C-Eye™

Low Cost, C/C++ Programmable, Standalone Vision System

Technical Manual

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

COPYRIGHT

C-Eye is a trademark of TERN, Inc.
Am186ES is a trademark of Advanced Micro Devices, Inc.
Paradigm C/C++ is a trademark of Paradigm Systems.
Microsoft, Windows, Windows98/2000/ME/NT/XP are trademarks of Microsoft Corporation.
IBM is a trademark of International Business Machines Corporation.

Version 2.00

October 21, 2010

© 1993-2010 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

Introduction

TERN *C-Eye™* controller is an innovative new solution for a wide range of vision applications: machine vision; check ID marking; pattern recognition; industrial process control; motion position detection; security monitoring.

The *C-Eye™* is the ideal board for adding low-power standalone digital image acquisition and recording to any embedded application.

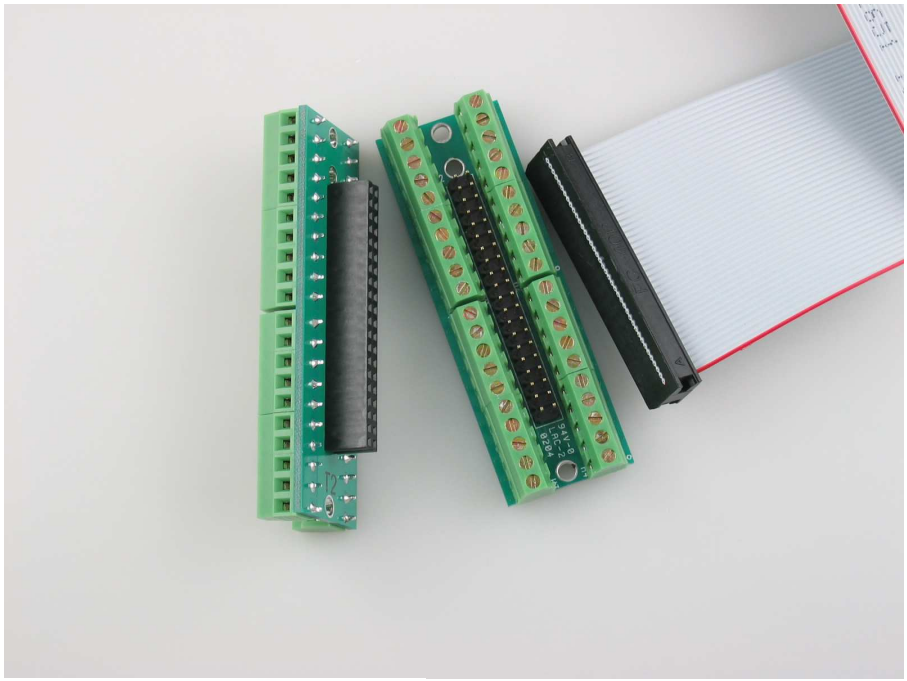
CMOS image sensors have become widely used on platforms like cellular phones, or PC/Web-based remote cameras. However, these cameras generally rely on a connection to other central systems for data storage, image processing, or power.

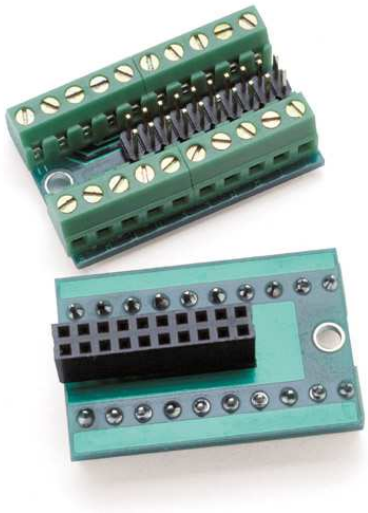
The *C-Eye™* provides a true stand-alone solution: real-time images available in bitmap format allow for simpler machine vision, low-power consumption with pre-programmed wake-up permits long-term deployment in the field, and tens of thousands of images can be stored (and later transferred to a PC) with onboard removable CompactFlash memory cards (up to 2 GB) .

The C/C++ programmable *C-Eye™* controller also allows the user application to access to any pixel of the hardware grabbed 640x480 image. User can program the *C-Eye™* to capture images, analyze any zones of interested pixels, and make control decisions based on that image processing result in real-time.

The *C-Eye™* is a complete controller including a 16-bit 40 MHz x86 CPU, 512KB Flash, battery backed SRAM, 1 MB image FIFO, an image sensor, two serial ports and a CompactFlash interface. The on-board image sensor has 640x480 active pixels. The image acquisition is done by hardware. After issuing a software command, the image data will be captured and stored in the on-board 1MB image buffer FIFO. The *C-Eye™* can capture, process and store .BMP files in the FAT16 compatible CF card at a rate of up to 4 frames per second. The 100M BaseT Ethernet port has TCP/IP support.. Two RS232 serial ports (SER0 and SER1) can handle 115,200 baud with high reliability. SER1 can also be hardware configured as RS485. There are: a real time clock with Battery backup, 10+ TTL I/O pins and 3 16-bit timer/counters. A high speed 16-bit parallel data-bus expansion header supports external USB interface for high speed data transfer to a PC.

Within 3x4 inches, the *C-Eye™* is designed to fit into an Aluminum Extrusion Enclosure for easy deployment and installation. All signals on two 20x2 pin headers are accessible outside of the enclosure. Optional high efficient Switching Regulator (LM2575) provides a shutdown feature(VOFF). It can enter µA poweroff mode and can be woken up by active-low signal from the on-board RTC or external signal. The poweroff mode allows the *C-Eye™* to operate in a remote location for a long period using only battery power. The *C-Eye™* uses 8.5V to 12V DC power supply with default linear regulator, or up to 30V DC with switching regulator without generating excessive heat. The user can use W40 (40 field removable screw terminals) to access all I/O signals.





W40 (photo on the TOP) and W20 (photo on the left) are field removable screw terminals. These convert pin header signals to screw terminals.

J1 Extension

You can connect the *C-Eye*™ to a PC's USB port with a TERN's CUSB interface (See **Chapter 2**). An utility *C-Eye Viewer.exe* from TERN is available to run on the PC, enable high speed transfer and display images from *C-Eye*™ to the PC via USB port.
The CUSB is connecting to the *C-Eye*™ board's expansion port (J1) via a flat cable. The J1 is installed with a type "B" header under the CF, with long pins at the bottom side of the PCB. A USB A-B cable is used.
A DEBUG cable (IDE10-DB9) is installed on SER0 and is connecting to a PC RS232 port for software remote downloading and debugging.
The 2nd RS232 port (SER1) is available and free to use. User can also use the *C-Eye Viewer.exe* utility to transfer images from the *C-Eye*™ to a PC via RS232 SER1 port link.

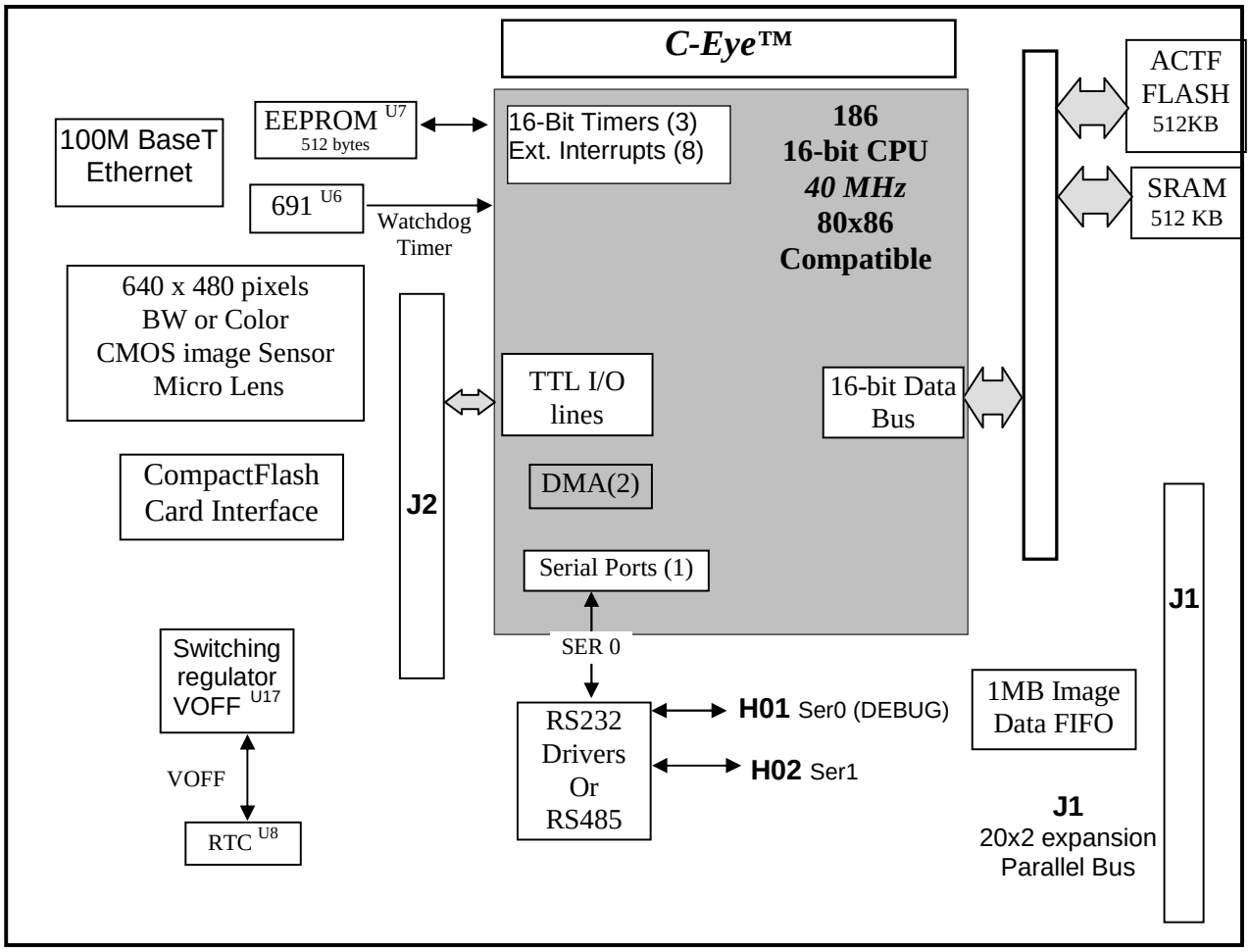


Figure 1.1 Functional block diagram of the C-Eye™

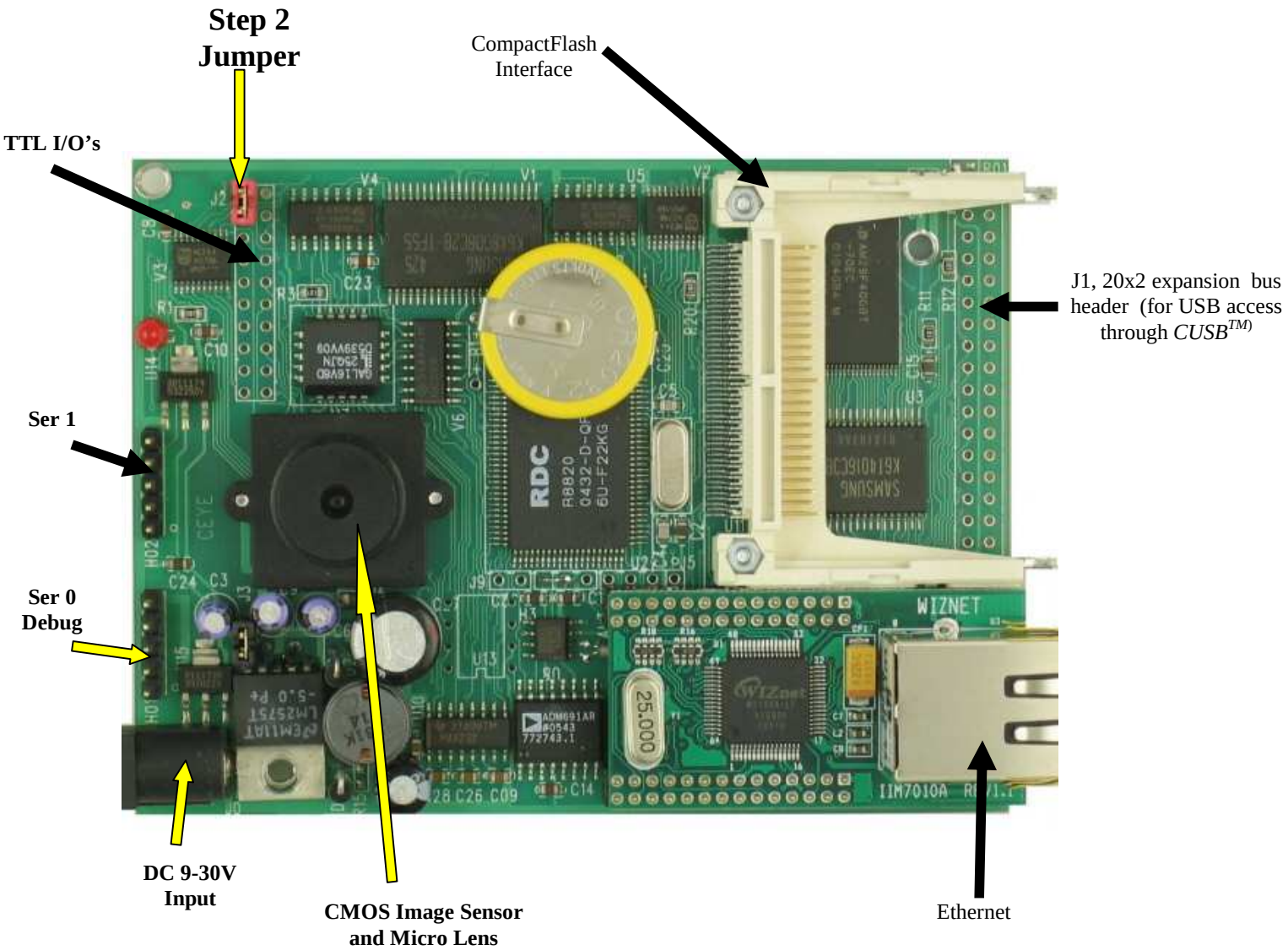
Features:

Features:

* CMOS Image Sensor (640x480, 320x240)
* Supports grayscale, 256-color, 24-bit color
* Wide viewing angle Micro Lens
* 4x3”, 9-30V DC Power, Peak <1W
* x86 16-bit CPU, CF with FAT file system
* 100M BaseT Ethernet with TCP/IP
* RS232/RS485, RTC, Battery.
* Real time clock, TTL I/Os

1.2 Physical Description

Below shows the physical description of the C-Eye™.



Programming Overview

An "ACTF Boot Loader" resides in the top protected sector of the 256KW on-board flash chip (29F400). At power-on/reset, the ACTF Utility will check the STEP 2 jumper (J2 pins 1+3). If the STEP 2 jumper is installed, the "jump address" located in the on-board serial EEPROM will be read out and the CPU will jump to that address for immediate execution. A DEBUG kernel (already pre-programmed at the factory) can be downloaded and programmed into the flash starting at address 0xFA000. Using the ACTF Utility, the "GFA000 <enter>" command will set the jump address to 0xFA000. The command will also run the DEBUG kernel, preparing the *C-Eye*™ for communication with the Paradigm C/C++ IDE for downloading and debugging applications. The following diagrams show the procedure for programming the *C-Eye*™. Steps include preparing the *C-Eye*™ for debugging, debugging the *C-Eye*™, standalone field test, and production.

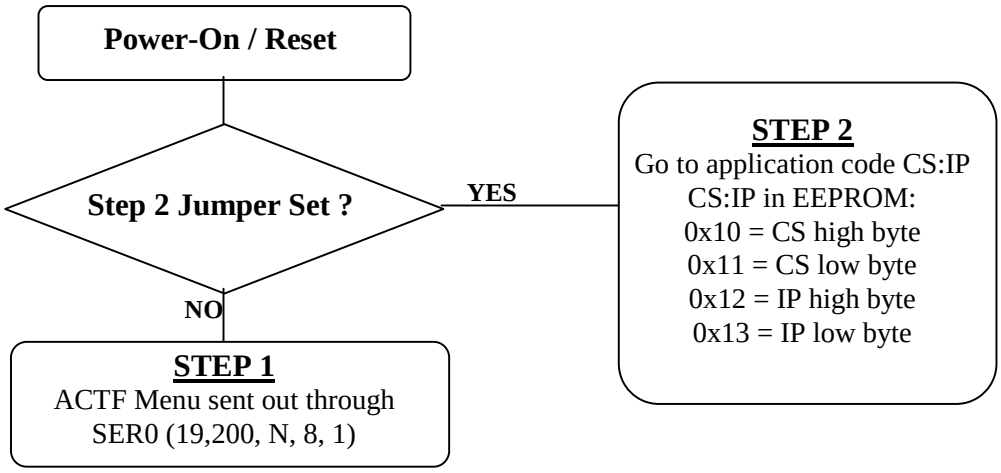


Figure 1.2 Flow Chart of ACTF Operation

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

Preparation for Debugging:

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

● Connect the SER0 to PC (COMx) via serial debug cable provided with the EV-P/DV-P. Using the Windows "Hyper Terminal", create a serial link based on 19,200, 8 bits, 1 stop, no parity.
● Power on WITHOUT the STEP 2 jumper installed (J2 pins 1&3). The ACTF text MENU should be sent out via serial port to "Hyper Terminal".
● Use the "D <enter>" command to initiate download. Select Transfer -> Send File, and select \tern\186\rom\re\l_debug.hex. Use the "G04000 <enter>" command to execute this script.
● Select Transfer -> Send File to select \tern\186\rom\ae86\L_29F400.hex. This is the debug kernel. Use the "GFA000 <enter>" command to set jump address and execute the debug kernel. The LED will blink twice and remain on. With the 40MHz board, use "ae86_115.hex".
● Set the STEP 2 jumper (J2 pins 1 & 3). The C-Eye is now ready to communicate with the Paradigm C/C++ IDE for debugging and application development.

Step 1: Debugging:

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

Step 2: Standalone Field Test:

● After completing STEP 1, by default, your application resides in the battery-backed SRAM starting at address 0x08000 (may be 0x10000 for ceye.ide project).
● Remove STEP 2 jumper and setup Hyper Terminal link via SER0. (Open Windows "Hyper Terminal" program. Set for 19,200, 8 bits, 1 stop, no parity).
● At power-on, ACTF menu will be sent to Hyper Terminal. Use the "G08000 <enter>" or "G10000" command to execute application. Install the STEP 2 jumper. At every power-on/reset, application at 0x08000 (or0x10000) will execute.
● Complete STANDALONE FIELD TEST. If return to STEP 1 is required, remove STEP 2 jumper and use the "GFA000 <enter>" command to run debug kernel to prepare to setup for communication with Paradigm C/C++ IDE.

Step 3: Production:

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

● Refer to Section 3.3 of the ACTF technical manual, found in the \tern_docs\manuals directory. Here you will find details on generating an ACTF downloadable HEX file based upon you application.
● Remove the STEP 2 jumper and create serial link using Hyper Terminal (19,200, N, 8, 1). At power-on/reset, you will see the ACTF menu at Hyper Terminal. Use the "D <enter>" command to initiate download process. Select Transfers -> Send Text File, and select \tern\186\rom\ae86\l_29f400.hex.
● This file will erase the flash and prepare the flash to accept ACTF downloadable application HEX file. Use the "G04000 <enter>" command to run script. Flash will be ready for application.
● Select Transfer -> Send Text File to select your ACTF downloadable application HEX file. Upon completion, use the "G80000 <enter>" command to execute application. This command also sets the jump address to point you application in flash. Set STEP 2 jumper (J2 pins 1&3). At power-on/reset application will execute.

The user's application program must reside in the SRAM (starting at address of 0x08000 by default based on \tern\186\config\186.cfg) for debugging in STEP 1, reside in the battery-backed SRAM for standalone field testing in STEP 2, and finally be programmed into the on-board flash for a complete product. For production, the user must produce an ACTF-downloadable HEX file for the application based on the DV-P Kit. From the ACTF Utility, use the command "G80000 <enter>" to point to the user's application code in the flash. The STEP 2 jumper must installed for every production-version board.

1.3 Minimum Requirements for System Development

Minimum Hardware Requirements

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

Minimum Software Requirements

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

Chapter 2: Installation

2.1 Software Installation

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

2.2 Hardware Installation

Overview

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

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

2.2.1 Connecting to the PC

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

The controller communicates through SER0 by default. Install the 5x2 IDE connector on the SER0 5x1 pin header (See Appendix C for connection details). ***IMPORTANT:*** *Note that the* ***red*** *side of the cable must point to pin 1 of the SER0 header.* The DB9 connector should be connected to one of your PC's COM Ports (COM1 or COM2).

2.2.2 Powering-on the C-Eye™

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

Connect +9-12V DC to the DC power terminal. The DC power jack adapter is center negative.

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

(See next page for connection diagram).

2.2.3 Connecting the C-Eye™

The proper connections required to debug the board (through Paradigm software).

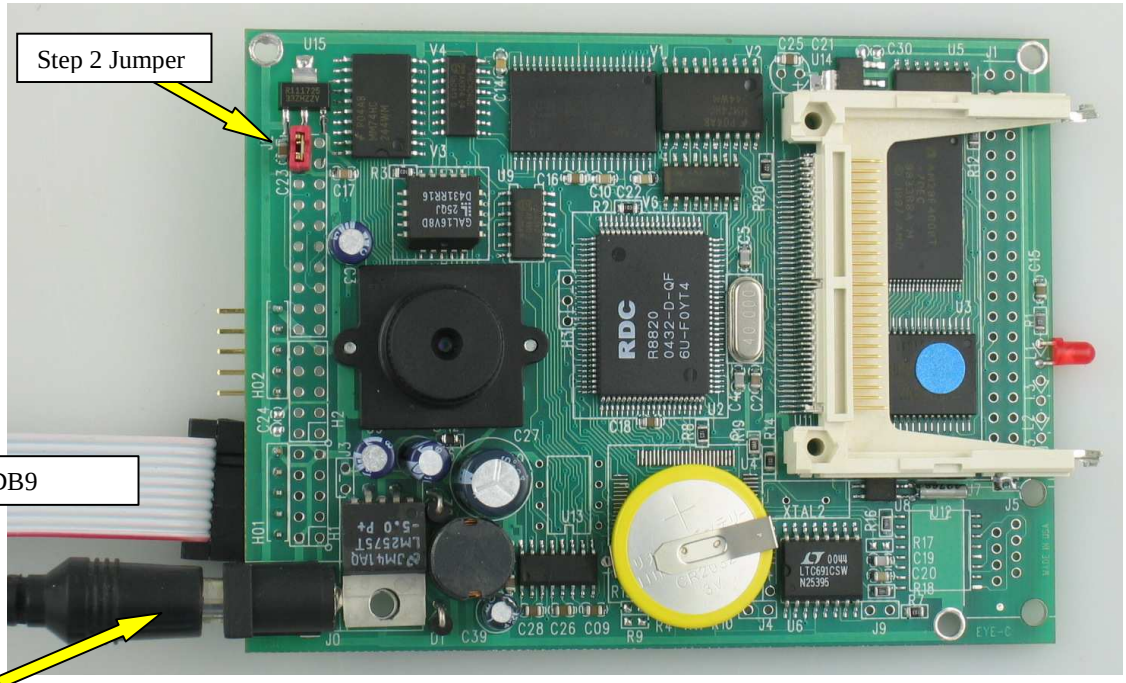


Figure 2.1: Debug Cable (Ser0), Power-Plug, and Step 2 Jumper shown.

NOTE: Remember to watch for the "**double blink**" off the LED. This indicates the **Debug Kernel** has been loaded with the **jump address** pointing to it. This is mandatory to commence downloading code through the Paradigm environment.

2.3 Getting Started – Taking a picture

The *C-Eye*™ project is in the path C:\Tern\186\Samples\CEye\CEye.IDE

2.3.1 The C-Eye™ Viewer

A program labeled *C-Eye Viewer* should be included in C:\Tern\186\Samples\CEye.

Figure 2.2: "C-Eye Viewer" location

This program will be used to upload images onto your PC.

2.3.2 "C-Eye Viewer" with RS232

To use *C-Eye Viewer* through SER 1 (H02), connect a secondary cable to H02.

1. Make sure this new cable has the same orientation as the SER 0 (H01) debug cable (red edge of cable nearest to pin 1 of header).

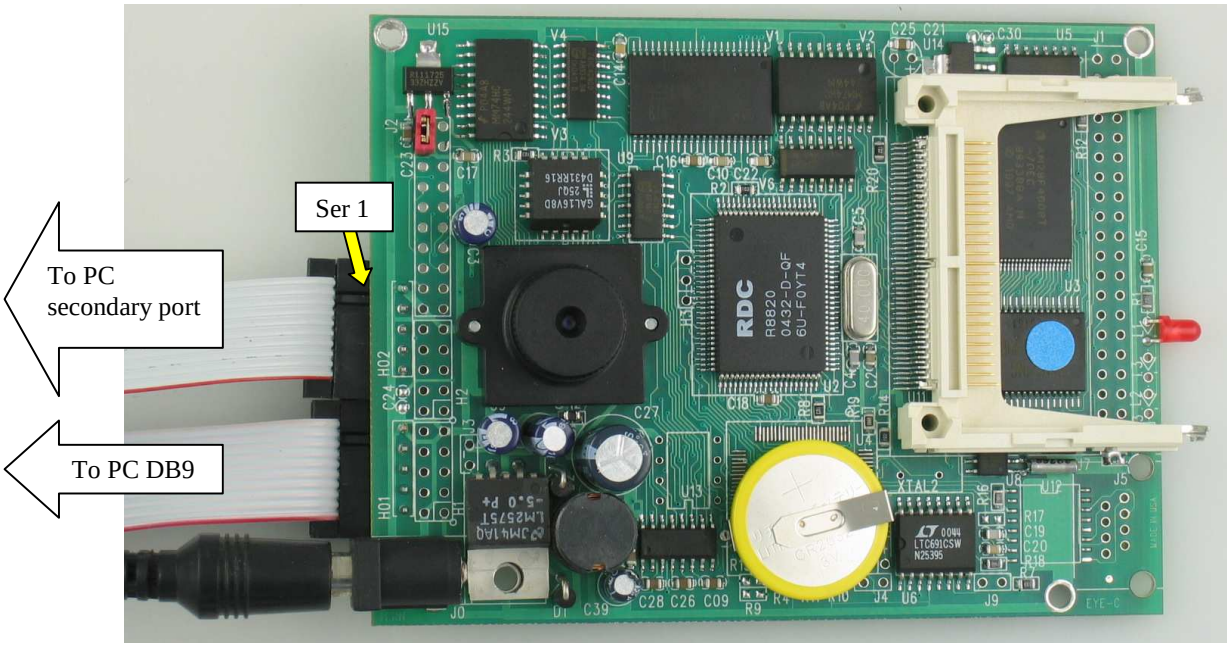


Figure 2.3: Serial port 1 connected

2. Download the sample C:\Tern\186\Samples\CEye\s232_bw320.axe.
3. Run the sample at full speed (Ctrl+F9).
4. Open the *C-Eye Viewer* application.

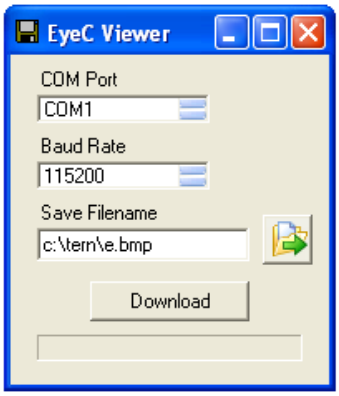


Figure 2.4: C-Eye Viewer application

5. Select your COM port, correct BAUD rate, and input a file name in which to save the picture.

   NOTE - Make sure to highlight the selected COM port.

6. To take a picture upright, position the *C-Eye*™ as shown in Figure 2.5.

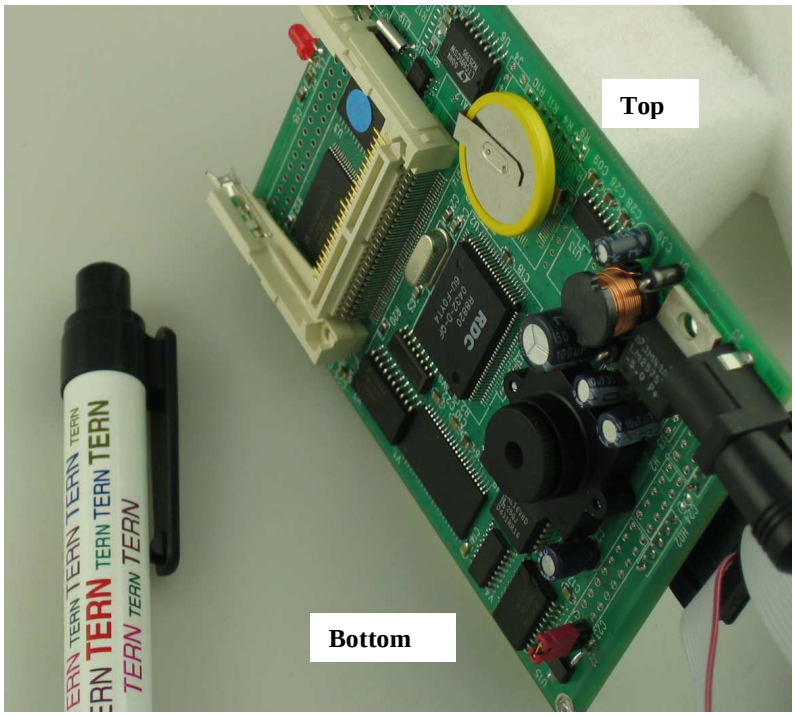


Figure 2.5: Upright picture position

7. Point the *C-Eye*™ at your desired target, and press "**Download**" on the *C-Eye Viewer*.

8. You will see your picture open on your screen

.

Figure 2.6: *C-Eye*™ 320x240 Image

2.3.3 "C-Eye Viewer" with USB

To use *C-Eye Viewer* through the USB port on your PC, you will need the *C-USB*™ board by Tern.

This board (shown in Figure 2.7) will interface the *C-Eye*™ board with your PC's USB port through the J1 header.

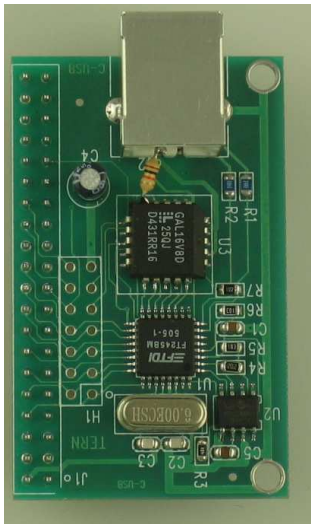


Figure 2.7: Tern C-USB™

1. Connect the *C-USB*™ and the *C-Eye*™ as in the following figure.

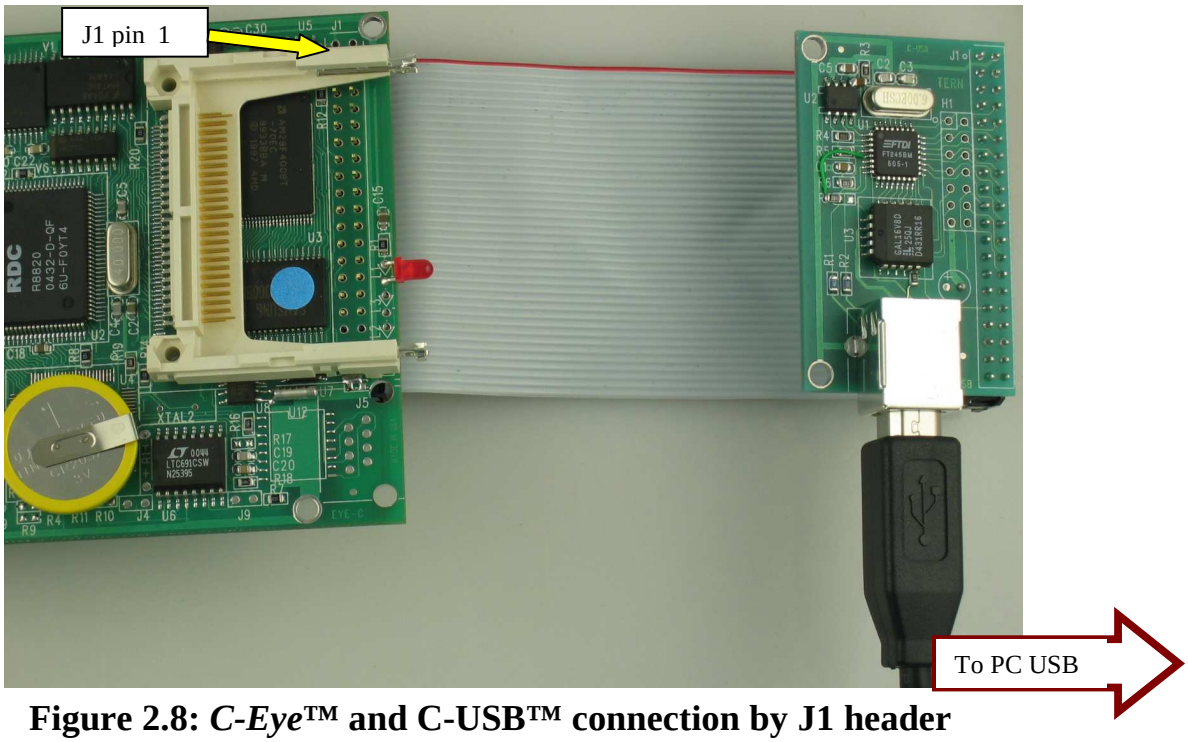


Figure 2.8: *C-Eye*™ and C-USB™ connection by J1 header

2. Download the sample C:\Tern\186\Samples\CEye\usb_bw320.axe.
3. Run the sample at full speed (Ctrl+F9).
4. Open the *C-Eye Viewer* application.
5. Select your COM port and input a file name in which to save the picture (BAUD rate not needed).
6. To take a picture upright, position the *C-Eye*™ as shown in Figure 2.5.
7. Point the *C-Eye*™ at your desired target, and press "**Download**" on the *C-Eye Viewer*.

2.3.4 Tips

1. Do NOT touch the lens.
2. Be cautious when turning the lens for focus. At a certain the point the focus lens will come loose, so turn carefully.

Chapter 3: Hardware

3.1 *Am186ES or RDC R8820*

The *C-Eye*™ can use two different CPUs. Both offer and support the same on-board peripherals as well as the on the CPU itself. The Am186ES, from AMD, has been End-of-life, while the R8820, from RDC, is active and is in full production.

The Am186ES/R8820 is based on the industry-standard x86 architecture. The Am186ES/R8820 controllers are higher-performance, more integrated versions of the 80C188 microprocessors. The Am186ES/R8820 has two asynchronous serial ports, 32 PIOs, a watchdog timer, additional interrupt pins, DMA to and from serial ports, a 16-bit reset configuration register, and enhanced chip-select functionality.

Clock

The system CLKOUTA signal is routed to J1 pin 4, default 40 MHz. CLKOUTA remains active during reset and bus hold conditions. The *C-Eye*™ initial function ae_init(); disables CLKOUTA and CLKOUTB with clka_en(0); and clkb_en(0);

You have to use clka_en(1); to enable CLKOUTA=CLK=J1 pin 4, in order to drive the CMOS image sensor.

External Interrupts and Schmitt Trigger Input Buffer

There are six external interrupts: INT0-INT4 and NMI.

/INT0, is routed to J2 pin 17. Free to use, Edge active high to low.
INT1 is not available.
INT2, J2 pin 19, tied to alarm output of the DS1337 RTC, can be shared with user application
INT3 is not available.
/INT4, used by Ethernet controllers CS8900A

NMI, tied 74HC14 inverters to /PFO of MAX691 supervisor chip. (More in Section 3.7)

The *C-Eye*™ uses vector interrupt functions to respond to external interrupts. Refer to the Am186ER User's manual for information about interrupt vectors.

Asynchronous Serial Port

The Am186ES and R8820 CPU has two asynchronous serial ports: SER0 and SER1. They support the following:

- Full-duplex operation
- 7-bit, and 8-bit data transfers
- Odd, even, and no parity
- One or two stop bits
- Error detection
- Hardware flow control
- DMA transfers to and from serial port
- Transmit and receive interrupts
- Maximum baud rate of 1/16 of the CPU clock speed
- Independent baud rate generators

The software drivers for the Async. serial port implement a ring-buffered DMA receiving and ring-buffered interrupt transmitting arrangement. See the sample file *s0_echo.c*

Timer Control Unit

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

Timer0 and Timer1 are connected to four external pins:

Timer0 output	= P10	= J2 pin 13
Timer0 input	= P11	used by EE, RTC
Timer1 output	= P1	= J2 pin 12
Timer1 input	= P0	= J2 pin 7

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

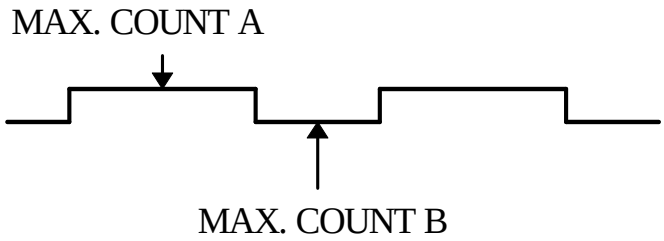
Timer2 is not connected to any external pin. It can be used as an internal timer for real-time coding or time-delay applications. It can also prescale timer 0 and timer 1 or be used as a DMA request source.

The maximum rate at which each timer can operate is 10 MHz, since each timer is serviced once every fourth CPU clock cycle. Timer inputs take up to six clock cycles to respond to clock or gate events. See the sample programs ***timer0.c*** and ***ae_cnt0.c*** in the **`\186\samples\ae`** directory.

PWM outputs

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.



Power-save Mode

The *C-Eye*™ is an ideal core module for low power consumption applications. The power-save mode of the Am186ES and R8820 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 DS1337 on the *C-Eye*™ has a VOFF signal routed to the switching regulator, LM2575. VOFF is controlled by the battery-backed DS1337. The VOFF signal can be programmed by software to be in tri-state or to be active low. The DS1337 can be programmed in interrupt mode to drive the VOFF pin at 1 second, 1 minute, or 1 hour intervals. The user can use the VOFF line to control the switching power supply that turns the power supply on/off. An example program using the VOFF signal can be found in \tern\186\samples\eyec\eye_voff.c.

PIO lines

The Am186ES/R8820 has 32 pins 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 a "normal" function pin. 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***	C-Eye ***Pin No.***	C-Eye ***Initial after ae_init(); function call***
P0	Timer1 in	Input with pull-up	J2 pin 7	Input with pull-up
P1	Timer1 out	Input with pull-down	J2 pin 12	Input with pull-up
P2	/PCS6/A2	Input with pull-up	H3.3	/PCS6
P3	/PCS5/A1	Input with pull-up	J2 pin 15	/PCS5
P4	DT/R	Normal	J2 pin 3	Input with pull-up: Step 2
P5	/DEN/DS	Normal	J2 pin 11	Input with pull-up
P6	SRDY	Normal	J2 pin 6	Input with external pull-up
P7	A17	Normal	N/A	A17
P8	A18	Normal	N/A	A18
P9	A19	Normal	N/A	N/A
P10	Timer0 out	Input with pull-down	J2 pin 13	Input with pull-down
P11	Timer0 in	Input with pull-up	N/A	N/A, EE, RTC
P12	DRQ0	Input with pull-up	RTC, EE, LED, WD	Output
P13	DRQ1	Input with pull-up	J2 pin 16	Input with pull-up
P14	/MCS0	Input with pull-up	J2 pin 4	Input with pull-up
P15	/MCS1	Input with pull-up	N/A	N/A
P16	/PCS0	Input with pull-up	J1 pin 19	/PCS0
P17	/PCS1	Input with pull-up	N/A	/PCS1 for CF
P18	/PCS2	Input with pull-up	J2 pin 5, /CTS1	Input with pull-up
P19	/PCS3	Input with pull-up	J2 pin 10, /RTS1	Input with pull-up
P20	RTS0	Input with pull-up	N/A	Input with pull-up
P21	CTS0	Input with pull-up	N/A	Input with pull-up
P22	TXD0	Input with pull-down	U10.10	Output
P23	RXD0	Input with pull-down	U10.9	Input with pull-down
P24	/MCS2	Input with pull-up	J2 pin 14	Input with pull-up
P25	/MCS3	Input with pull-up	J2 pin 9	Input with pull-up
P26	UZI	Input with pull-up	J2 pin 8	Input with pull-up*
P27	TxD1	Input with pull-up	U13.4, U10.11	TxD1
P28	RxD1	Input with pull-up	U13.1, U10.12	RxD1
P29	S6/CLKSEL1	Input with pull-up	J2.18	Output
P30	INT4	Input with pull-up	/INT4 = U4.32	Input with pull-up
P31	INT2	Input with pull-up	U8.7 RTC Alarm	Input with pull-up

* Note: P6, P26 and P29 must NOT be forced low during power-on or reset.

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

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

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

```
outport(0xff78,0xc7bc);      // PDIR1: TxD, RxD, PCS0, PCS1, P29& P22 Output
outport(0xff76,0x2040);      // PIOM1
outport(0xff72,0xee73);      // PDIR0: A18, A17, PCS6, PCS5, P12 Output
outport(0xff70,0x1040);      // PIOM0
```

The C function in the library ae.lib can be used to initialize PIO pins.

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

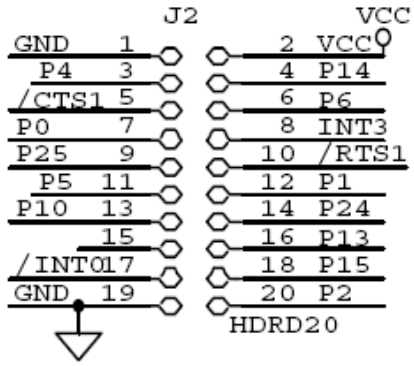
Where bit = 0-31 and mode = 0-3, see the table above.

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 *C-Eye*™ system for on-board components. You can only use the PIO pins available on J2 header (see figure below). See the appended C-Eye -Man schematics for more details.



3.2 *Real-time Clock DS1337*

The DS1337 serial real-time clock is a low-power clock/calendar with two programmable time-of-day alarms and a programmable square-wave output. Address and data are transferred serially via a 2-wire, bidirectional bus. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The data at the end of the month is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either 24-hour or 12-hour format with AM/PM indicator.

The *C-Eye*™ uses the same RTC as the FN controller. The RTC can be accessed via software drivers *rtc1_init()* and *rtc1_rds()*. Refer to sample code in the tern\186\samples\fn\fn_rtc.c. The DS1337 data sheet can be found on TERN CD under **tern_docs\parts** directory and is named **ds1337.pdf**.

It is also possible to configure the real-time clock to raise an output line attached to an external interrupt, at 1/64 second, 1 second, 1 minute, or 1 hour intervals. This can be used in a time-driven application, or the **VOFF** signal can be used to turn on/off the controller using the switching power supply, LM2575. See the sample program, \tern\186\samples\eyec \eye_voff.c.

3.3 *On-board Supervisor with Watchdog Timer*

The MAX691/LTC691 (U6) is a supervisor chip. With it installed, the *C-Eye*™ has several functions: watchdog timer, battery backup, power-on-reset delay, power-supply monitoring, and power-failure warning. These will significantly improve system reliability. The watchdog timer is activated by setting a jumper on J9 of the *C-Eye*™ . 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 = WDI pin of the MAX691) should be arranged such that the WDI pin is accessed at least once every 1.6 seconds. If the J9 jumper is on and the WDI 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 *C-Eye*™ 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/R8820 has an internal watchdog timer. This is disabled by default with **ae_init()**.

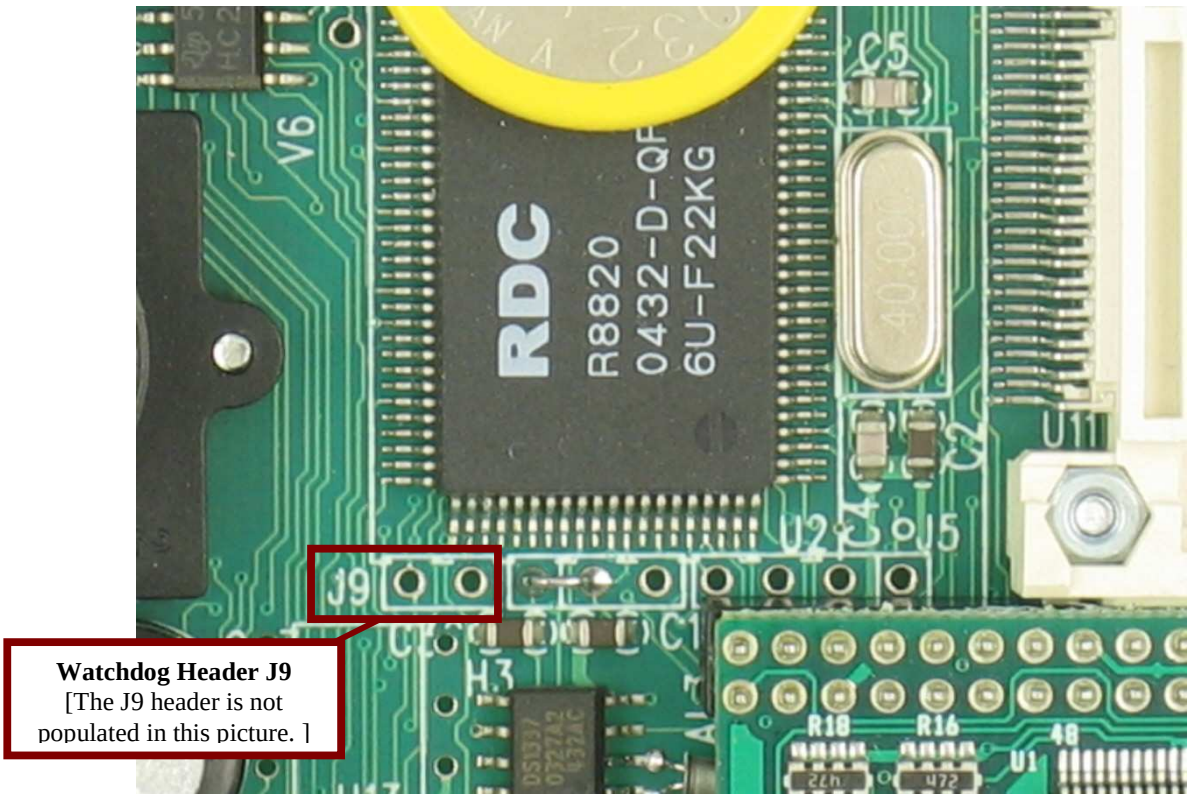


Figure 3.1 Location of watchdog timer enable jumper

3.4 *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 DS1337 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 *Power Fail Monitoring*

The MAX691 and the NMI interrupt line can be used to monitor power supply. User configurable resistors at locations R6 and R7 can be used to create a voltage divider across PFI, where the source voltage is tied to +V. The input threshold for the PFI is rated a 1.3 volts. If PFI drops below this, the MAX691 supervisor will assert /PFO, which will drive the NMI interrupt line. The use may program the NMI ISR to take critical steps before complete power fail. See \tern\186\samples\ae\intx.c for details on interrupts.

3.6 *EEPROM*

A serial EEPROM of 512 bytes (24C04) is installed in U7. The *C-Eye*™ 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, 0x00 – 0x1F. The addresses 0x20 to 0x1FF are for user application.

3.7 *Compact Flash Interface*

By utilizing the compact flash interface on the *C-Eye*™ , users can easily add widely used 50-pin CompactFlash standard mass data storage cards to their embedded application. TERN software supports Linear Block Address mode, 16-bit FAT flash file system. Users can write/read files to/from the CompactFlash card. Users can also transfer files to and from a PC via a Compact Flash card reader. (sandisk.com).

This allows the user to log huge amounts of data from external sources. Files can then be accessed via compact flash reader on a PC.

The **tern\186\samples\ EyeC** directory includes sample code, eyec_cf.c, to show reads and writes of raw data by sector. In addition, **tern\186\samples\fn\fs_cmds1.c** is a simple file system demo with serial port based user interface. Sample project and programs are available under **tern\186\samples\EyeC,** Refer to the eyec.ide which has the demo built and ready for download.

3.8 *Power-off Mode with VOFF signal*

A Low Power version *C-Eye*™ can use the on-board Real-Time-clock DS1337 to control the VOFF signal.

A sample program (c:\tern\186\samples\ eyec \eye_voff.c) demonstrates using the real-time clock to control

timed "on/off" of the controller. When OFF, the controller consumes less than 100 uA current.

To run the sample, you will need an *C-Eye*™ with installed R15(1M ohm) installed, Real-Time Clock, Battery, and Switching Regulator(LM2575).

Hardware Configuration

Install R15=1M ohm on the *C-Eye*™ . The VOFF signal is connected to the LM2575 VOFF pin. The VOFF signal is on J3 pin 1. When the VOFF signal is HIGH (2V+), the LM2575 regulators are disabled (shut down) and very little current (less than 100 uA) is consumed.

When the signal is LOW, the regulators are enabled. The VOFF pin must be connected to a weak pull-up (R15) to +12V. This means, by default, the board will be "shut-down" even with power connected to the inputs to the regulator.

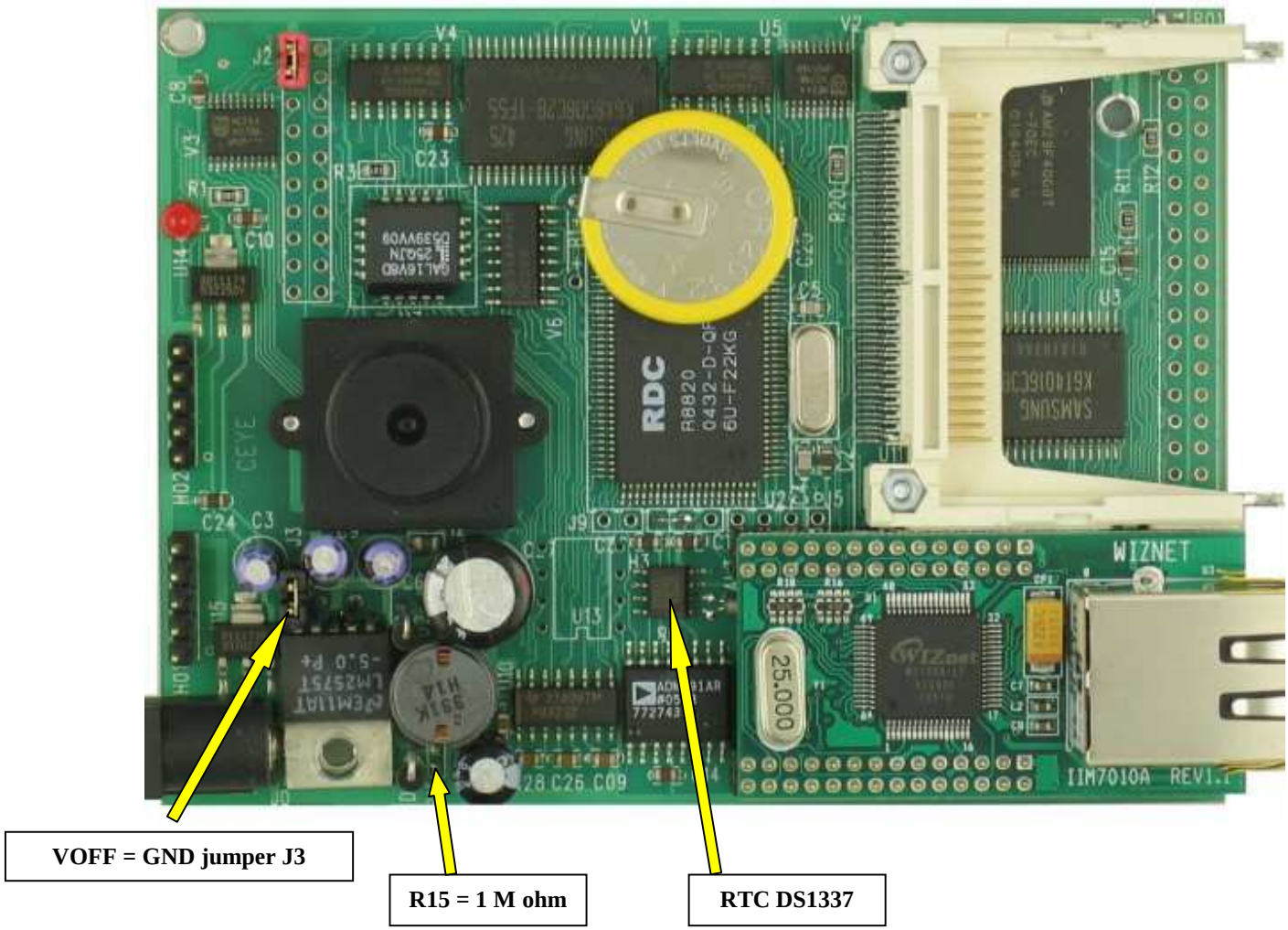
You can control power on or off via VOFF pin with the following ways:

1) Mechanical jumper shorting VOFF (J3.1) to GND (J3.2) forces the regulator to be enabled. This is the default state for debugging.

2) Use on-board real time clock (which is powered by on-board battery).

The real-time clock's alarm (/INTA) can pull a pin LOW. The alarm here acts as a wake-up: at specific times, the RTC will waken your board.

See the sample program (c:\tern\186\samples\eyec\eye_voff.c) on using the real-time clock to control the enabling/disabling of the board at specified real-times.



3.9 *CMOS Image Sensor and Micro Lens*

OmniVision OV7640 CMOS image sensor is used on the *C-Eye*™ . Visit www.ovt.com for details.

A Micro Lens holder is installed covering the image sensor. User can adjust the focus by turning the threaded lens.

The utility CEye_viewer.exe is available under c:\tern\186\samples\ceye. User can run this utility on your Windows based PC and grab images from the *C-Eye*™ via RS232 serial port SER1, or via USB port with CUSB interface board.

A 1 MB image FIFO is on-board to grab and store the image.

Walkthrough on the beginnings of *C-Eye*™ usage can be found in Section 2.3 of this manual.

3.10 *100 MHz BaseT Ethernet*

An WizNet™ Fast Ethernet Module can be installed to provide 100M Base-T network connectivity. This Ethernet module has a hardware LSI TCP/IP stack. It implements TCP/IP, UDP, ICMP and ARP in hardware, supporting internet protocol DLC and MAC. It has 16KB internal transmit and receiving buffer which is mapped into host processor's direct memory. The host can access the buffer via high speed DMA transfers. The hardware Ethernet module releases internet connectivity and protocol processing from the host processor. It supports 4 independent stack connections simultaneously at a 4Mbps protocol processing speed. An RJ45 8-pin connector is on-board for connecting to 10/100 Base-T Ethernet network. A software library is available for Ethernet connectivity. Lines used by the Ethernet device include P14 and /INT4. These should not be user-activated if Ethernet option is installed.

Figure 3.2 WizNet™ Ethernet Module

Chapter 4: Software

Please refer to the Technical Manual on TERN's CD under tern_docs\manuals\software_kit.pdf for details on debugging and programming tools.

Guidelines, awareness, and problems in an interrupt driven environment

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

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

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

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

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

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

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

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

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

These functions retrieve the data for a specified address in memory space. Once again, the **segment** address is shifted left by four bits and added to the **offset** to find the 20-bit address. This address is then output over the address bus, and the hardware component mapped to that address should return either an

8-bit or 16-bit value over the data bus. If there is no component mapped to that address, this function will return random garbage values every time you try to peek into that address.

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

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

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

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

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

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

4.1 AE.LIB

ae.lib is a C library for basic operations. It includes the following modules: AE.OBJ, SER0.OBJ, SER1.OBJ, and AEEE.OBJ. You need to link to AE.LIB in your applications and include the corresponding header files in your source code. 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, from CPU
SER1.H	Internal SER1
AEEE.H	on-board EEPROM

4.2 Functions in AE.OBJ

4.2.1 Initialization

ae_init

This function should be called at the beginning of every program running on *C-Eye*™. 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, on TERN's CD under amd_docs.

- Initialize the upper chip select to support the on-board flash. The CPU registers are configured such that:
  - Address space for the Flash 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. Most pins are set up as default input, except for P29 (used for driving the LED), pins for SER0, and others.

```
outport(0xff78,0xc7bc);     // PDIR1, TxD,RxD,PCS0,PCS1,P29&P22 Output
outport(0xff76,0x2040);     // PIOM1
outport(0xff72,0xec7b);     // PDIR0, A18,A17,PCS6,PCS5, P12 Output
outport(0xff70,0x1000);     // PIOM0
```

- Configure the PPI 82C55 to all inputs. You can reset these by writing to the command register.

```
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 RealTime 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

Only /INT0 is available on Header J2 pin 17 for *C-Eye*™ user free to use.

There are up to six external interrupt sources on the AM186ES, consisting of five maskable interrupt pins (**INT4-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.

TERN provides functions to enable/disable all of the 5 maskable 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);
```

Sample code is also available in the **tern\186\samples\ae** directory, '**intx.c**'.

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 nmi_init(void interrupt far (* nmi_isr)());
```

4.2.3 I/O Initialization

Two ports of 16 I/O pins each are available on AM186ES. 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, initialize the appropriate pins in one of the four available modes. 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 14 of the AMD Am186ES User's Manual. Also see Table 3.2 in this 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. Refer to **'ae_speed.c'** for the fastest possible access.

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 *C-Eye*™ 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 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 for pulse-width modulation with a variable duty cycle. These timers contain two max counters, where the output is high until the counter counts reaches maxcount A before switching and counting 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.

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 if the interrupt bit in the **T0CON/T1CON** is set, with other behavior possible depending on the value specified for the control register. If the interrupt bit is not set, the user can poll the status if the **MC** bit in the timer control registers. Polling the **MC** bit offers a way to monitor timer status without using interrupts.

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, and no I/O pins.

4.2.5 Vision related functions:

void ceye_acq_img(unsigned char stat) ;

Starting hardware image acquisition. It may take 100 ms to complete 640x480x2 pixels.

During this 100 ms time, CPU can do anything, but ceye_en_FIFO();

Change the "unsigned char stat" every time to start image acquisition.

The current state of the hardware imaging can be checked with img_state();

If the return value of the img_state(); is not the same as the "unsigned char stat",

the one frame image acquisition is not complete yet.

unsigned char img_state() ;

return the current state of the imaging hardware.

User can use this returned value to determine if one frame imaging is complete.

void ceye_en_FIFO(void) ;

Enable FIFO to software access.

4.2.6 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. See \tern\186\samplesfn\fn_rtc.c for a sample program. 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 rtc1_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 rtc1_rds
Arguments: char* realTime
Return value: int error_code

This function is slightly different from the rtc_rd function. It places the current value of the real time clock into a character string instead of the TIM structure, making it a more convenient function than rtc_rd.

This function places the current value of the real time clock in the char* realTime. The string has a format of "week year10 year1 month10 month1 day10 day1 hour10 hour1 min10 min1 second10 second1". The **rtc1_rds** function also places a null terminating character at the end of the time string. It is important to note that you must be sure to make the destination character string long enough to hold the real time clock value plus the null character. A destination character string that is too short will result in the data immediately following the character string in memory to be overwritten, causing unknown results.

For example "3040503142500\0" represents Wednesday May 3, 2004 at 02:25.00 pm. There are only two positions for the year, so the user must decide how to determine the hundreds and thousands digit of the year. Here we just assume "04" correlates to the year 2004.

The length of char * realTime must be at least 14 characters, 13 plus one null terminating character.

This function returns 0 on success and returns 1 in case of error, such as the clock failing to respond.

Void rtc1_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 Friday June 6, 2003, 10:55:30 am, the byte array would be initialized to: unsigned char t[14] = { 5, 0, 3, 0, 6, 0, 6, 1, 0, 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 ACTF Boot Utility or from some other address.

4.3 Functions in SER0.OBJ

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

The Am186ES provides two asynchronous serial ports. SER0 and SER1
This section will discuss functions in ser0.h

By default, SER0 is used by the DEBUG kernel (ae86_115.hex) for application download/debugging in STEP 1 and STEP 2. 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 for **SER0** and SER1. Function Argument	Baud Rate
1	110
2	150
3	300
4	600
5	1200
6	2400
7	4800
8	9600
9	19,200 (default)
10	38,400
11	57,600
12	115,200
13	250,000
14	500,000
15	1,250,000
16	28,800

Table 4.1 Baud rate values for ser0 only

After initialization by calling *s0_init()*, SER0 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, *ser0_in_buf* (whose size is specified by the user), will automatically store the receiving serial data stream into the memory by DMA0 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 *serhit0()* and take out the data from the buffer with *getser0()*, 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.

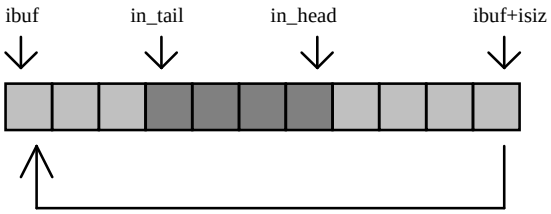


Figure 4.1 Circular ring input buffer

The input buffer (ibuf), buffer size (isiz), and baud rate (baud) are specified by the user with s0_init() with a default mode of 8-bit, 1 stop bit, no parity. After s0_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 Control Register (SP0CT) if necessary, as described in chapter 12 of the Am186ER 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 getser0() 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 serhit0() 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 getser0() to get the serial input data byte by byte using FIFO from the buffer. The in_tail pointer will automatically increment after every getser0() call. It is not necessary to suspend external devices from sending in serial data with /RTS. Only a hardware reset or s0_close() can stop this receiving operation.

For transmission, you can use putser0() to send out a byte, or use putsers0() to transmit a character string. You can put data into the transmit ring buffer, s0_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 putser0() 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.

```
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;
```

S1_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 with the specified parameters. **b** is the baud rate value shown in Table 4.1. Arguments **ibuf** and **isiz** specify the input-data buffer, and **obuf** and **osiz** specify the location and size of the transmit ring buffer.

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

There are a couple different functions used for transmission of data. You can place data within the output buffer manually, incrementing the head and tail buffer pointers appropriately. If you do not call one of the following functions, however, the driver interrupt for the appropriate serial-port will be disabled, which means that no values will be transmitted. This allows you to control when you wish the transmission of data within the outbound buffer to begin. Once the interrupts are enabled, it is dangerous to manipulate the values of the outbound buffer, as well as the values of the buffer pointer. The following functions are shown as 'putsern', where n is the serial port in use. This section applies only to SER0, thus 'putser0'.

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 s0_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 port available on the Am186ER processor has many other features that might be useful for your application. If you are interested in having more control, please read Chapter 12 of the manual for a detailed discussion of other features available to you.

4.4 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 application use.

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

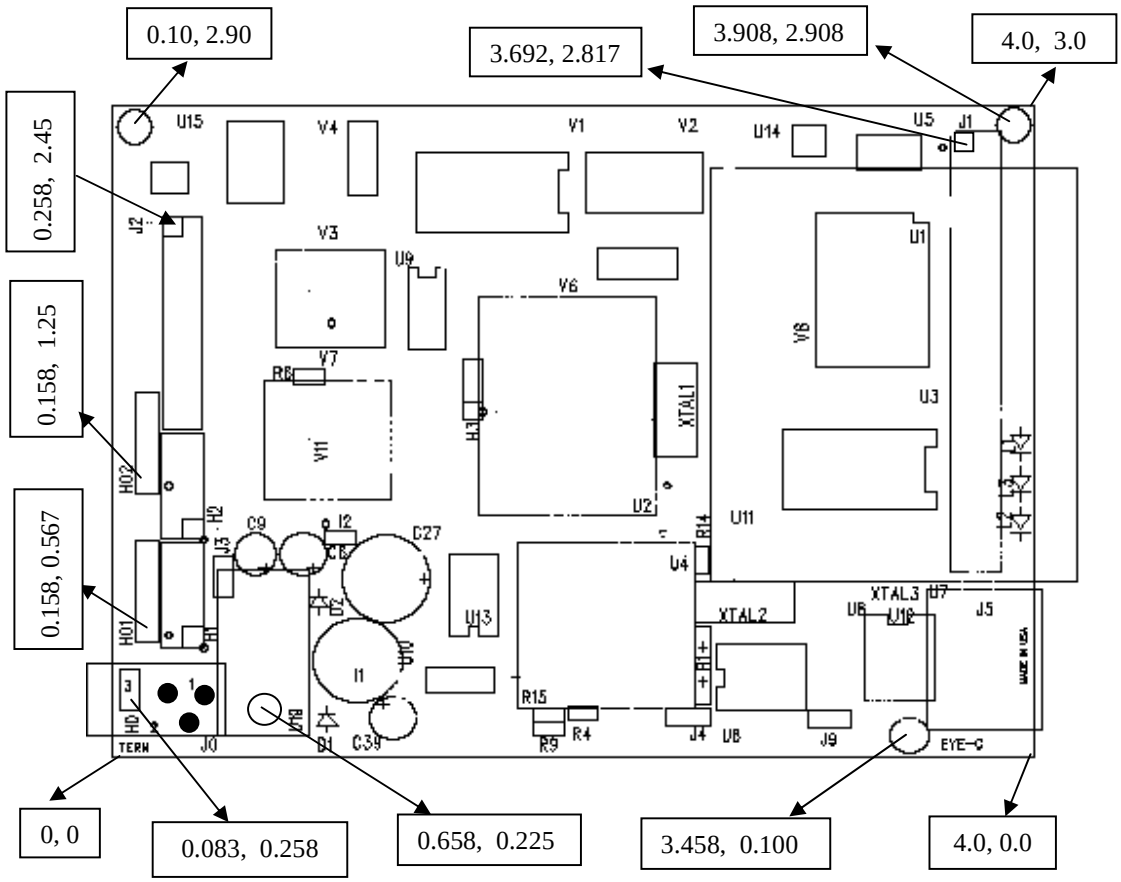
This function is used to write the passed in **dat** to the specified **addr**. The return value is 0 in success.

ee_rd
Arguments: int addr
Return value: int data

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

Appendix A: Layout

C-Eye™ mechanical dimensions. All dimensions are in inches.



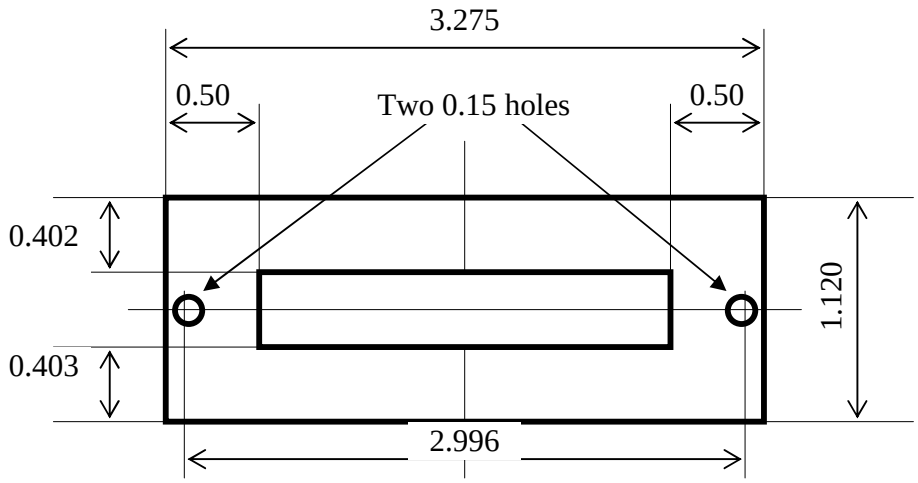
Appendix B: Enclosure Dimensions

End Panel Dimensions: (All in inches)

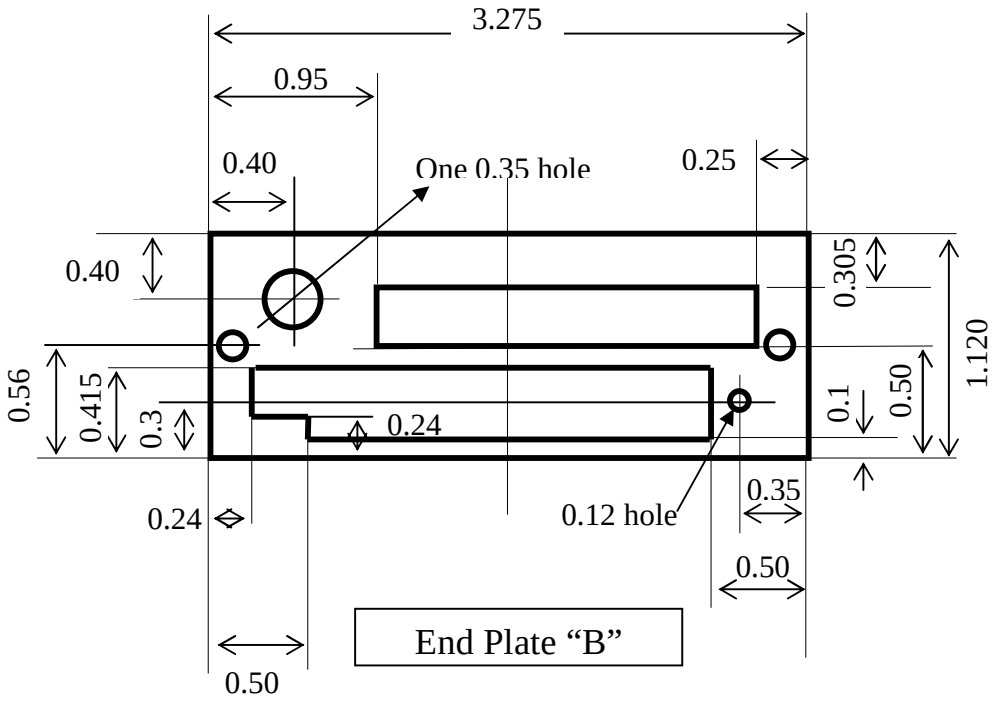
September 7, 2005

Extrusion Technology RS-3010-4020-TRN, cut to a 4" length, two machined end panels and screws, all pieces powder coat painted, overspray allowed.

Josh Murray Program Manager
XTech
781-963-7200, x104
Email: jmurray@xtech-outside.com *Web Site: www.xtech-outside.com



End Plate “A”



End Plate “B”

Appendix C: Debug Cable Diagram

The SER0 RS232 port is a 5x2 (sometimes 5x1) pin-header**.** We need to connect only pin 1, 3, 5, 7, and 9 (outlined in yellow) of the DEBUG cable. Make sure the pin 1 of the cable is connecting to the pin 1 of the SER0 header.

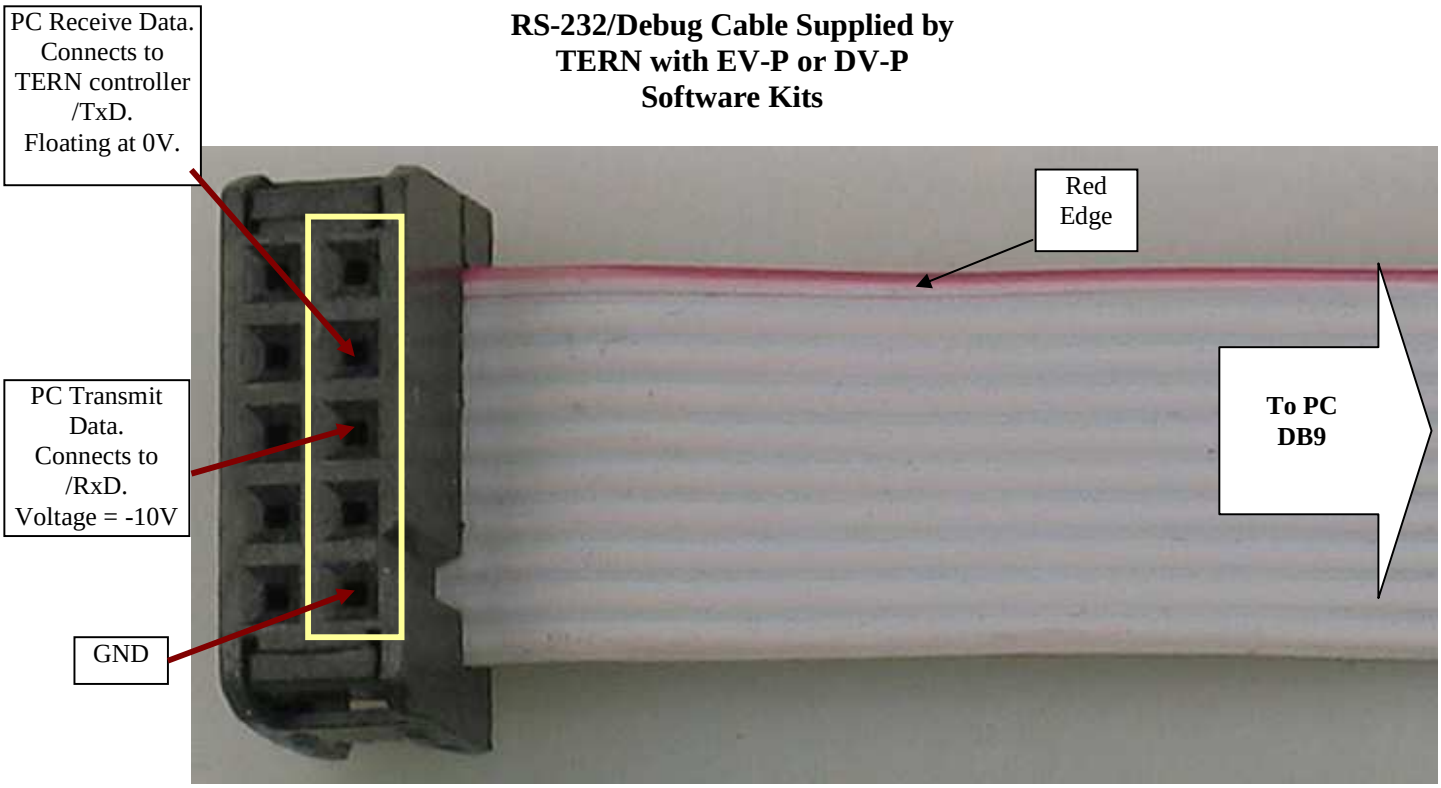
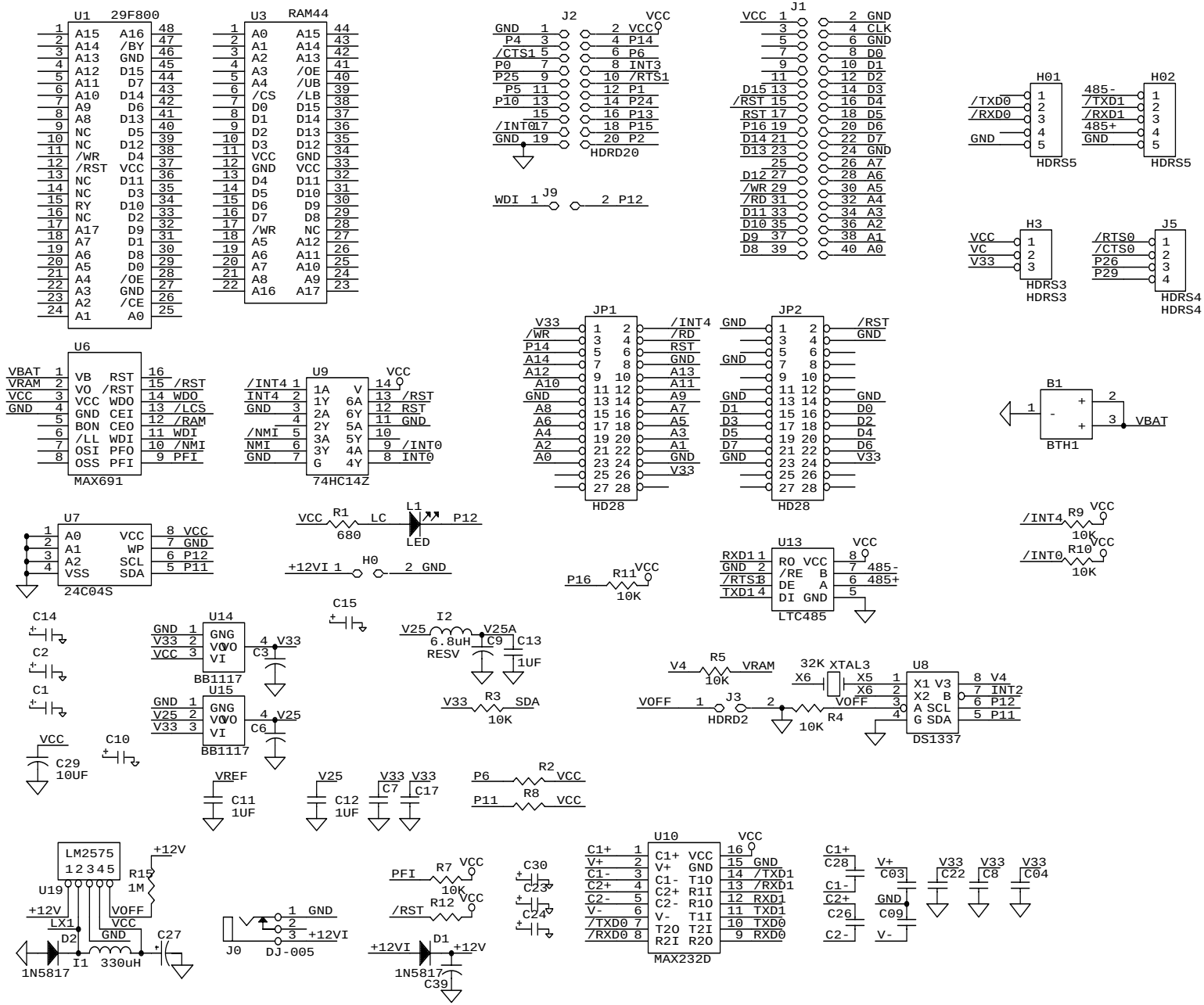


Diagram of Debug Cable (5x2, 10-pin socket shown)



TERN/STE		
Title	C-PROGRAMMABLE EYE	
Size: B	Document Number: CEYE-MAN.SCH	REV:
Date: May 4, 2007	Sheet 1 of 1	