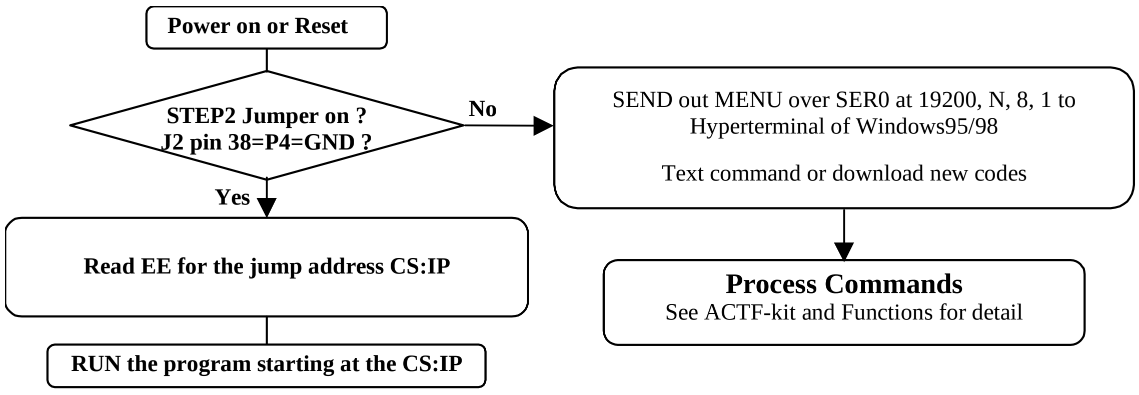
This function can be used to retrieve data from components in I/O space. You will find that most hardware options added to TERN controllers are mapped into I/O space, since memory space is valuable and is reserved for uses related to the code and data. Using I/O mappings, the address is output over the 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 AE86 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 (TDREM115.HEX) and 38,400 (TDREM384.HEX) are available. A loader file L_TDREM.HEX and both debugger files TDREM115.HEX and TDREM384.HEX, are included in the EV/DV disk under the **`c:\tern\186\samples\AE86`** directory.

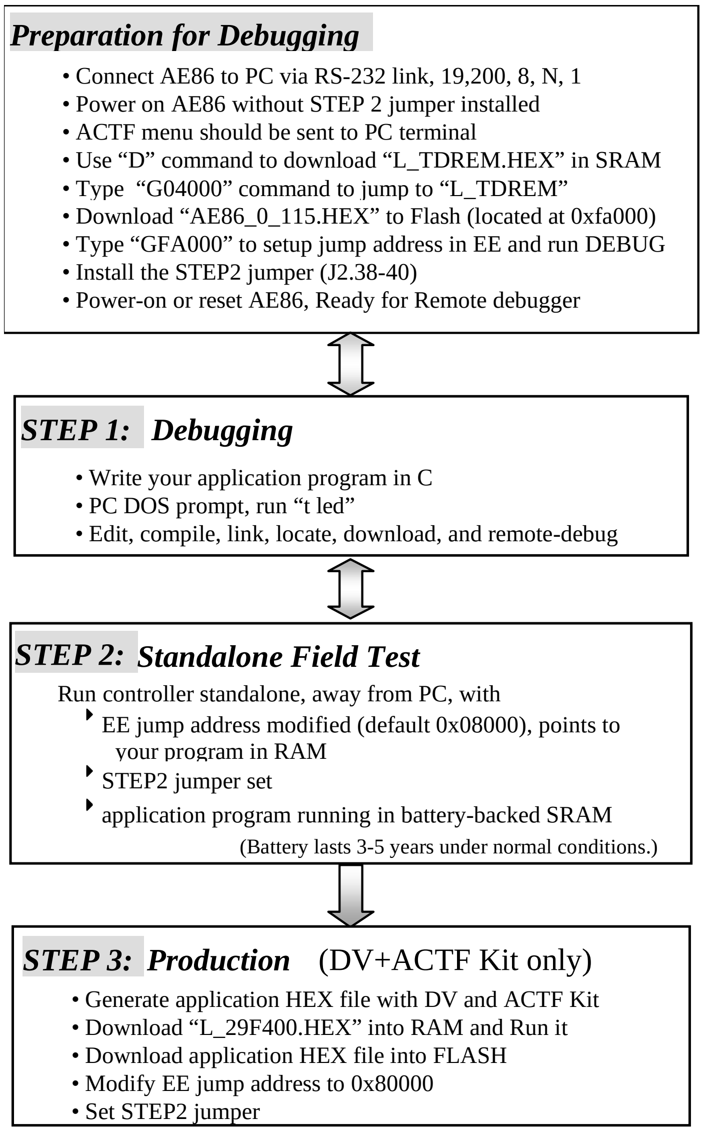
A functional diagram of the ACTF (embedded in the AE86) is shown below:


Power on or Reset
STEP2 Jumper on ?
J2 pin 38=P4=GND ?
No
Yes
SEND out MENU over SER0 at 19200, N, 8, 1 to Hyperterminal of Windows95/98
Text command or download new codes
Read EE for the jump address CS:IP
Process Commands
See ACTF-kit and Functions for detail
RUN the program starting at the CS:IP

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\ae86`** directories.

4.1.1 Steps for AE86-based product development



Preparation for Debugging

- Connect AE86 to PC via RS-232 link, 19,200, 8, N, 1
- Power on AE86 without STEP 2 jumper installed
- ACTF menu should be sent to PC terminal
- Use “D” command to download “L_TDREM.HEX” in SRAM
- Type “G04000” command to jump to “L_TDREM”
- Download “AE86_0_115.HEX” to Flash (located at 0xfa000)
- Type “GFA000” to setup jump address in EE and run DEBUG
- Install the STEP2 jumper (J2.38-40)
- Power-on or reset AE86, Ready for Remote debugger

STEP 1: Debugging

- Write your application program in C
- PC DOS prompt, run “t led”
- Edit, compile, link, locate, download, and remote-debug

STEP 2: Standalone Field Test

Run controller standalone, away from PC, with

- EE jump address modified (default 0x08000), points to your program in RAM
- STEP2 jumper set
- application program running in battery-backed SRAM

(Battery lasts 3-5 years under normal conditions.)

STEP 3: Production (DV+ACTF Kit only)

- Generate application HEX file with DV and ACTF Kit
- Download “L_29F400.HEX” into RAM and Run it
- Download application HEX file into FLASH
- Modify EE jump address to 0x80000
- Set STEP2 jumper

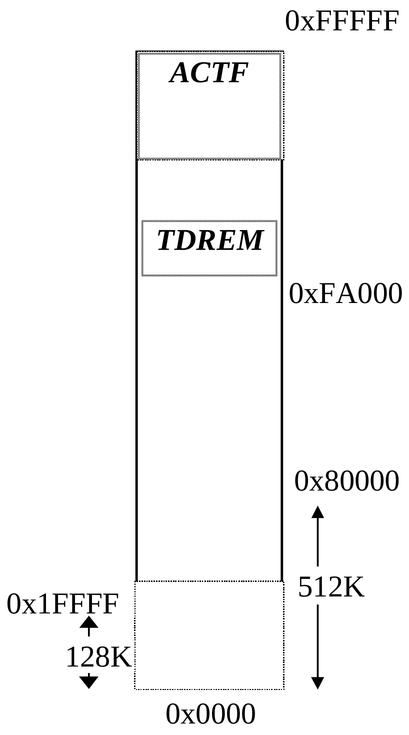
There is no ROM socket on the AE86. The user’s application program must reside in SRAM for debugging in STEP1, reside in battery-backed SRAM for the standalone field test in STEP2, and finally be programmed into Flash for a complete product.

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.

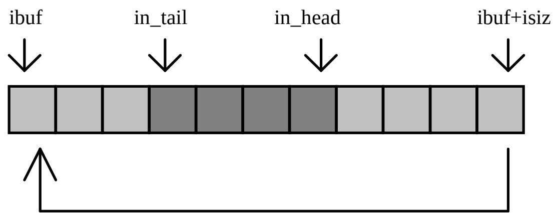


Figure 4.1 Circular ring input buffer

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
4	600
5	1200
6	2400
7	4800

Function Argument	Baud Rate
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: **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.

4.6 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

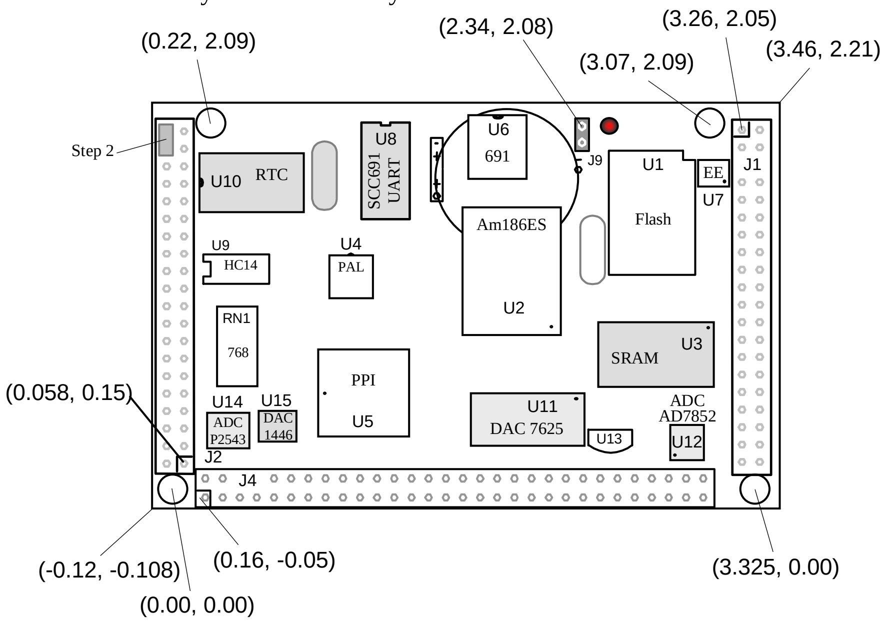
This function is used to write the passed in **dat** to the specified **addr**. The return value is 0 in success.

ee_rd
Arguments: int addr
Return value: int data

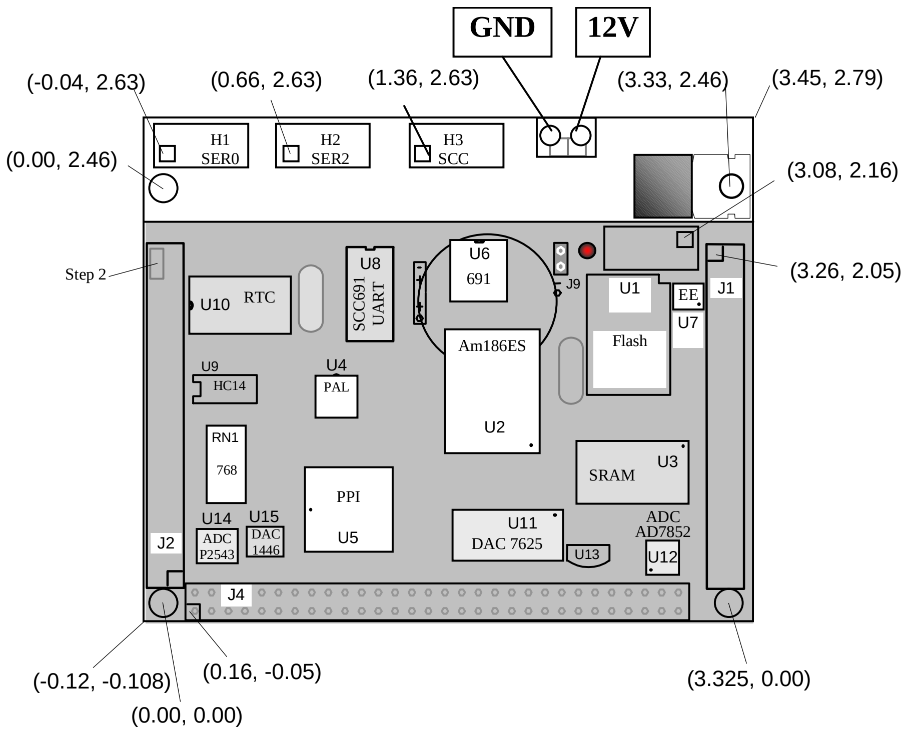
This function returns one byte of data from the specified address.

Appendix A: A-Engine86 Layout

The **A-Engine86** measures 3.6 by 2.3 inches. Its layout is shown below. All dimensions are in inches.



The **A-Engine86-P** measures 3.6 x 2.9 x 0.3 inches. Its layout is shown below. The lower portion of the board (shaded) is identical to the **A-Engine**, refer to the layout shown above.



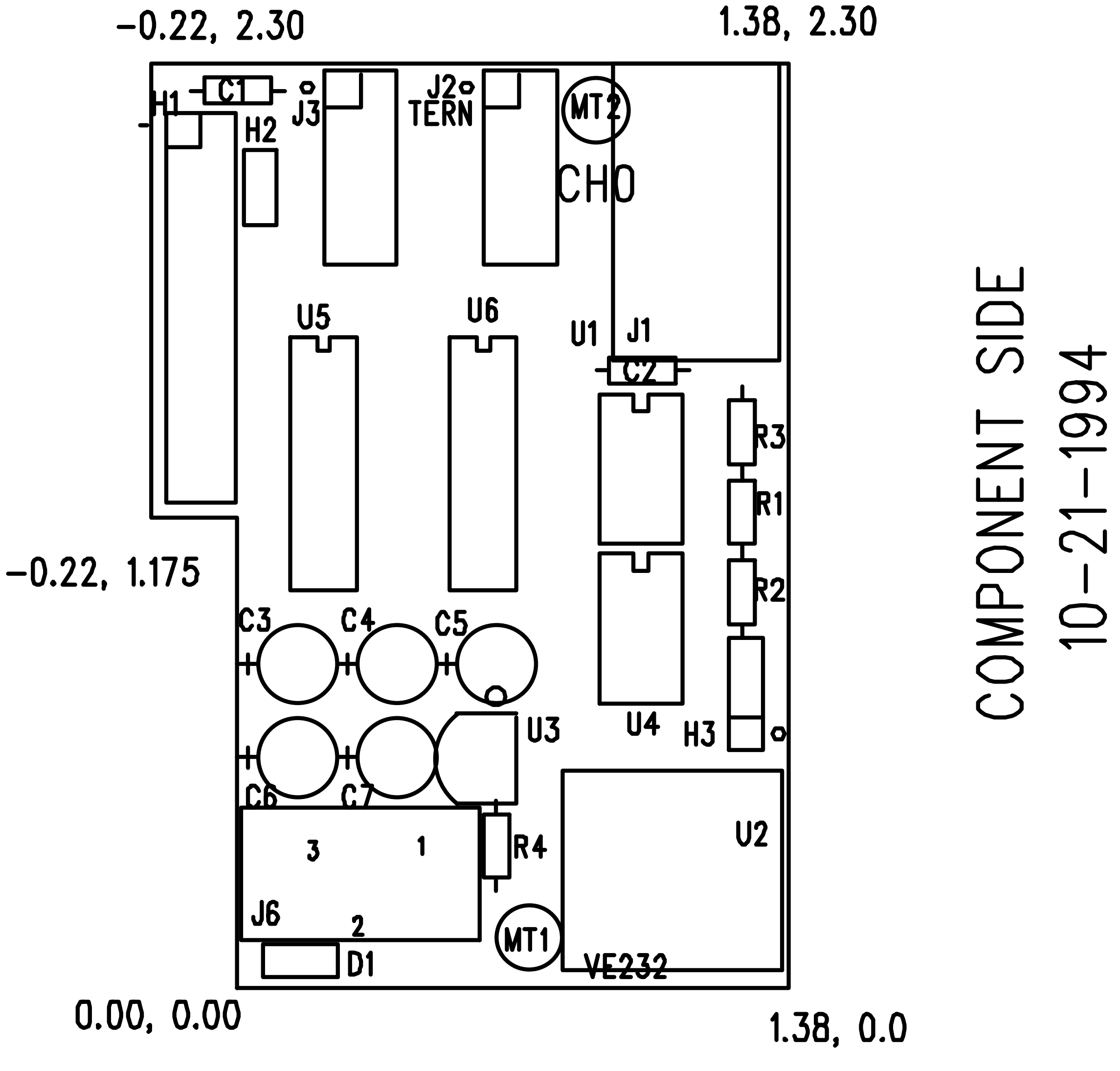
The ***A-Engine86-P*** (**AE86-P**) is a new version of the ***A-Engine86*** with an on-board regulated 5-volt power and either RS232 or 485 drivers, eliminating the need for the VE232. Measuring 3.6 x 2.9 x 0.3 inches, the ***AE86-P*** is ideal for applications in which space is very limited, where the VE232's extra height could not be accommodated.

The ***A-Engine*** can be installed on TERN controllers, such as the ***PowerDrive***, ***SensorWatch***, ***LittleDrive***, or ***MotionC.*** TERN offers custom hardware and software designs, based on the ***A-Engine86,*** or other TERN controllers.

The ***AE86-P*** supports an optional LM2575 switching regulator (U19). The switching regulator consumes less power and generates less heat than the standard linear regulator. Furthermore, the switching regulator can be disabled to power-off the ***AE86-P*** via the VOFF pin of the real-time clock. The battery-backed real-time clock can be programmed to power-on the A-Engine86-P (*see sample program poweroff.c*)

Appendix B: VE232 Pin Layout

All dimensions are in inches.



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

SR (Channel Status Register):

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

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

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

ACR (Auxiliary Control Register):

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

ISR (Interrupt Status Register):

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MPI Pin Change	MPI Pin Current State	Not Used	Counter Ready	Delta Break	RxRDY/ FFULL	TxEMT	TxRDY
0 = no 1 = yes	0 = low 1 = high		0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes

IMR (Interrupt Mask Register):

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MPI Change Interrupt	MPI Level Interrupt	Not Used	Counter Ready Interrupt	Delta Break Interrupt	RxRDY/ FFULL Interrupt	TxEMT Interrupt	TxRDY Interrupt
0 = off 1 = 0n	0 = off 1 = 0n		0 = off 1 = 0n	0 = off 1 = 0n	0 = off 1 = 0n	0 = off 1 = 0n	0 = off 1 = 0n

CTUR (Counter/Timer Upper Register):

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

CTLR (Counter/Timer Lower Register):

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

Appendix D: 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 E: Serial EEPROM Map

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

0x00	Node Address, for networking		
0x01	Board Type	00	VE
		10	CE
		01	BB
		02	PD
		03	SW
		04	TD
		05	MC
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		

Appendix F: Software Glossary

The following is a glossary of library functions for the A-Engine86.

void ae_init(void) ae.h

Initializes the AM188ES processor. The following is the source code for ***ae_init()***

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

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

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

Reference: led.c

void ae_reset(void) ae.h

Resets AM188 processor.

void delay_ms(int m) ae.h

Approximate microsecond delay. Does not use timer.

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

Reference: led.c

void led(int i) ae.h

Toggles P12 used for led.

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

Reference: led.c

void delay0(unsigned int t) ae.h

Approximate loop delay. Does not use timer.

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

Reference:

void pwr_save_en(int i) ae.h

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

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

Reference: ae_pwr.c

void clka_en(int i) ae.h

Enables signal CLK respectively for external peripheral use.

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

Reference:

void hitwd(void) ae.h

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

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

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

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

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

Reference: ae_pio.c

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

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

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

Reference: ae_pio.c

unsigned int pio_rd(char port) ae.h

Reads a 16 bit PIO port.

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

Reference: ae_pio.c

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

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

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

Reference: ae_ppi.c

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

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

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

Reference: ae_ppi.c

int inport(int portid) dos.h

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

```
Var:  portid – I/O address
```

Reference: ae_ppi.c

int inportb(int portid) dos.h

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

```
Var:  portid – I/O address
```

Reference: ae_ppi.c

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

Writes to the serial EEPROM.

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

Reference: ae_ee.c

int ee_rd(int addr) aeee.h

Reads from the serial EEPROM. Returns 8-bit data

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

Reference: ae_ee.c

int ae_ad12(unsigned char c) ae.h

Reads from the 11-channel 12-bit ADC. Returns 12 bit AD data of the previous channel.
In order to operate ADC, I20,I21,I22 must be output and P11 must be input.
P11 is shared by RTC, EE. It must left high at power-on/reset.
Unipolar:
Vref- = 0x000
Vref+ = 0xfff
Use 1 wait state for Memory and I/O without RDY, < 300 us execution time
Use 0 wait state for Memory and I/O with VEP010, < 270 us execution time

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

Reference: ae_ad12.c

void io_wait(char wait) ae.h

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

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

Reference:

void rtc_init(unsigned char * time) ae.h

Sets real time clock date, year and time.

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

Reference: rtc_init.c

int rtc_rd(TIM *r) ae.h

Reads from the real time clock.

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

Reference: rtc.c

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

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

Timer 0, 1, 2 initialization.

```
Var:  tm – Timer mode. See pg. 8-3 and 8-5 of the AMD CPU Manual
      ta – Count time a (1/4 clock speed).
      tb – Count time b for timer 0 and 1 only (1/4 clock).
            Time a and b establish timer duty cycle (PWM). See
            hardware chapter.
      t#_isr – pointer to timer interrupt routine.
```
Reference: timer.c, timer1.c, timer02.c, timer2.c, timer0.c timer12.c

---

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

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

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

---

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

Serial port 0, 1 initialization.

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

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

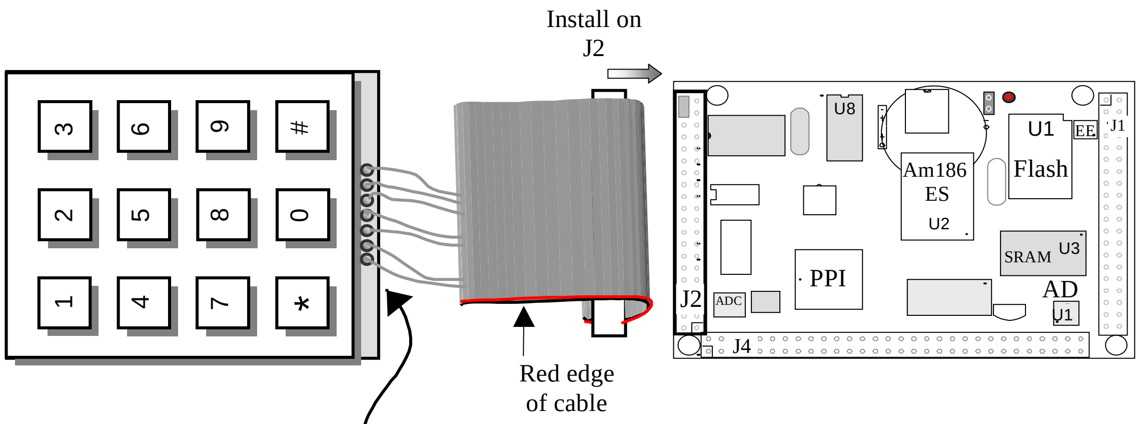
Retrieves a fixed length character string from the input buffer. If the buffer contains less characters than the length requested, *str* will contain only the remaining characters from the buffer. Appends a '\0' character to the end of *str*. Returns the retrieved string length.

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

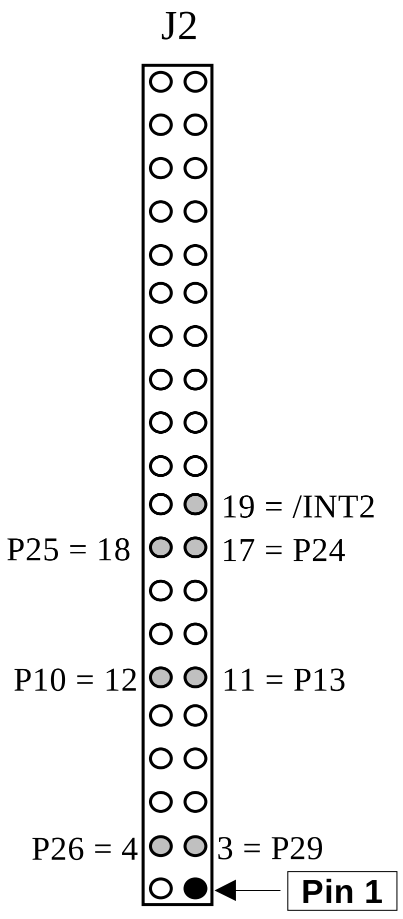
Reference: ser1.h, ser0.h for source code.

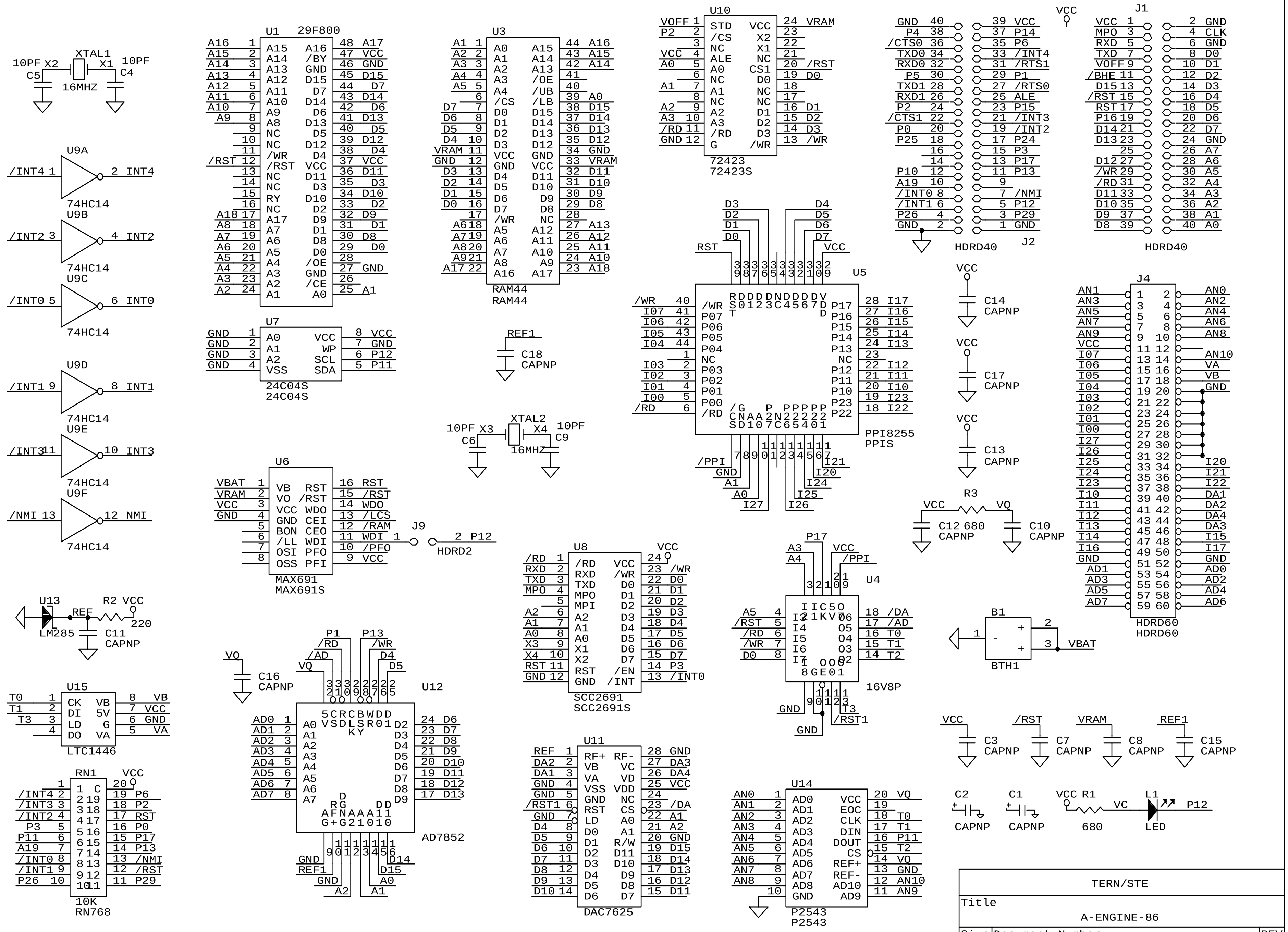
Appendix G: A-Engine86 Keypad Interface

The following diagram shows connections for interfacing a 3x4 keypad to an A-Engine86. Please refer to the sample file **`ae_kpad.c`** in the **`samples\ae`** directory for more information.

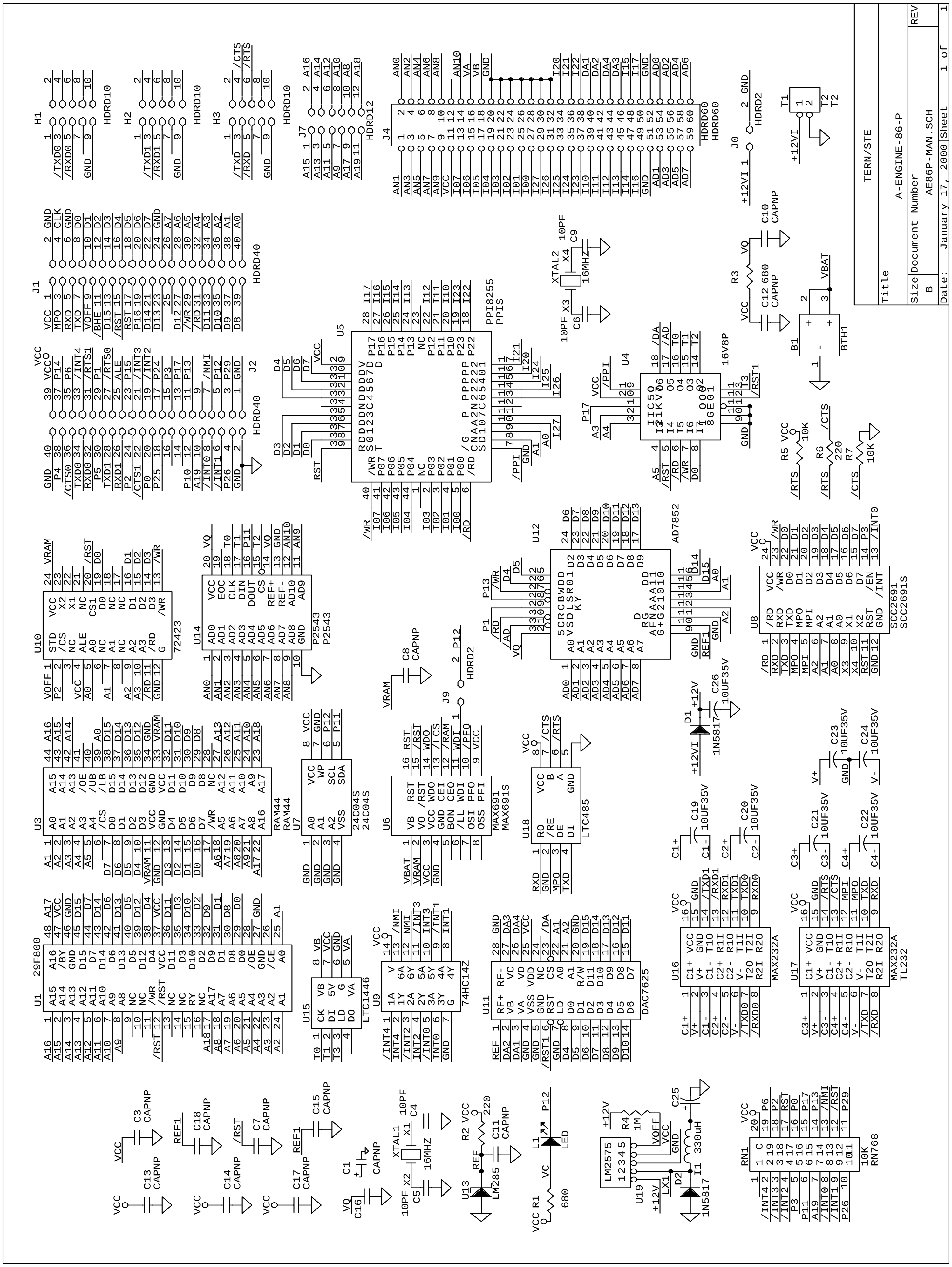


Keypad signals:	**J2: pin signal**
R2	19 = /INT2
R3	18 = P25
C3	17 = P24
R4	11 = P13
C1	12 = P10
R1	4 = P26
C2	3 = P29





TERN/STE		
Title A-ENGINE-86		
Size B	Document Number AE86-MAN.SCH	REV
Date: January 17, 2000	Sheet 1 of 1	



TERN/STE

Title A-ENGINE-86-P

Size B | Document Number AE86P-MAN.SCH | REV

Date: January 17, 2000 | Sheet 1 of 1