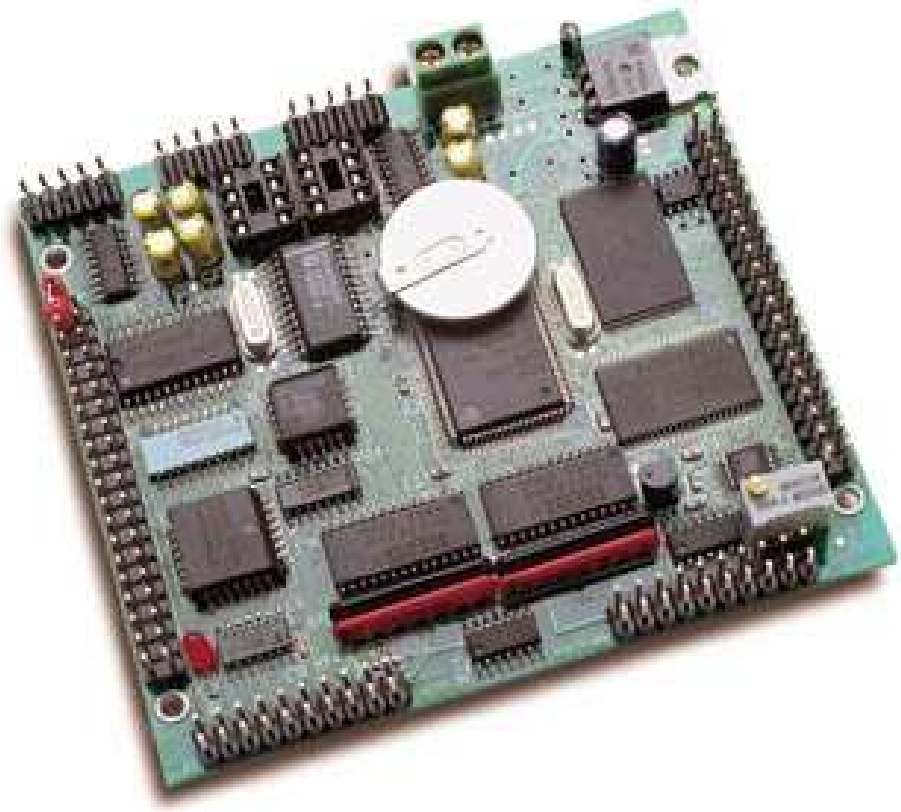
A-Engine86-D™

40 MHz AM186ES, with 16-bit Flash and SRAM,
8 300 KHz 12-bit ADCs, 8 12-bit DACs, 6 16-bit counters, 3 UARTs



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 3.00

June 25, 2010

© 2001-2010 TERN INC.
1950 5th Street, Davis, CA 95616, USA
Tel: 530-758-0180 Fax: 530-758-0181
Email: sales@tern.com ***http://www.tern.com***

Important Notice

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

Chapter 1: Introduction

1.1 Functional Description

Measuring 3.6 x 3.2 x 0.3 inches, the ***A-Engine*86-D** (***AE86D***) 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 ***AE86D*** 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-D (AE86D)** is a new version of the ***A-Engine*86** with 8 channels of high speed parallel 12-bit DACs, additional 3 channels of 16-bit external counters, and on-board regulated 5-volt power and either RS232 or 485 drivers. The ***AE86D*** is based on the 40 MHz Am186ES processor, which has both internal and external 16-bit data path. The ***AE86D*** provides a true 16-bit data bus at J1 20x2 header. The ***AE86D*** 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.

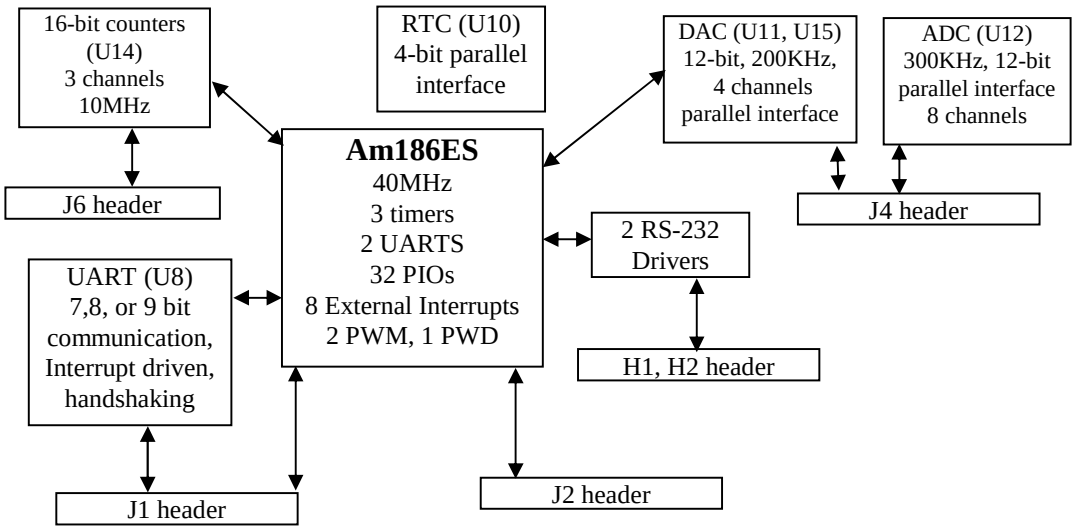


Figure 1.1 Functional block diagram of the A-Engine86-D

In addition to offering a 16-bit external data bus, the ***AE86D*** 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-P (TERN's Development 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, and second. A 512-byte serial EEPROM is included to allow storage of constants, calibration codefficients, and other important values.

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.

The Am186ES provides three 16-bit programmable timers/counters. Two timers can be used to generate non-repetitive or variable-duty-cycle waveforms as PWM outputs. Timer1 can count or time external events, at a rate of up to 10 MHz. In addition, Pulse Width Demodulation (PWD), a distinctive feature, is also available on-board. It 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.

An additional three 16-bit programmable high-performance counters (71054, NEC), each with its own clock input, gate input, and output, can be clocked up to 10 MHz are standard.

The ***AE86D*** provides 32 user-programmable, multifunctional I/O pins from the CPU. Schmitt-trigger inverters are provided for external interrupt inputs and external counter 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. Two 4-channel, high-speed (200KHz) parallel DACs (DA7625) can be installed. The DAC outputs are buffered by on-board operational amplifiers to provide a total of 8 analog outputs (default to 0-5V), with hardware adjustable gain and offset.

The ***A-Engine*86-D** works with other TERN expansion boards, such as ***MMB, P300, P100***, ***MC2140, MCP2540, or FC-0.***

The **AE86D** 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 **AE86D** via the VOFF pin of the real-time clock. The battery-backed real-time clock can be programmed to power-on the **A-Enginre86-D** (see sample program poweroff.c, in the tern\186\samples\ae directory)

On the **AE86D**, an optional RS485 (two wires, half-duplex) or RS422 (4 wires, full-duplex) driver can be installed for SER1. All chips are surface mounted for highest reliability. All options can be installed with no conflict on the same board. An optional switching regulator can be installed to reduce power consumption and heat.

1.2 Features

Features:

• 3.6 x 3.2 x 0.3”
• 190/30 mA normal/power-save
• 40 MHz, 16-bit CPU (Am186ES), program in C/C++
• 256 KW 16-bit Flash
• 64KW or optional 256KW 16-bit SRAM*
• 16-bit external data bus, 32 PIOs
• Real-time clock (RTC72423) with back-up battery*
• 3 serial ports RS232/485*,
• 512-byte non-volitile EEPROM
• 2 PWM channels and 1 PWD channel
• 3 16-bit Timer/counters from Am186ES
• 8 ch. 300 KHz 12-bit ADC (AD7852), 0-5V analog input*
• Optional switching regulator supports 9-30V power input
• 3 ch. 16-bit counters (82C54)*
• 8 ch. 12-bit DAC (DAC7625), 5µs settling time, 0-5V output*
*optional

1.3 Physical Description

The physical layout of the **A-Engine86-D** is shown in Figure 1.2. All dimensions are in inches.

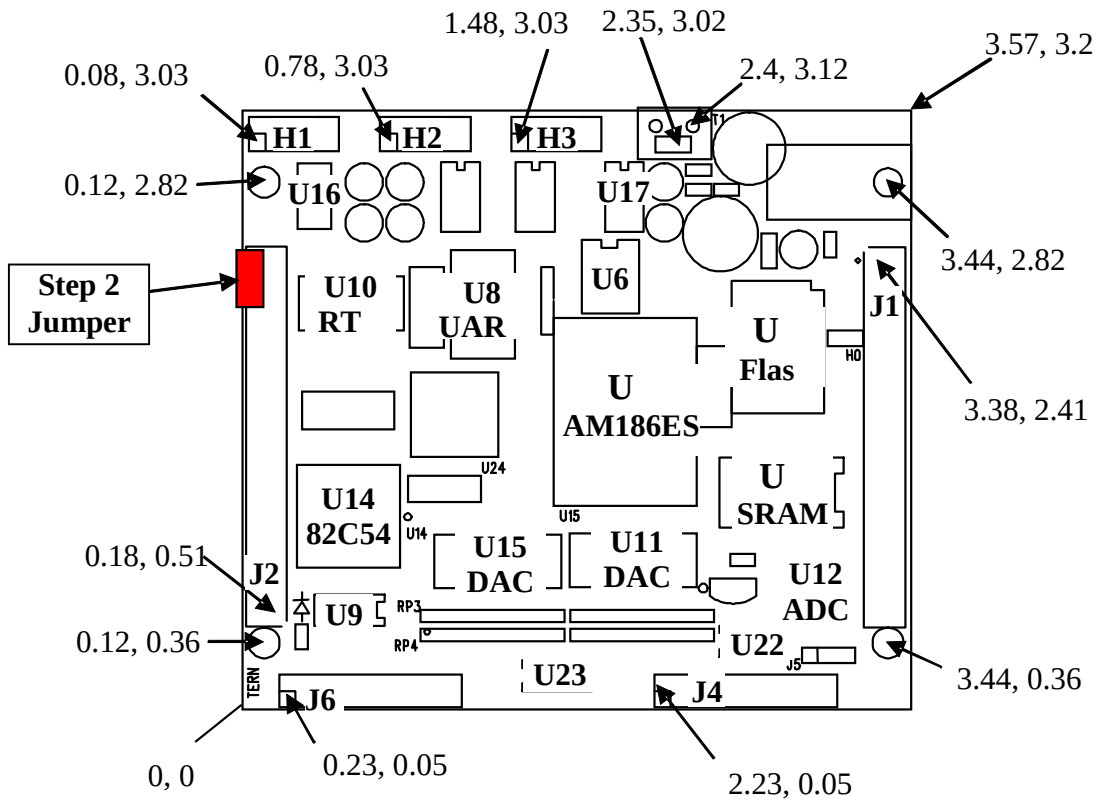
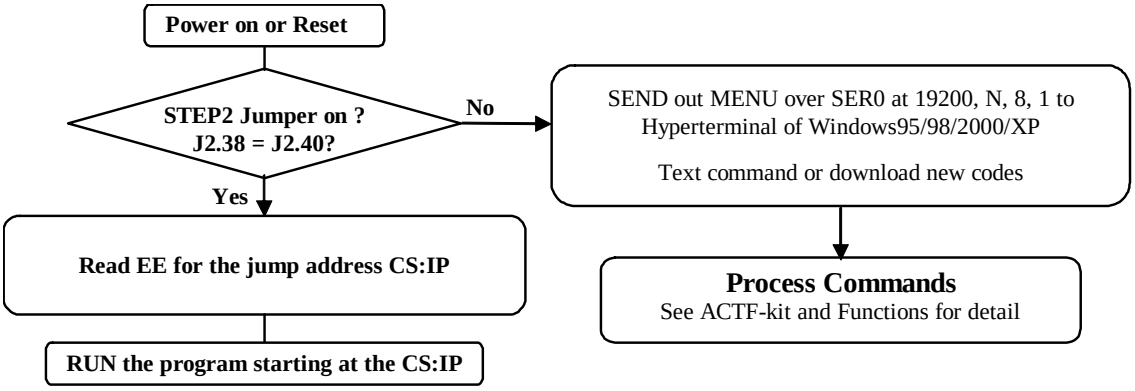


Figure 1.2 Physical layout of the A-Engine86-D

1.4 A-Engine86-D Programming Overview

At the factory, an ACTF utility is loaded into the upper sector of the on-board flash. This ACTF utility is protected and executes at every power up. Upon power up, the ACTF utility will perform the process as described by the flow chart below. The remainder of this section will be divided into parts: Prepare for Debug Mode (STEP 1), Debug Mode (STEP 1), Standalone Mode (STEP 2), and finally, Production (STEP 3). For your convenience, the preparation for debug mode is done at the factory, meaning you can begin at STEP 1: Debug Mode.



1.4.1 1.4.1 Prepare for Debug Mode (STEP 1):

To run the **AE86D** in Step 1, the debug mode, a debug kernel must be loaded into the on-board flash. This is done at the factory for your convenience. This debug kernel must be running to communicate with the Paradigm C/C++ programming environment. It resides in the on-board flash at address 0xFA000. To run the debug kernel and prepare for debug mode, do the following:

1. Link the **A-Engine86-D** to your PC and prepare a hyper terminal session. Configure the terminal to 19,200 Baud, 8 bits, No parity, and 1 stop. Connect to SER0 (H1) of the **A-Engine86-D**.
2. Power on the **A-Engine86-D without** the STEP 2 installed. The STEP 2 jumper is a red jumper installed on the J2 header pins 38 and 40.
3. At power up, you should see the ACTF Utility menu at your hyper terminal:

   ACTF/ACTR Copyright(c) 1996 STE CA USA. All rights reserved.

   >C C FUNCTIONS

   >D Download an Intel Extend Hex file into SRAM

   >G Goto and Run

   >H HELP

   >M MENU

   >U Upload a block of Binary data

   The “G” command allows you to jump to a location and immediately begin execution. It also sets the start-up jump address. Type “GFA000”, then <enter>. Your **A-Engine86-D** will jump to that location in the flash and begin to run the debug kernel. The on-board LED will blink twice, then stay on. This indicates the **A-Engine86-D** is correctly running the debug kernel.
4. Now install the STEP 2 jumper (red jumper installed at J2 pins 38 and 40).
5. Now at start up, the ACTF Utility will check if the STEP 2 jumper is installed. If the STEP 2 jumper is installed, the CPU will fetch the start up jump address (which we set in instruction 4 to point to the debug kernel, 0xFA000) and jump to that address for execution. Your **A-Engine86-D** is now ready to communicate with the Paradigm C/C++ for Debug Mode. If the STEP 2 jumper is not detected, the ACTF Utility will send out it’s start up menu, and you will be back to instruction 3.

When you jump to the debug kernel (by typing “GFA000”, then <enter> at the ACTF menu), if you do not see the on-board LED blink twice then stay on, the debug kernel has been erased. It must be loaded again to run STEP 1 and communicate with the Paradigm C/C++ software. See the section 1.5.1 for instructions on how to load the debug kernel.

1.4.2 1.4.2 STEP 1: Debug Mode

After completing the pervious section, your **A-Engine86-D** is ready to communicate with the Paradigm C/C++ Environment and debug source code. Use samples provided in the **c:\tern\186\samples\ae86d** and **c:\tern\186\samples\ae** directories to generate source code. Debug your code as needed. You can then go to STEP 2: Standalone Mode.

1.4.3 1.4.3 STEP 2: Standalone Mode

Now that you have debugged your source code, you are ready to test it in standalone mode. To run standalone, do the following:

1. Remove the STEP 2 jumper. Prepare a hyper terminal session as desribed by section 1.4.1.
2. At power-on, you will see ACTF menu. (The key is that the STEP 2 jumper is not installed, so the CPU does not fetch the jump address).
3. ***You now want to jump to you program. In STEP 1, the Paradigm C/C++ environment downloads your program into the SRAM, starting at address 0x08000. We now want to use the same “G” command as before, but jump to your program, not the debug kernel. Type “G08000”, then <enter>. The CPU will then jump to your program in the SRAM for immediate execution. It will also set the start-up jump address to 0x08000.
4. ***Re-install the STEP 2 jumper (J2 pins 38and 40). Now at every power up, the ACTF utility will see the STEP 2 jumper and fetch the jump address, which now points to your program in the SRAM. Your program will now execute in standalone mode at every power up.
5. When finished with STEP 2: Standalone Mode, you can go back to STEP 1:Debug Mode by repeating instructions 1 & 2 of this section. Then use “GFA000”, then <enter> to jump back to the debug kernel. The A-Engine86-D is now ready to communicate with the Paradigm C/C++ Environment.
6. This cycle between STEP 1 and STEP 2 can be done until your program is complete.

*** These instructions assume your controller has been equipped with the back-up battery. This battery will hold your code in the SRAM even without applied power. If your TERN controller does not have a back-up battery, you may NOT power-down your board after downloading your code to the SRAM. To run standalone you must set the jump address at the ACTF Utility, and assert /RST to cycle power, do not power down. Refer to the **AE86D** schematic at the end of this manual for additional details on the AE86D’s /RST lines. ***

1.4.4 1.4.4 STEP 3: Production

This step only applies to those users who have purchased the full Development version (DV-P) of the Paradigm C/C++ Environment.

1. When you have finished development of your program, your are ready to use your source code to generate an Intel Extend HEX file, which can then be burned into the on-board flash for a production version of the **A-Engine86-D**.
2. Inside Paradigm C/C++, change the config node of your target from "186.cfg" to "actf186.cfg" (Path=c:\tern\186\config\actf186.cfg). This is done by right-mouse clicking on the config node and selecting "Edit Node Attributes".
3. Open "actf186.cfg" for editing. Follow the instructions at the top of the config file. Save and close.
4. Right-mouse click on the "axe" node of your target and select "Target Expert". Within the Target Expert window, change **PDREMOTE/ROM** to **No Target/ROM**.
5. Now right-mouse click on the "axe" node and select "Build Node". You have now generated an Intel Extend Hex file. The name of the file will have the same name as your target, in the same working directory, but with the ".HEX" file extension.

   For example, if the name of my target is "My_Program.axe", then I will have created "My_Program.hex" in the same directory.
6. Go to Section 1.5.2 of this manual and follow the instruction for burning HEX files into the on-board flash.

1.5 Burning HEX Files into the On-board Flash

This section will cover two processes:

(1) Burning the debug kernel into the flash to prepare for debug mode.

(2) Burning you application HEX file into the flash to complete a production version.

1.5.1 1.5.1 Burning the debug kernel into the flash

1. Power on the **A-Engine86-D without** the STEP 2 jumper installed. See the ACTF menu at the hyper terminal.
2. Type "D", then <enter> to alert the ACTF utility that you are ready to send a file into the SRAM. You should see the following at your terminal:

```
ACTF/ACTR Copyright(c) 1996 STE CA USA.  All rights reserved.
>C     C FUNCTIONS
>D     Download an Intel Extend Hex file into SRAM
>G     Goto and Run
>H     HELP
>M     MENU
>U     Upload a block of Binary data
D
Ready to recieve Intel Extend HEX file at 19200 baud
```

3. At the hyper terminal menu, select **Transfers**, **Send Text File**. Go to c:\tern\186\rom\ae86 and select "l_tdrem.hex". This will download into the SRAM starting at address 0x04000. As it downloads you will see the terminal window fill with UUUU…

4. When it finishes, you will see:

   UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
   UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
   UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
   UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
   UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
   UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
   UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
   UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
   UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
   UUUUUUUUUUUUUUUUUEND of File Record

   CHKSUM=0

   CS:IP = 04000

   Use the "G" command to run. Type "G04000", then <enter>. It will then erase the flash and prepare to burn a HEX file into the flash.

5. When it has finished preparing the flash, you will see:

   ERASING AM29F400 SECTOR 9 0xFA000 to 0xFBFFF !

   SECTOR 9

   Ready to recieve TDREM???.HEX file at 19200 baud

6. At the terminal menu, select **Transfers, Send Text File**. Go to c:\tern\186\rom\ae86 and select "ae86_115.hex". This is the debug kernel. It will download into the flash starting at address 0xFA000.

7. When it finishes, the ACTF utility will reset and you will see the ACTF menu. Type "GFA000" then <enter> to jump to and execute the debug kernel. The start up jump address will also be set to 0xFA000.

8. Install the STEP 2 jumper. At power up, your **A-Engine86-D** will execute the debug kernel and be ready to communicate with Paradigm C/C++ for STEP 1: Debug Mode.

1.5.2 1.5.2 Burning your application HEX file into the flash

1. Follow instructions 1-2 of the above section, section 1.5.1.

2. Go to **Transfers, Send Text File**. Go to c:\tern\186\rom\ae86 and select "l_29f400.hex". When it has finished downloading, type "G04000", then <enter> to prepare the flash. The files "l_tdrem.hex" and "l_29f400.hex" are similar in that they both download into the SRAM (at address 0x04000) and prepare part of the flash for a HEX file. Yet, they differ in which part of the flash they prepare. "l_tdrem.hex" prepares only a single upper sector for the small debug kernel, while "l_29f400.hex" prepares all but the upper two sectors (0x80000 – 0xF9FFF) for your user application.

3. When it finishes preparing the flash, you will be prompted to send your application. Go to **Transfers, Send Text File** and select your application HEX file created by Section 1.4.4. Your application will download into the flash starting at address 0x80000 (not to be confused with 0x08000, the starting address of your program in

STEP 2: Standalone Mode in the SRAM). After it finishes, the ACTF utility will reset.

4. Now all that is needed is to set the jump address to 0x80000. Type “G80000” then <enter>. Your application will then execute out of the flash. The start up jump address is now set to 0x80000.
5. Install the STEP 2 jumper.
6. At every power-up, the CPU will jump to 0x8000 for immediate execution of your program. To get back to debug mode go to section 1.5.1.

1.6 Minimum Requirements for A-Engine86-D System Development

1.6.1 Minimum Hardware Requirements

> PC or PC-compatible computer with serial COMx port that supports 115,200 baud
> A-Engine86-D controller
> 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)

1.6.2 Minimum Software Requirements

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

With the EV-P Kit, you can program and debug the A-Engine86-D 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 the Development Kit (DV-P 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.

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

2.2 Hardware Installation

Overview

> Connect debug cable:
>
> To communicate with the ACTF Utility and for Debugging (STEP 1), the AE86D uses SER0 (H1). Install the IDC connector on H1 and the DB9 to your PC's COMx port.

> Connect wall transformer:
>
> Connect 9V wall transformer to power and plug into power jack adapter which installs in the two pin screw terminal, T1.

2.2.1 Connecting the FlashCore-N to the PC

The following picture (Figure 2.1) illustrates the connection between the A-Engine86-D and the PC. The AE86D is linked to the PC via serial/debug cable.

The *ae86_115.hex* debug kernel communicates through SER0 by default. Install the 5x2 IDC connector to the SER0 header (H1). **IMPORTANT:** Note that the **red** side of the cable must install onto pin 1 of the H1 header. The DB9 connector should be connected to one of your PC's COM Ports (COM1 or COM2).

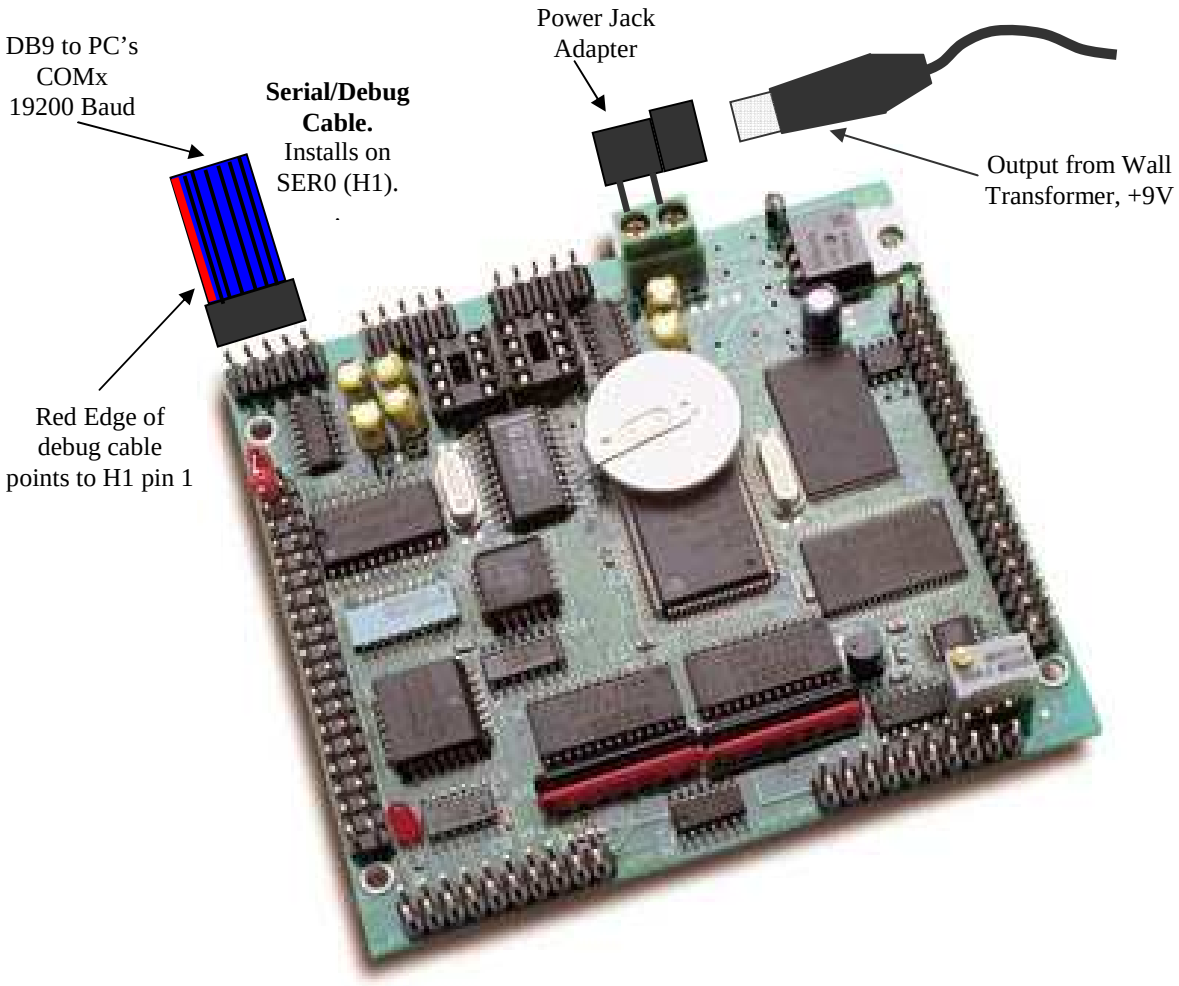


Figure 2.1 Connecting the A-Engine86-D to the PC

2.2.2 Powering-on the A-Engine86-D

Connect the wall transformer +9V DC output to the A-Engine86-D power jack adapter which connects to the A-Engine86-D via T1 pin 1(+12V In) and T1 pin 2 (GND). See Figure 2.1 above.

Important: The output of the wall transformer is center-negative. Be sure to verify correct polarity when powering on your FN.

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=P31, J2 pin 19
INT3, J2 pin 21
INT4, J2 pin 33
INT5=P12=DRQ0, J2 pin 5, used by AE86D as output for LED/EEPROM/Watchdog
INT6=P13=DRQ1, J2 pin 11, ADC U12 Busy
/NMI, J2 pin 7

Three external interrupt inputs, /INT0-1 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.

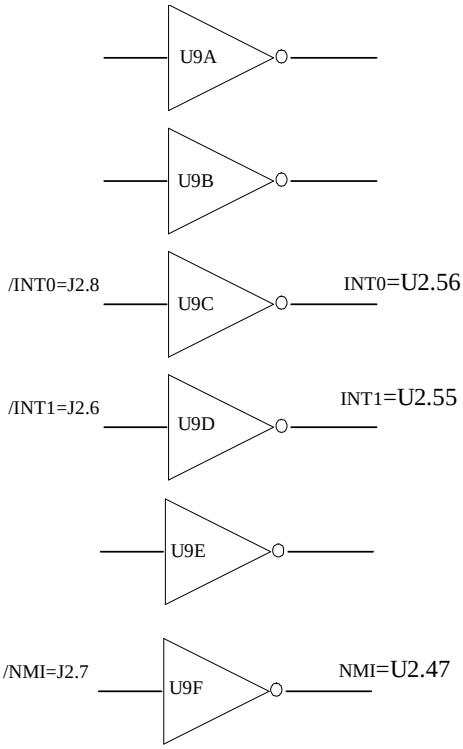


Figure 3.1 External interrupt inputs

The AE86D 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***. Serial Port 0 (SER0) is the debug port for the AE86D and is always buffered with an RS-232 driver. Serial Port 1 (SER1) is buffered with an RS-232 driver by default but is configurable to RS-485 (half-duplex) or RS-422 (full-duplex) by option.

An optional external SCC2691 UART is located in position U8. For more information about the external UART SCC2691, please refer to section 3.4.3 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 = J2 pin 14
Timer1 output = P1 = J2 pin 29
Timer1 input = P0 = J2 pin 20

Timer0 input (P11) is used by the on-board EEPEOM and 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.

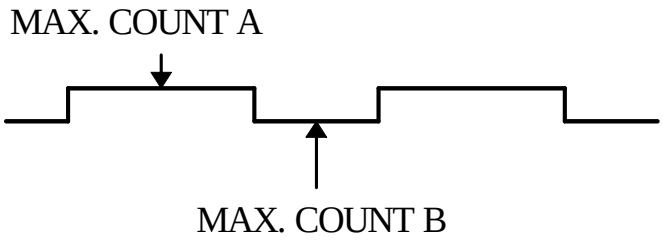
Timer2 is not connected to any external pin. It can be used as an internal timer for real-time coding or time-delay applications. It can also prescale timer 0 and timer 1 or be used as a DMA request source.

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-D 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-D 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).

Signal	**Pin**	**Function**
P2	/PCS6	U10 RTC72423 chip select at base I/O address 0x0600
P3	/PCS5	U8 SCC2691 UART chip select at base I/O address 0x0500
P4	/DT	STEP2 jumper
P7	A17	Upper address line. RESERVED ALWAYS
P8	A18	Upper address line. RESERVED ALWAYS
P11	Timer0 input	U7 24C04 EE data input
P12	DRQ0/INT5	Output for LED, U7 serial EE clock, Hit watchdog
P13	/INT6	U12 ADC BUSY
P17	/PCS1	Input line to U4 PAL. Not recommended for application

Signal	Pin	Function
P22	TxD0	Default SER0 debug
P23	RxD0	Default SER0 debug
/INT0	J2 pin 8	U8 SCC2691 UART interrupt

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, giving a clock speed of 40 MHz. Details regarding this can be found in the Software chapter, and in the Am186ES User's Manual. Slower components, such as most LCD interfaces, might find the maximum programmable wait state of 15 cycles still insufficient. Due to the high bus speed of the system, some components need to be attached to I/O pins directly.

For details regarding the chip select unit, please see Chapter 5 of the Am186ES User's Manual.

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

I/O space	Select	Location	Usage
0x0000-0x00ff	/PCS0	J1 pin 19=P16	USER*
0x0100-0x01ff	/PCS1	J2 pin 13=P17	U4 PAL Input.
			Address decoded as shown below.
0x100	L0	U11 pin 7	Load U11 DAC
0x102	L1	U15 pin 7	Load U15 DAC
0x106	L3	U11 and U15 pin 6	U11, U14 DAC Reset
0x108 – 0x10e	L4 - L7	J6 pins 2, 4, 6, 8	TTL level output
0x110 – 0x116	/DA	U11 pin 23	U11 DAC, 4 channels
0x120 – 0x126	/DA1	U15 pin 23	U15 DAC, 4 channels
0x130 – 0x136	/CNT	U14 pin 24	U14 Timer/Counter
0x180 – 0x188	/AD	U12 pin 31	AD7852, 8 channels
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 such as the FC-0 or MMB.

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

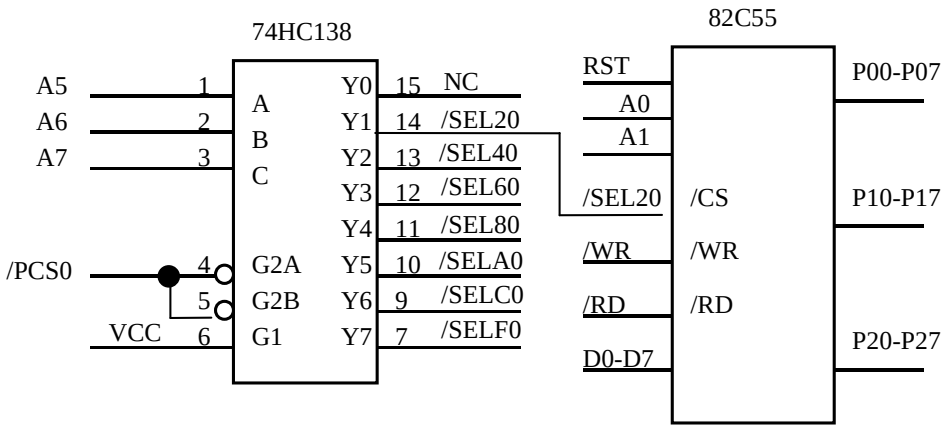


Figure 3.2 Interface the A-Engine86-D to external I/O devices

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 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\ae\samples\ae\poweroff.c`**.

3.4.3 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 the SCC2691 data sheet on the TERN CD under tern_docs\parts. The SCC2691 on the A-Engine86-D 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. The AE86D offers an additional RS-232 driver to buffer the SCC2691 UART. It can provide full-duplex RS-232 communication and is located at U17. The MAX232 (U17) offers four independent drivers to buffer RxD, TxD, CTS, and RTS. These signals are routed to the H3 5x2 pin header and are all RS-232 level.

The SCC2691utilizes an on-board 8MHz crystal to generate one of 13 different baud rates. See chapter 4 for details on initialization and baud rate selection. Whereas SER0 and SER1 (UARTs from the Am186ES) use DMA to fill the input buffer, the SCC2691 is interrupt driven. A special Interrupt-Service-Routine (ISR) is reserved for the SCC2691 to fill the input buffer. On the AE86D, /INT0 is reserved for this

purpose, and is therefore necessary to not mask or re-initialize this interrupt line. Doing so will create unpredictable behavior on the SCC2691. The schematic and chapter 4 of this manual, as well as the data sheet (tern_docs\parts) provide additional details about the SCC2691. See also the **tern\186\samples\ae** directory for sample code, including ae_scc.c, ae_scc1.c, and scc_poll.c.

3.5 Other Devices

A number of other devices are also available on the A-Engine86-D. 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-D 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-D. 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-D 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()**.

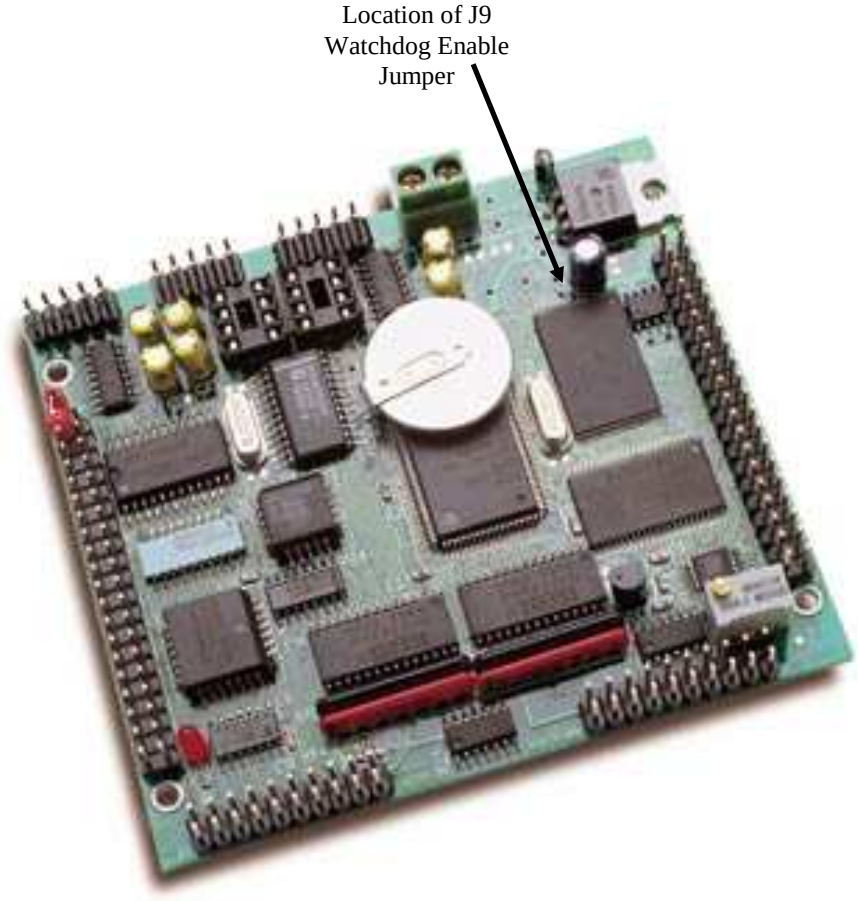


Figure 3.3 Location of watchdog timer enable jumper

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.

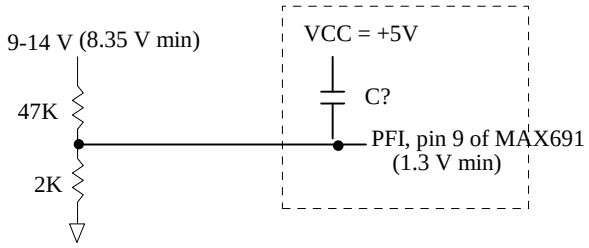


Figure 3.4 Using the supervisor chip for power failure detection

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 512 bytes (24C04) is installed at U7. The A-Engine86-D uses the P12=SCL(serial clock) and P11=SDA (serial data) to interface with the EEPROM. The EEPROM can be used to store important data such as a node address, calibration coefficients, and configuration codes. It typically has 1,000,000 erase/write cycles. The data retention is more than 40 years. EEPROM can be read and written by simply calling the functions **ee_rd()** and **ee_wr()**.

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

3.5.3 Timers / Counters

The NEC PD71054 Programmable Timer/Counter (PTC) chip provides three high-performance, programmable counters for the A-Engine86-D. Three 16-bit counters, each with its own clock input, gate control, and output pins, can be clocked from DC to 10 MHz. All related pins are located on the J6 pin header. Under software control, they can generate accurate timer delays. There are six programmable count modes. Refer to TERN's CD-ROM for AE86D schematics for more details. Sample programs are available under **c:\tern\186\samples\ae86d**. The data sheet can be found under tern_docs\parts as "82C54.pdf".

3.5.4 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 3.2 µs) conversion time to complete one conversion, based on a 5 MHz ADC clock (5MHz => 200ns/clock * 16 clocks = 3.2µs). The busy signal has a 3.2 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 **ae86d_ad.c** can be found in the **`c:\tern\186\samples\ae86d`** directory. The 7852 A/D pins are located on pins AD0 to AD7 on J4 pins 13 - 20.

3.5.5 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 5µs. Two DAC7652s can be installed on the AE86D for a total of 8 analog outputs.

In addition, these outputs are buffered by ops with a default gain of 2 and a user adjustable offset voltage, giving a final output range of 0-5V by default. The offset voltage can be asjusted to yield a final output range of -5V– 0V to +2.5V - +7.5V

The AE86D 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 **ae86d_da.c** may be found in the

c:\tern\186\samples\ae86d 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-D 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.

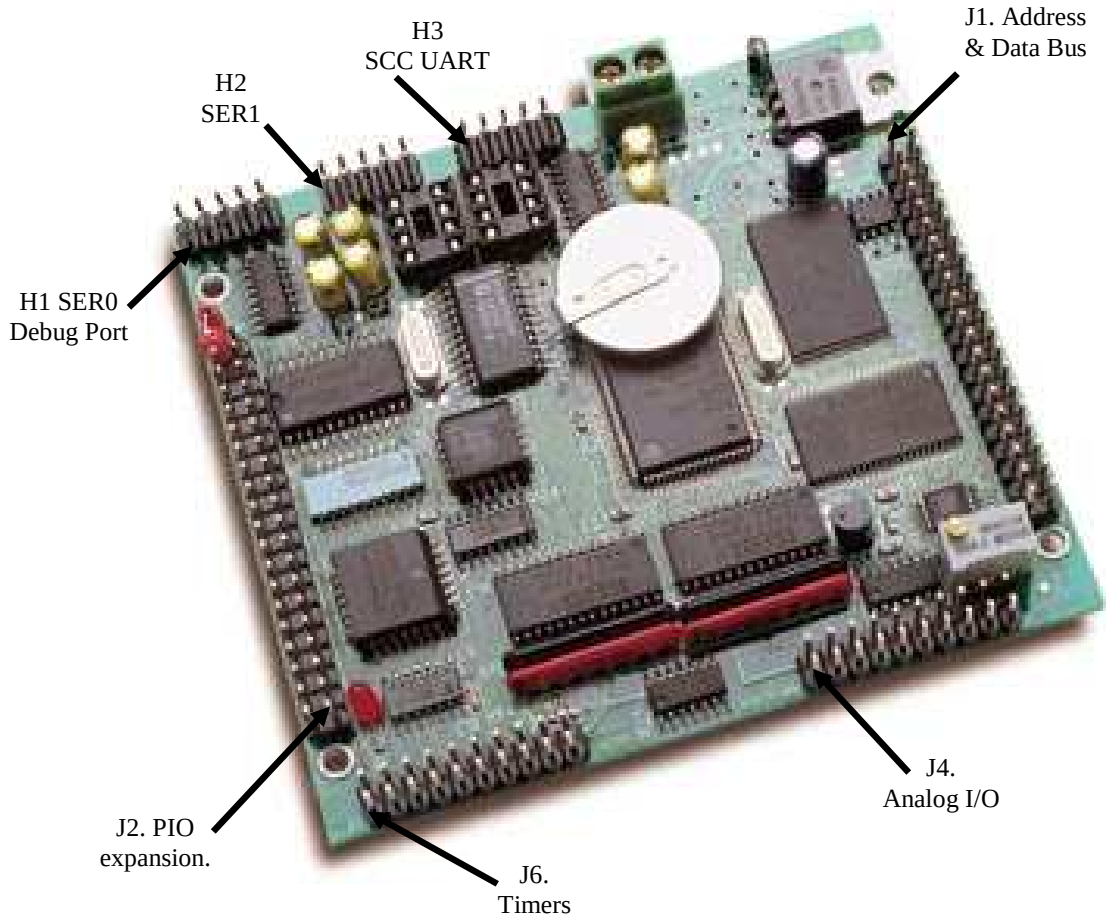


Figure 3.5 Pin 1 locations for A-Engine86-D headers. All arrows point to Pin1.

J2 Signal			
GND	40	39	VCC
P4	38	37	P14
/CTS0	36	35	P6
TxD0	34	33	P30
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	P31
P25	18	17	P24
	16	15	P3
P11	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
/INT0-1	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 and J6 connector for Timers, ADC, DAC

J4 Signal			
GND	1	2	GND
V8	3	4	V7
V6	5	6	V5
V4	7	8	V3
V2	9	10	V1
GND	11	12	GND
AD1	13	14	AD0
AD3	15	16	AD2
AD5	17	18	AD4
AD7	19	20	AD6

J6 Signal			
O0	1	2	L4
G2	3	4	L7
G1	5	6	L6
G0	7	8	L5
CNT2	9	10	T1
CNT1	11	12	T2
CNT0	13	14	T2
O1	15	16	O2
GND	17	18	VCC
V+	19	20	V-

Figure 3.6 J4and J6 connectors

3.6.3 Jumpers

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

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	10x2	ADC, DAC	
J6	10x2	Timer I/O	
J9	2x1	Watchdog timer	Enabled if Jumper is on Disabled if jumper is off

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 AE.LIB (done)

AE.LIB is a C library for basic A-Engine86-D operations. It includes the following modules: AE.OBJ, SER0.OBJ, SER1.OBJ, SCC.OBJ, and AEEE.OBJ. You need to link AE.LIB in your applications and include the corresponding header files. The following is a list of the header files:

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

4.2 Functions in AE.OBJ

4.2.1 A-Engine86-D Initialization

ae_init

This function should be called at the beginning of every program running on A-Engine86-D 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
```

The following statements do not apply to the AE86D since it does not have an 82C55.

>> 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
```

4.2.2 External Interrupt Initialization

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

4.2.3 I/O Initialization

Two ports of 16 I/O pins each are available on the A-Engine86-D. 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.

4.2.4 Timer Units

The three timers present on the A-Engine86-D 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.

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.

4.2.5 Analog-to-Digital Conversion

Parallel ADC AD7852

The high-speed AD7852 ADC unit (U12) is mapped in 0x180-0x188. To start an ADC conversion on channel 0, A I/O write, outportb(0x180+?,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(0x180); will return the previous ADC conversion result, with only upper 12-bit data D15-D4 valid. A sample program ***ae86d_ad.c*** demonstrating the use of the AD7852 is included in **`tern\186\samples\ae86d`**.

4.2.6 Digital-to-Analog Conversion

Parallel DAC7625

The high-speed DAC DA7625 (U11) is mapped in 0x110-0x116. Only even addresses are valid since we are in 16-bit external operation.

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

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

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

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

The high-speed DAC DA7625 (U15) is mapped in 0x120-0x126. Only even addresses are valid since we are in 16-bit external operation.

Use outport(0x120, dac); to write upper 12-bit D15-D4 data into DAC channel 1

Use outport(0x122, dac); to write upper 12-bit D15-D4 data into DAC channel 2

Use outport(0x124, dac); to write upper 12-bit D15-D4 data into DAC channel 3

Use outport(0x126, dac); to write upper 12-bit D15-D4 data into DAC channel 4

It is necessary to drive the /LD input of the DACs low to load the values into the DACs. The DAC at U11 is routed to signal L0, and the U15 to L1. You can call outportb to drive these low. L0 is mapped to I/O space 0x100, and L1 is mapped to I/O space 0x102. Refer to the sample code ae86d_da.c in the tern\186\samples\ae86d directory.

Details regarding hardware, such as pin-outs and performance specifications, can be found in the Hardware chapter.

4.2.7 71054 Timer/Counter

The 71054 timer/counter supports 3 16-bit independent timers/counters. Each can operate in one of 6 modes, including, one-shot, rate generator, square wave mode, software triggered mode, and hardware triggered strobe. The 71054 has 4 16-bit registers mapped into I/O space, 0x130, 0x132, 0x134, and 0x136. The control register is located at 0x136 and timer0, 1, and 2 registers are mapped into 0x130, 0x132, and 0x134 respectively. Refer to sample code, ae86d_c0.c and ae86d_c3.c, in the tern\186\samples\ae86d

directory. The data sheet will also help for programming the counter in other modes. The data sheet is named 82C54.pdf in the tern_docs\parts directory.

4.2.8 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.

4.3 Functions in SER0.OBJ/SER1.OBJ

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

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

Two asynchronous serial ports are integrated in the 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-D provided at the end of this manual.

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

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

Function Argument	Baud Rate
1	110
2	150
3	300

Function Argument	Baud Rate
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.

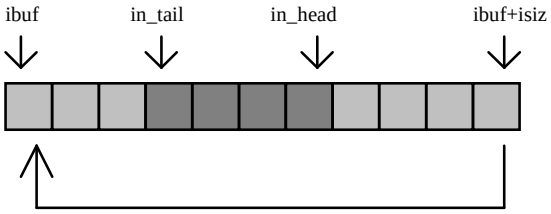


Figure 4.1 Circular ring input buffer

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, and no parity communication.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Miscellaneous Serial Communication Functions

One thing to be aware of in both transmission and receiving of data through the serial port is that TERN drivers only use the basic serial-port communication lines for transmitting and receiving data. Hardware flow control in the form of **CTS** (Clear-To-Send) and **RTS** (Ready-To-Send) is not implemented. There are, however, functions available that allow you to check and set the value of these I/O pins appropriate for whatever form of flow control you wish to implement. Before using these functions, you should once again be aware that the peripheral pin function you are using might not be selected as needed. For details, please refer to the Am186ES User's Manual.

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

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

Completing Serial Communications

After completing your serial communications, 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.

4.4 Functions in SCC.OBJ

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

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

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

Function Argument	Baud Rate
1	110
2	150
3	300
4	600
5	1200
6	2400
7	4800
8	9600 (default)
9	19,200
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: **putser*n***

putsers_scc
See: **putsers*n***

getser_scc
See: **getser*n***

getsers_scc
See: **getsers*n***

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

scc_cts
See: **s*n*_cts**

scc_rts
See: **s*n*_rts**

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

scc_close
See: **s*n*_close**

serhit_scc
See: **s*n*_hit**

clean_ser_scc
See: **clean_s*n***

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

scc_err
Arguments: none
Return value: unsigned char val
The returned value **val** will be in the form of 0ABC0000 in binary. Bit A is 1 to indicate a framing error. Bit B is 1 to indicate a parity error, and bit C indicates an over-run error.

4.5 Functions in AEEE.OBJ

The 512-byte serial EEPROM (***24C04***) provided on-board allows easy storage of non-volatile program parameters. This is usually an ideal location to store important configuration values that do not need to be

changed often. Access to the EEPROM is quite slow, compared to memory access on the rest of the controller.

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

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

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

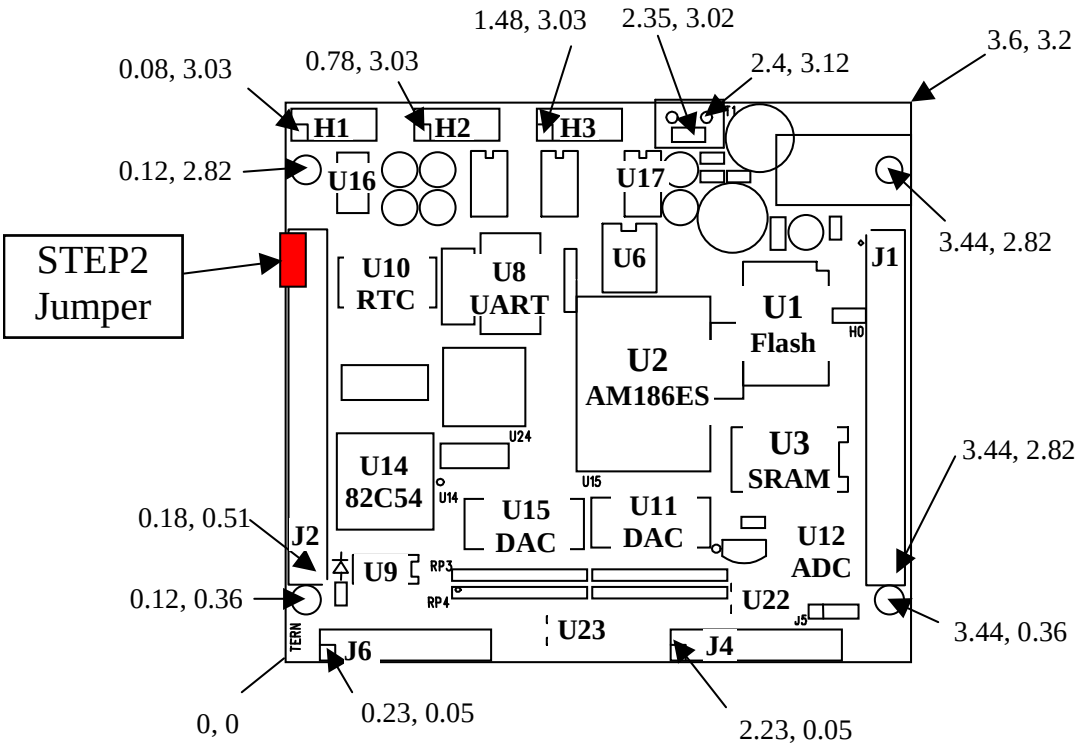
The EEPROM shares line P11 with the ADC. If the ADC is enabled, it can interfere with the EEPROM. The ADC is enabled if I20 is low. In the init function, it is brought high so that you can access the EEPROM. Be aware that if you modify the PPI control register by calling outportb(0x0103, xx); then all 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).

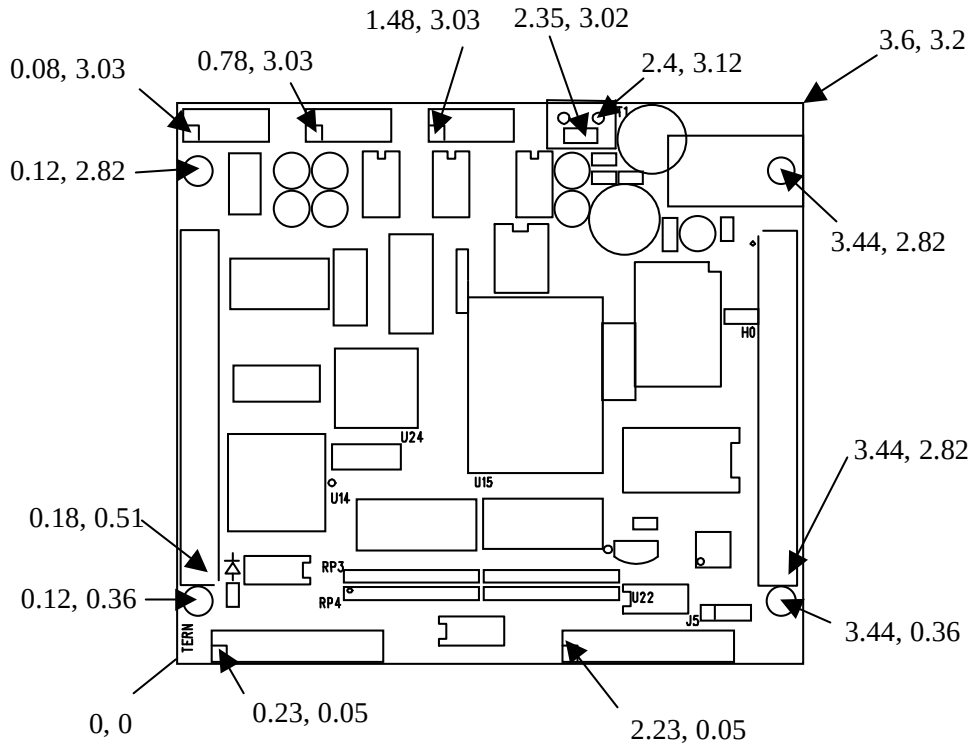
ee_wr
Arguments: int addr, unsigned char dat
Return value: int status

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.

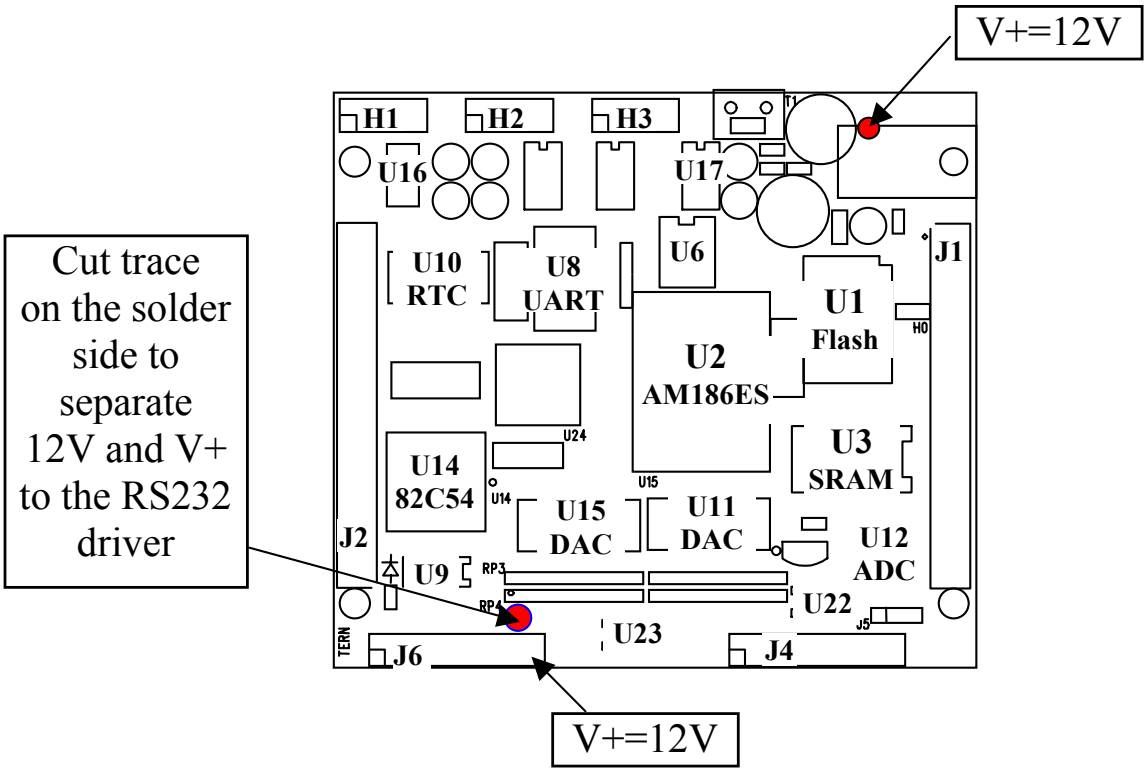




AE86-D layout

A-Engine86-D Modification for

DAC 0-10V analog outputs Gain=4



As default, the eight channels of DAC7625 are buffered by on-board amplifier with Gain=2, outputting 0-5V.
The amplifiers are powered by V+, generated from the RS232 drivers.

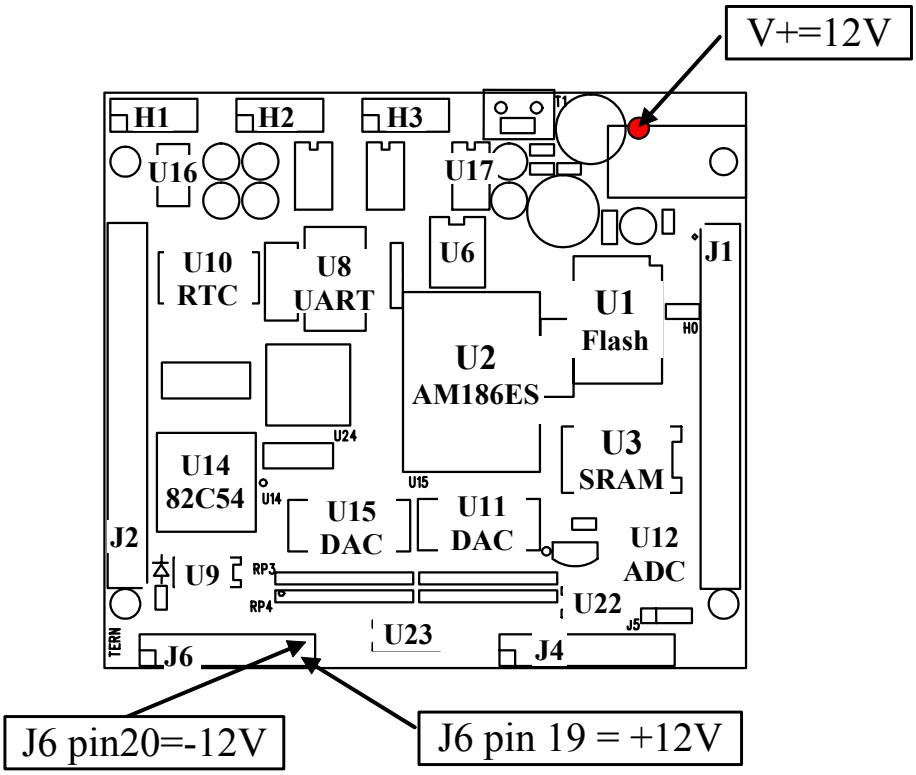
In ordered to output 0-10V for the DAC, the amplifiers on-board must be powered by 12V.

Modifications:

1) Change Gain=4. RP1=RP3=4.99K, RP2=RP4=20K.
2) Cut trace on the solder side to separate the RS232 V+ from J6 pin 19.
3) Add wire from Diode input = 12V to J6 pin 19.

A-Engine86-D Modification for

DAC -10V to +10V analog outputs Gain=8

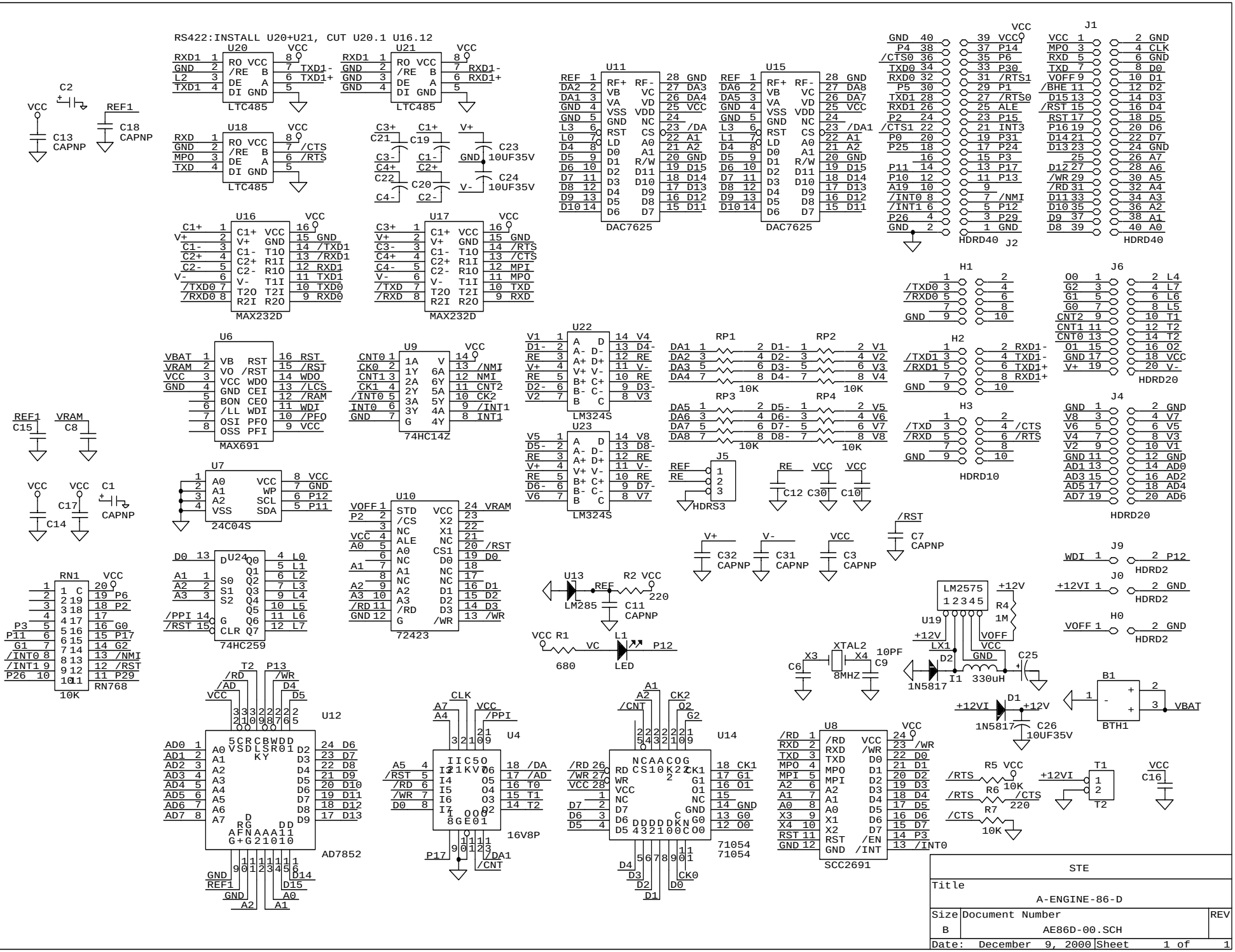


As default, the eight channels of DAC7625 are buffered by on-board amplifier with Gain=2, outputting 0-5V.
The amplifiers are powered by V+ = +8V and V- = -8V,
generated from the on-board RS232 drivers.

In ordered to output +-10V for the DAC, the amplifiers on-board must be powered by +-12V.

Modifications:

1) Change Gain=8. RP1=RP3=2K, RP2=RP4=16.2K.
2) Supply external -12V(max. -14V) to J6 pin 20.
3) Add wire from Diode input = 12V(max. +14V) to J6 pin 19.
   The Max. DC power voltage to AE86D must be less than +14V.


RS422:INSTALL U20+U21, CUT U20.1 U16.12
U20
LTC485
U21
LTC485
U18
LTC485
U11
DAC7625
U15
DAC7625
J2
HDRD40
J1
HDRD40
C2
C13
CAPNP
C18
CAPNP
REF1
C23
10UF35V
C24
10UF35V
C19
C20
C21
C22
U16
MAX232D
U17
MAX232D
H1
H2
H3
HDRD10
J6
HDRD20
J4
HDRD20
U6
MAX691
U9
74HC14Z
U22
LM324S
U23
LM324S
RP1
RP2
RP3
RP4
10K
J5
HDRS3
C12
C30
C10
U7
24C04S
U10
72423
C7
CAPNP
C32
CAPNP
C31
CAPNP
C3
CAPNP
C15
C8
C17
C14
C1
CAPNP
RN1
10K
RN768
U24
74HC259
U13
LM285
R2
220
C11
CAPNP
R1
680
L1
LED
LM2575
U19
R4
1M
L1
330uH
D2
1N5817
C25
XTAL2
8MHZ
10PF
C9
C6
D1
1N5817
C26
10UF35V
B1
BTH1
J9
HDRD2
J0
HDRD2
H0
HDRD2
U12
AD7852
U4
16V8P
U14
71054
71054
U8
SCC2691
R5
10K
R6
220
R7
10K
T1
T2
C16
STE
Title
A-ENGINE-86-D
Size
B
Document Number
AE86D-00.SCH
REV
Date: December 9, 2000
Sheet 1 of 1