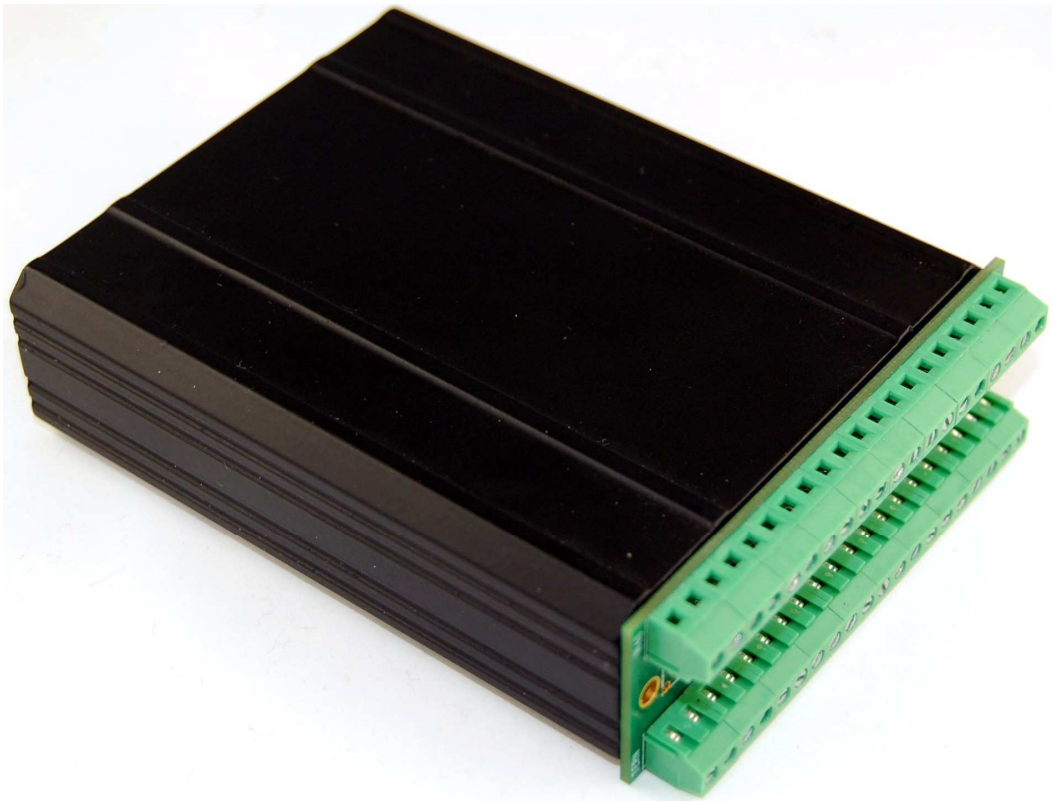
U-Box™

Low cost portable data logger with 16-bit and 24-bit ADCs, DACs, USB



Technical Manual

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

Version 2.0

April 29, 2013

© 1993-2012 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

Boxed in an extruded aluminum enclosure of 4.0x3.2 inches, the ***U-Box***™(***UB***) is designed as a low cost data logger and an industrial embedded controller. It integrates 16-bit ADCs, 24-bit ADCs, 16-bit DACs, CompactFlash and a High Speed USB 1.1/2.0 slave interface.

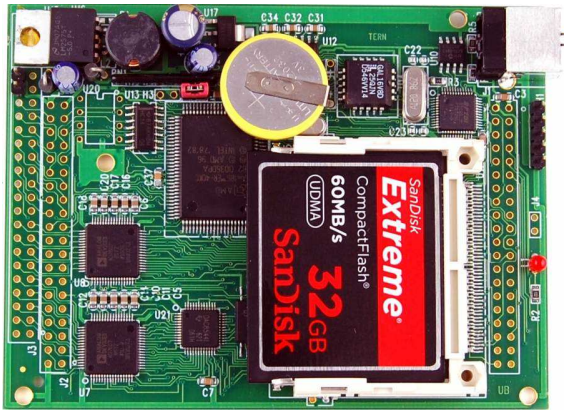
The ***UB*** is based on a high performance C/C++ programmable x186 CPU (AMD186ER or RDC R1100) supporting Flash, SRAM, timer/counters, 20+ PIOs ADC, and DAC.

Up to two true bipolar, simultaneous sampling ADCs(AD7606, 16-bit or AD7607, 14-bit) can be installed. The ADC accepts ±10V or ±5V true bipolar analog signals while sampling at throughput rates up to 200 kSPS for all 8 analog inputs. Each analog input contains second-order antialising filter, sample-and-hold amplifier and clamp protection tolerant up to ±16.5V. With 1M ohm analog input impedance, a 7000V ESD rating, and sustaining up to ±10 mA input current. Four channel 16-bit DAC (DAC8544) can output analog voltage(0-5V). A CompactFlash receptacle allows access to mass storage CompactFlash cards. A real time clock (RTC72423B) can be installed.

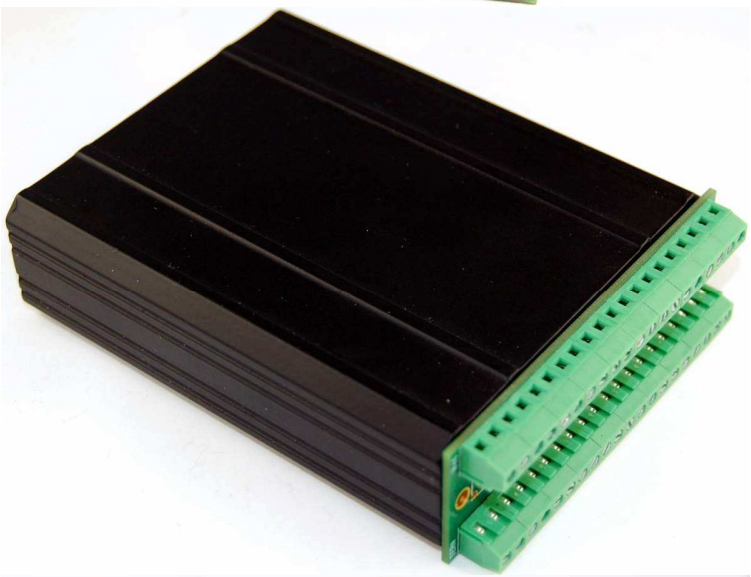
Providing a USB 1.1/2.0 slave USB-B port, the USB stack chip (FT232H, FTDI) handles USB stack processing. No USB specific firmware programming is required. Two types of USB software drivers are available: VCP and D2xx. Data transfer rate can go up to 8 MB/sec with D2xx driver.

User can operate the controller with simple ASCII commands based on TERN ACT firmware.

The ***UB***™ is powered by external 9-30V DC with on-board regulator. Optional Switching Regulator (LM2575) provides a shutdown feature (VOFF). It can enter µA poweroff mode and can be waked-up by an active-low signal either from on-board RTC or external source. Two versions of **UB** are available: the **UB80** is based on 80 MHz R1100, and **UB40** is based on 40 MHz Am186ER. The ***UB***™ supports TERN expansion boards mounted on the top and bottom via J1&J2 headers.




SanDisk
Extreme
CompactFlash
60MB/s
UDMA
SanDisk
32GB



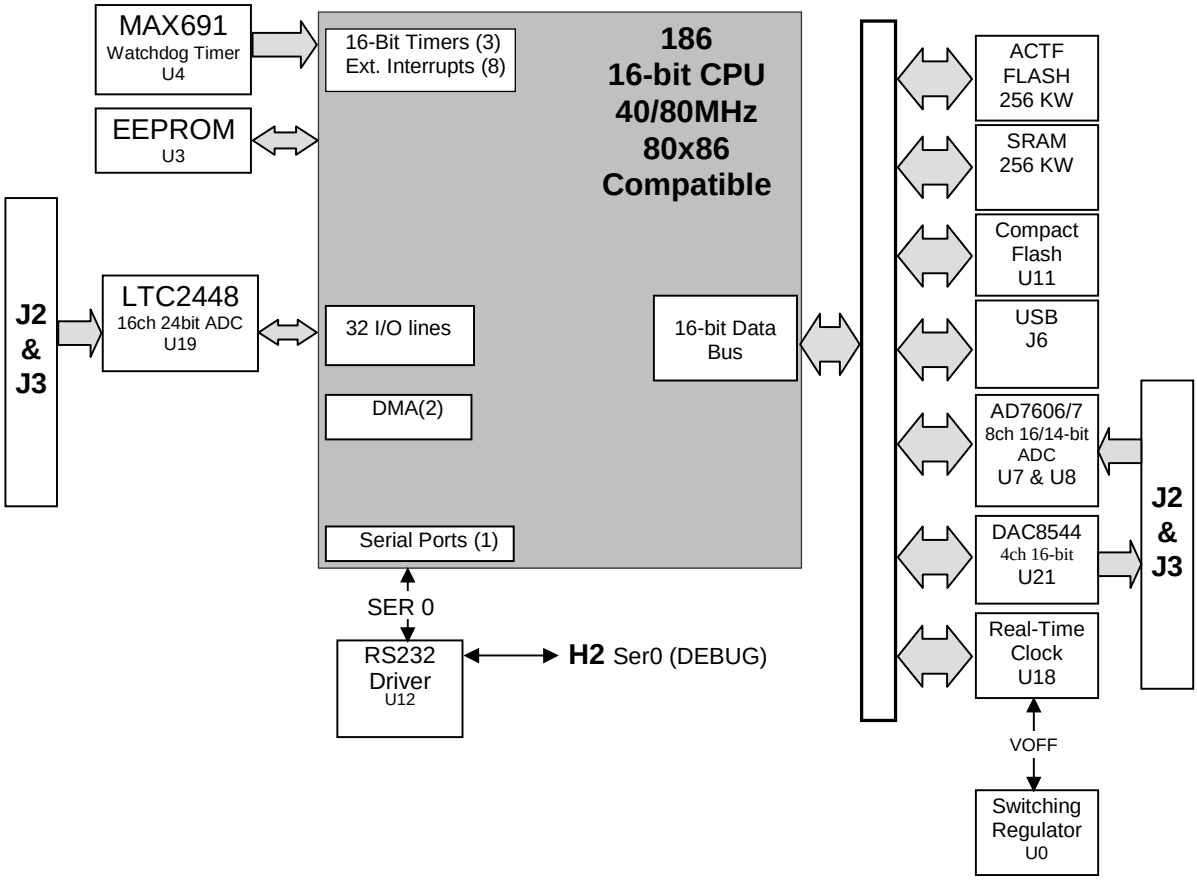


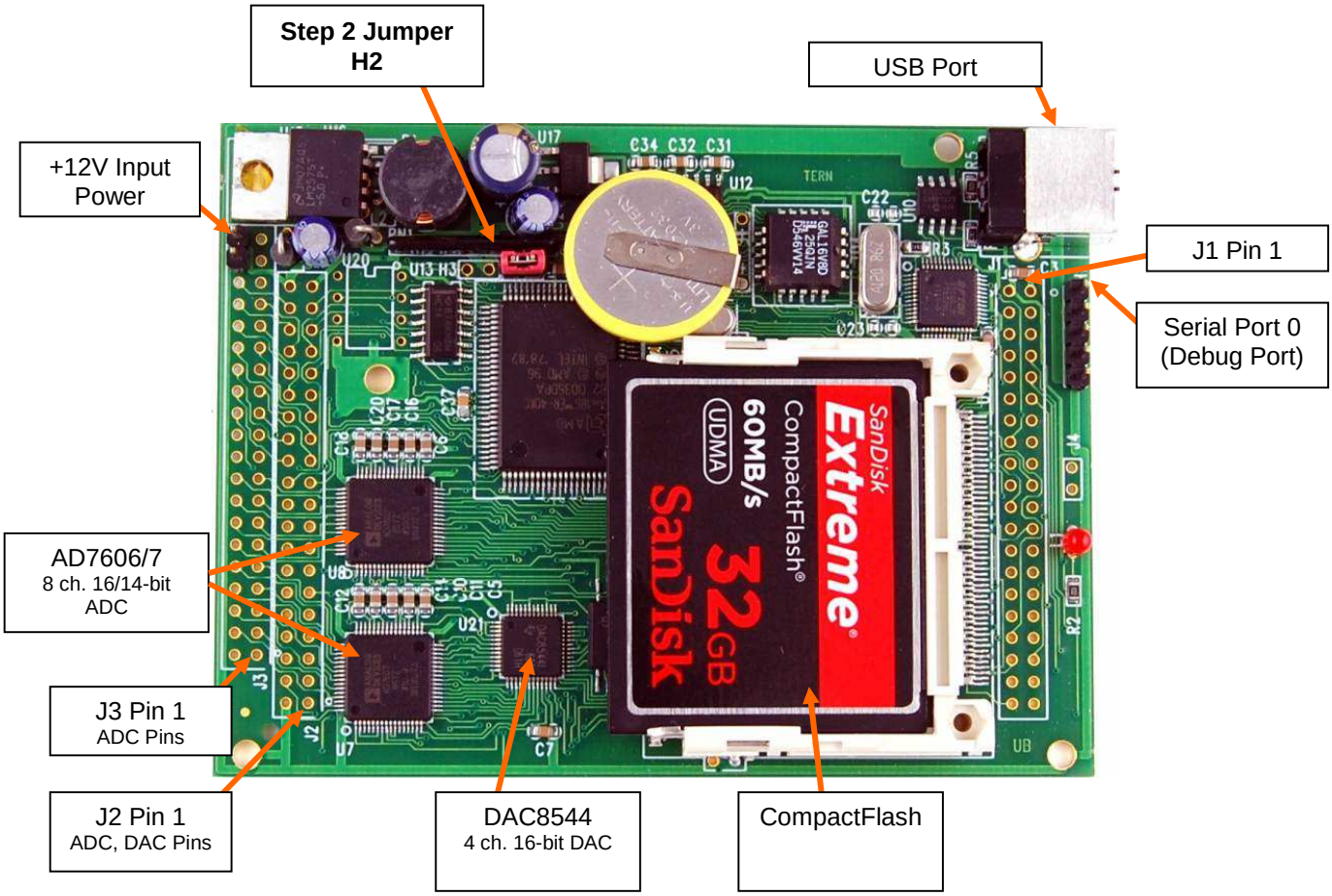
Figure 1.1 Functional block diagram of the UB

Features:

* 3.0 x 3.6", external 9-30V DC power, 200 mA.
* High Speed USB 1.1/2.0 slave interface
* Up to 16 channels of 16-bit ADC, ±10V bipolar inputs
* 16 channels of 24-bit ADCs, 4 16-bit DAC (0-5V) outputs
* CompactFlash with FAT file system support
* Complete C/C++ programmable environment
* One RS-232 serial port, x186 CPU, Flash, SRAM, RTC, PIOs
* Supports TERN expansion boards: GR4, GR16, B48
* Aluminum box with screw terminals

1.2 Physical Description

Below shows the physical description of the UB.



1.3 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 (H2 pins 1&2). If the STEP 2 jumper is installed, the "jump address" located in the on-board serial EEPROM will be read 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 UB for communication with the Paradigm C/C++ IDE for downloading and debugging applications. The following diagrams show the procedure for programming the UB. Steps include preparing the UB for debugging, debugging the UB, standalone field test, and production.

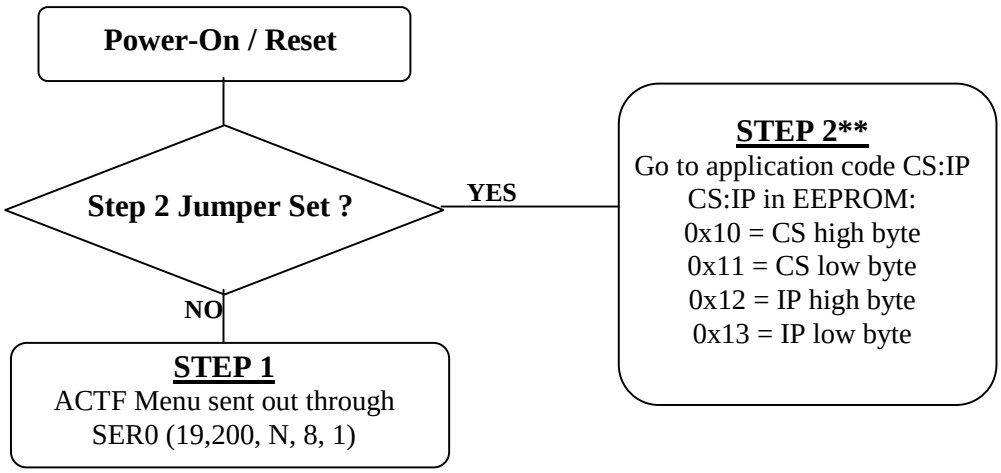


Figure 1.2 Flow Chart of ACTF Operation

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.

***** U-Box does not run in STEP 2. Stand alone testing can only be performed in STEP3 where the code is programmed into non-volatile FLASH.***

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 UB (SER0, H2) to PC (COMx) via serial debug cable provided with the EV-P/DV-P. Using the Windows "Hyper Terminal", create a serial link based on 19,200, 8 bits, 1 stop, no parity.
- Power on the UB WITHOUT the STEP 2 jumper installed (J1 pins 1 & 2). 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\re\re40_115 (re80_115.hex for 80MHz). This is the debug kernel. Use the "GFA000 <enter>" command to set jump address and execute the debug kernel. The LED will blink twice and remain on.
- Set the STEP 2 jumper (J1 pins 1 & 2). The UB is now ready to communicate with the Paradigm C/C++ IDE for debugging and application development.

Step 1: Debugging:

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

****Step 2: Standalone Field Test:**

- U-Box does not work in STEP2. Proceed to STEP3 for standalone field test.

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 the ACTF technical manual, found in the \tern_docs\manuals directory. Here you will find details on generating an ACTF downloadable HEX file based your application.
- Remove the STEP 2 jumper and create serial link using Hyper Terminal (19,200, N, 8, 1). At power-on/reset, you will see the ACTF menu at Hyper Terminal. Use the "D <enter>" command to initiate download process. Select Transfers -> Send File, and select \tern\186\rom\re\l_29f40r.hex.
- This file will erase the flash and prepare the flash to accept ACTF downloadable application HEX file. Use the "G04000 <enter>" command to run script. Flash will be ready for application.
- Select Transfer -> Send File to select your ACTF downloadable application HEX file. Upon completion, use the "G80000 <enter>" command to execute application. This command also sets the jump address to point your application in flash. Set STEP 2 jumper (H2 pins 1&2). At power-on/reset application will execute.

There is no ROM socket on the UB. 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 be installed for every production-version board.

1.4 Minimum Requirements for UB System Development

Minimum Hardware Requirements

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

Minimum Software Requirements

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

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

Chapter 2: Installation

2.1 Software Installation

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

2.2 Hardware Installation

Overview

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

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

2.2.1 Connecting the UB to the PC

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

The UB communicates through SER0 by default. Install the 5x2 IDC connector on the SER0 H2 5x2 pin header. ***IMPORTANT:*** *Note that the* ***red*** *side of the cable must point to pin 1 of the H1 header.* The DB9 connector should be connected to one of your PC's COM Ports (COM1 or COM2).

2.2.2 Powering-on the UB

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

Connect +9-12V DC to the input power pin J3.38 = 9VI and ground to J3.40 = GND. A power jack adapter is included with the TERN EV-P/DV-P kit. It can be used to connect the output of the power jack adapter and the UB. Note that the output of the power jack adapter is center negative.

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

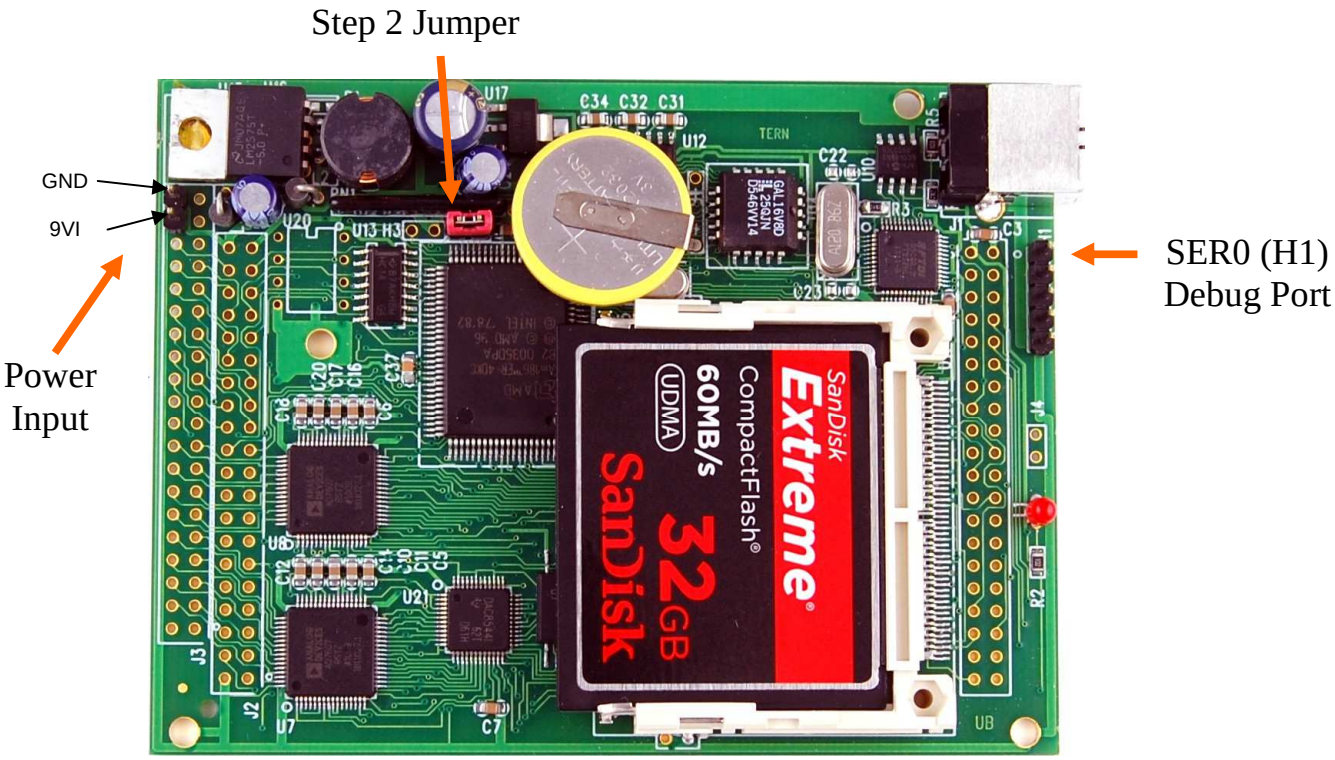


Figure 2.1 UB Power Input and Step2 Jumper

Figure 2.2 UB with Power Jack Adaptor

Chapter 3: Hardware

3.1 *Am186ER AND RDC R1100*

The UB is compatible with two different CPUs. Both offer and support the same on-board peripherals as well as the on the CPU itself, aside from a few differences. The Am186ER, from AMD, uses times-four crystal frequency, while the R1100, from RDC, uses times-eight. The UB uses a 10MHz system clock, giving the Am186ER a CPU clock of 40MHz and the R1100 a CPU clock of 80MHz. Both CPUs operate at +3.3V, with lines +5V tolerant. The RDC 1100 supports the same 80C188 microprocessor instruction set as the Am186ER, yet uses an internal RISC core architecture.

3.2 *Am186ER – Introduction*

The Am186ER is based on the industry-standard x86 architecture. The Am186ER controllers are higher-performance, more integrated versions of the 80C188 microprocessors. In addition, the Am186ER has new peripherals. The on-chip system interface logic can minimize total system cost. The Am186ER has one asynchronous serial port, one synchronous serial port, 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.

In addition, the Am186ER has 32KB of internal volatile RAM. This provides the user with access to high speed zero wait-state memory. In some instances, users can operate the UB without external SRAM, relying only on the Am186ER's internal RAM.

3.3 *RDC R1100 – Introduction*

The RDC 1100 is based on RISC internal architecture, yet still supports the same 80C188 microprocessor instruction set. It provides faster operation than the Am186ER, allowing it to operate at up to 80MHZ, based a 10MHz system clock and times-eight crystal operation. The RDC R1100 does not offer internal RAM like the Am186ER, so external SRAM is mandatory if using the RDC R1100.

3.4 *Am186ER – Features*

3.4.1 Clock

Due to its integrated clock generation circuitry, the Am186ER microcontroller allows the use of a times-four crystal frequency. The design achieves 40 MHz CPU operation, while using a 10 MHz crystal.

The R1100 offers times-eight crystal frequency, achieving 80MHz operation based on a 10MHz crystal.

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

CLKOUTA remains active during reset and bus hold conditions. The UB 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.

3.4.2 External Interrupts and Schmitt Trigger Input Buffer

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

/INT0, J2 pin 8, is free for application use
/INT1, J2 pin 6, is free for application use
INT2, J2 pin 1 (P31), is free for application use
/INT3, J2 pin 21, is free for application use
/INT4, J2 pin 33, is free for application use
NMI, J2 pin 7, is free for application use

Five external interrupt inputs, /INT0-1, /INT3-4, and NMI are buffered by Schmitt-trigger inverters (U9, 74HC14) in order to increase noise immunity and transform slowly changing input signals to fast changing and jitter-free signals. As a result of this buffering, these pins are capable of only acting as input.

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

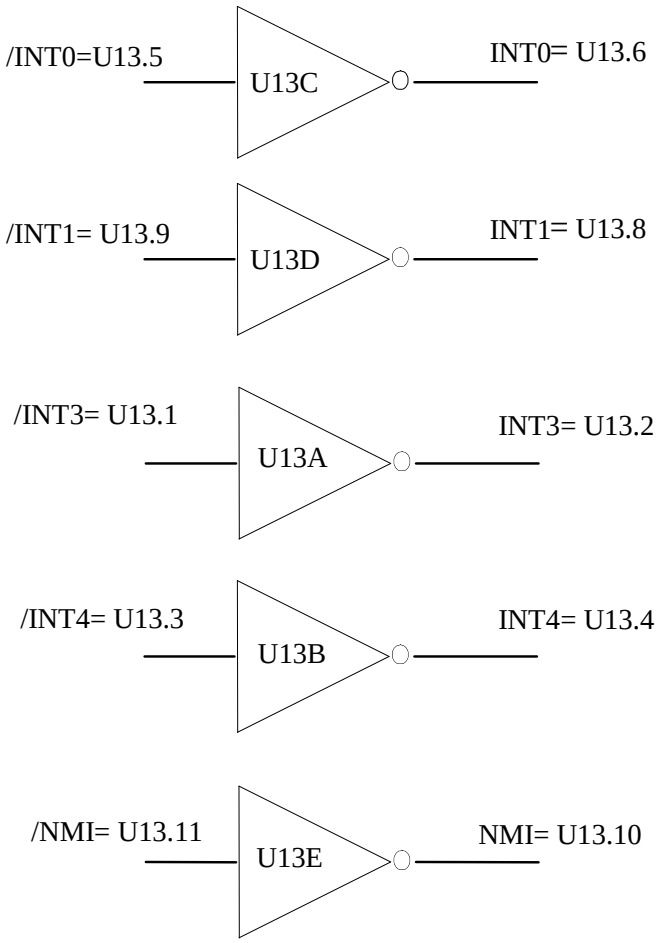


Figure 3.1 External interrupt inputs

The UB uses vector interrupt functions to respond to external interrupts. Refer to the Am186ER User's manual for information about interrupt vectors.

3.4.3 Asynchronous Serial Port

The Am186ER and R1100 CPU has one asynchronous serial channel. It supports 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 (Am186ER ONLY)
- Transmit and receive interrupts
- Maximum baud rate of 1/16 of the CPU clock speed
- Independent baud rate generators

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

Note that while the Am186ER supports DMA transfers to and from its asynchronous serial port, the R1100 does not. Despite this difference, the TERN software drivers for the asynchronous serial port support both CPUs.

3.4.4 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	= Used by AD7606 U7 and U8
Timer0 input	= P11	= Used by LTC2448 U19
Timer1 output	= P1	= J2 pin 29
Timer1 input	= P0	= J3 pin 4

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.

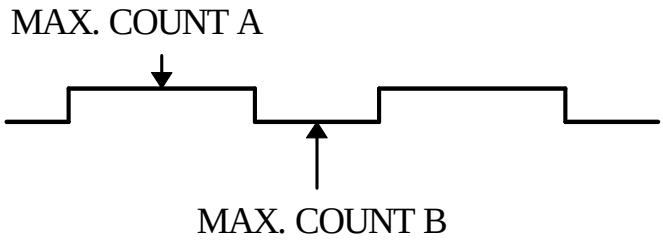
Timer 0 output, P1, is used as the clock input for the AD7606 and is not available for application use.

The maximum rate at which each timer can operate is 10 MHz for the Am186ER and 20MHz for the R1100, 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.

3.4.5 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.



3.4.6 Power-save Mode

The UB is an ideal core module for low power consumption applications. The power-save mode of the Am186ER 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.

3.5 *Am186ER PIO lines*

The Am186ER 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*	*UB Pin No.*	*UB Initial after ae_init(); function call*
P0	Timer1 in	Input with pull-up	J2 pin 4	Input with pull-up
P1	Timer1 out	Input with pull-down	J2 pin 29	Input with pull-up
P2	/PCS6/A2	Input with pull-up	U7 & U8 pin 4	/PCS6
P3	/PCS5/A1	Input with pull-up	U7 & U8 pin 5	/PCS5
P4	DT/R	Normal	J3 pin 3, H2 pin 1	Input with pull-up: Step 2
P5	/DEN/DS	Normal	U8 pin 15	Input with pull-up
P6	SRDY	Normal	U7 & U8 pin 8	Input with external pull-up
P7	A17	Normal	N/A	A17
P8	A18	Normal	N/A	A18
P9	A19	Normal	U9.21	Input with pull-up
P10	Timer0 out	Input with pull-down	U7 & U8 pin 3	Input with pull-down
P11	Timer0 in	Input with pull-up	U19 pin 2	Input with pull-up
P12	DRQ0	Input with pull-up	J1 pin 26	Output
P13	DRQ1	Input with pull-up	U13 pin 12	Input with pull-up
P14	/MCS0	Input with pull-up	J2 pin 37	Input with pull-up
P15	/MCS1	Input with pull-up	U7 & U8 pin 11	Input with pull-up
P16	/PCS0	Input with pull-up	J1 pin 19	/PCS0
P17	/PCS1	Input with pull-up	U14 pin 2	/PCS1 for U14 PAL
P18	/PCS2	Input with pull-up	J2 pin 31	Input with pull-up
P19	/PCS3	Input with pull-up	J2 pin 22	Input with pull-up
P20	SCLK	Input with pull-up	J2.5, U19.38	Input with pull-up
P21	SDATA	Input with pull-up	J2.3, U19.37	Input with pull-up
P22	SDEN0	Input with pull-down	U3.6	Output
P23	SDEN1	Input with pull-down	U19.36	Input with pull-down
P24	/MCS2	Input with pull-up	U7 pin 15	Input with pull-up
P25	/MCS3	Input with pull-up	J19 pin 34	Input with pull-up
P26	UZI	Input with pull-up	U21 pin 27	Input with pull-up*
P27	TxD	Input with pull-up	N/A	TxD0
P28	RxD	Input with pull-up	N/A	RxD0
P29	S6/CLKSEL1	Input with pull-up	N/A	Output
P30	INT4	Input with pull-up	/INT4 = J2.33	Input with pull-up
P31	INT2	Input with pull-up	J2 pin 1	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

UB 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 **re_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 UB system for on-board components. 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	AD7606 U7&U8
P3	/PCS5	AD7606 U7&U8
P4	/DT	Step Two jumper
P5	/DEN/DS	AD7606 U8
P6	SRDY	AD7606 U7&U8
P7	A17	Upper address line – **Never use by application**
P8	A18	Upper address line – **Never use by application**

Signal	Pin	Function
P9	A19	USB FT232
P10	Timer0 out	AD7606 U7&U8
P11	Timer0 in	LTC2448 U19
P13	DRQ1	USB FT232
P15	/MCS1	AD7606 U7&U8
P17	/PCS1	PAL U14
P20**	SCLK	Synchronous Clock for U19
P21**	SDAT	Serial Interface for U19
P22	SDEN0	EEPROM
P23	SDEN1	LTC2448 U19
P24	/MCS2	AD7606 U7
P26	UZI	DA8544
P27	TxD0	Serial Port 0
P28	RxD0	Serial Port 0
P26*	/CLKSEL2	Used at power-up/reset to determine system clock multiplier
P29*	/CLKSEL1	Reserved for EEPROM, LED and Watchdog timer

Important Notes:

* The Am186ER CPU uses the P26 and the P29 lines to determine the system clock multiplier at power-up or reset. The CPU has internal pull-ups on these lines to select the default multiplier of four-times (AMD) or eight-times (RDC). It is critical that the user allow these lines to remain high during power-up or reset. Failure to do so will result in undesirable operation. In addition, P6 must also be allowed high duing power-on or reset.
** The SCLK and SDAT lines are the synchronous serial port on the Am186ER. The AD2448 ADC uses these lines. The user is free to use the SCLK and SDAT lines for their application only if the ADCs is disabled first. This is needed so as not to have more than one device trying to occupy the SDAT line simultaneously.

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

3.6 *I/O Mapped Devices*

3.6.1 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 100 ns for both CPUs, while the CPU clock is 25ns for the Am186ER and 12.5ns for the R1100. Details regarding this can be found in the Software chapter, and in the Am186ER 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 may need to be attached to I/O pins directly.

For details regarding the chip select unit, please see Chapter 5 of the Am186ER 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		U14 PAL
0x0100-0x011f	/RTC	U18	RTC
0x0120-0x012f	/AD	U8	AD7606
0x0130-0x013f	/AD1	U7	AD7606
0x0140-0x014f	/RDU	U9	USB
0x0150-0x015f	/DA	U21	DA8544
0x0160-0x016f	/CV	U8 & U9	AD7606
0x01e0-0x01ff	/CF	U11	CF
0x0200-0x02ff	/PCS2	J2 pin 31 = P18	USER
0x0300-0x03ff	/PCS3	J2 pin 22 = P19	USER
0x0400-0x04ff	/PCS4		Reserved
0x0500-0x05ff	/PCS5		P3
0x0600-0x06ff	/PCS6		P2

*PCS0 may be used for other TERN peripheral boards, such as FC-0, P50, P100, MM-A.

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

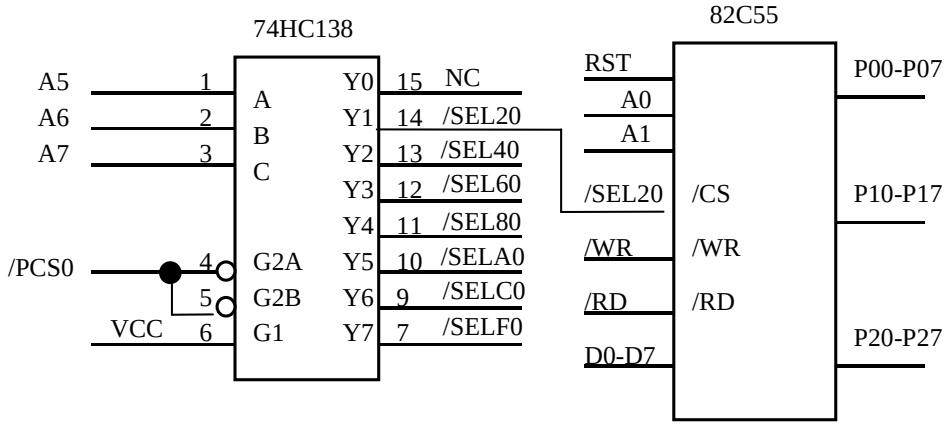


Figure 3.2 Interface the UB to external I/O devices

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 0x20 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.

3.6.2 Real-time Clock RTC72423

If installed, the real-time clock RTC72423 (EPSON, U18) is mapped in the I/O address space 0x0100. It must be backed up with a lithium coin battery. The RTC is accessed via software drivers ***rtc_init()*** or ***rtc_rd()***.

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

example of a program showing a similar application can be found in **`tern\v25\samples\ve\poweroff.c`**.

3.7 *Other Devices*

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

3.7.1 On-board Supervisor with Watchdog Timer

The MAX691/LTC691 (U6) is a supervisor chip. With it installed, the UB has several functions: watchdog timer, battery backup, power-on-reset delay, power-supply monitoring, and power-failure warning. These will significantly improve system reliability.

3.7.2 Watchdog Timer

The watchdog timer is activated by setting a jumper on J4 of the UB. 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 P29 = WDI pin of the MAX691) should be arranged such that the WDI pin is accessed at least once every 1.6 seconds. If the J4 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 UB 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 J4 jumper is off, which disables the watchdog timer.

The Am186ER has an internal watchdog timer. This is disabled by default with **ae_init()**.

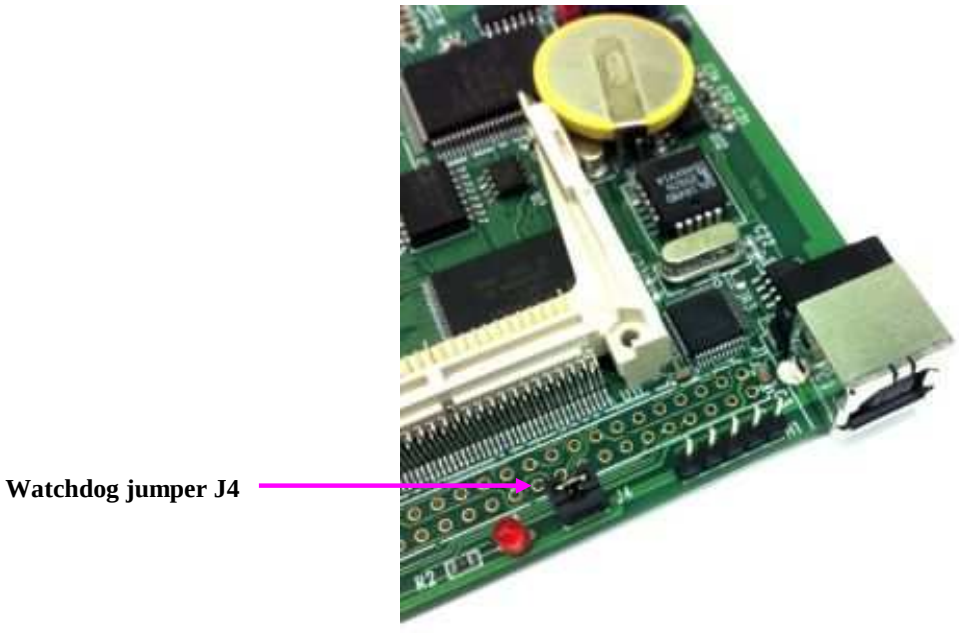


Figure 3.3 Location of watchdog timer enable jumper

3.7.3 Battery Backup Protection

The backup battery 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 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.

3.7.4 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.

3.7.5 EEPROM

A serial EEPROM of 512 bytes (24C04) is installed in U3. The UB uses the P22=SCL (serial clock) and P29=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.

3.7.6 24-bit, 16-channel ADC(LTC2448)

A 24-bit LTC2448 sigma-delta ADC can be installed in U19. The LTC2448 chip offers 8 ch. differential or 16 ch. single-ended input channels. Variable speed/resolution settings can be configured. A peak single-channel output rate of 5 KHz can be achieved. The LTC2448 switches the analog input to a 2 pf capacitor at 1.8MHz with an equivalent input resistance of 110K ohm. The ADC works well directly with strain gages, current shunts, RTDs, resistive sensors, and 4-20mA current loop sensors. The ADC can also work well directly with thermocouples in the differential mode. The TLC2448 can be referenced by VCC, or a 5 or 2.5V precision reference with a internal temperature sensor can be installed in U20. Inputs are routed directly to header J2 and J3 on pins E00-E15 (see schematics).

Refer to the sample program \tern\186\samples\ub\ub_ad24.c.

3.7.7 AD7606 16-bit parallel and AD7607 14-bit high speed ADC

The BE supports a maximum of two (U7 & U8) AD7606 16-bit ADC or AD7607 14-bit ADC. Both types of ADC can accept ±10V or ±5V true bipolar analog signals while sampling at throughput rates up to 200 kSPS for all 8 analog inputs. Each analog input contains second-order antialansing filter, sample-and-hold amplifier and clamp protection tolerant up to ±16.5V. With 1M ohm analog input impedance, a 7000V ESD rating, and sustaining up to ±10 mA input current, the analog inputs are designed to survive in a rough industrial environment. The ***UB*** allows *simultaneous* sampling on all eight analog inputs. Using the 16-bit parallel DMA interface, the UB can transfer 8 channels of 16-bit (AD7606) or 14-bit (AD7607) data into the SRAM or CompactFlash with minimal software overhead.

See sample program \tern\186\samples\ub\ub_ad16.c for details on reading the ADC. The sample program is also included in the pre-built sample project: \tern\186\samples\ub\ub.ide.

3.7.8 DAC8544, 16-bit parallel high speed DAC

The DAC8544 (U21) is a low-power, quad-channel, 16-bit, voltage output DAC. Its on-chip precision output amplifier allows rail-to-rail (0-5V) voltage swing to be achieved at the output.

A sample program ub_da.c may be found in the c:\tern\186\samples\ub directory.

3.7.9 Compact Flash Interface

By utilizing the compact flash interface on the **UB**, users can easily add widely used 50-pin CF standard mass data storage cards to their embedded application via RS232, TTL I2C, or parallel interface. TERN software supports Linear Block Address mode, 16-bit FAT flash file system, RS-232, TTL I2C or parallel communication. Users can write/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\ub** directory includes sample code, **ub_cf.c**, to show reads and writes of raw data by sector. In addition, **tern\186\samples\ub\ub_fs_cmd_usb.c** is a simple file system demo with usb based user interface. Refer to **ub.pdl** which has the demo built and ready for download.

3.7.10 High-Speed USB 1.1/2.0 Slave FT232H

FTDI's TF232H chip provids a USB 1.1/2.0 slave USB-B port. The FT232H handles USB stack processing and no USB specific firmware programming is required. The FT232H is configured to interface with the UB CPU using the FT245 style asynchronous FIFO interface. When configured in this mode, the pins on the FT232H connect directly to the databus of the CPU and is selected using an I/O mapped chip select.
The FT232H uses two types of USB software drivers: VCP or D2xx. When the FT232H is configured to use the VCP driver, the USB connection creates a virtual COM port on the PC. This allows the UB to communicate to a terminal program as if it were connected via RS232. The sample program \tern\186\samples\ub\ub_usb_vcp.c shows how to use the USB port as a virtual COM port.
By default, the FT232H is configured to use the D2xx driver. The data transfer rate can go as high as 8 MB/sec with the D2xx driver. The D2xx driver provides a dymanic linked library that the user can use in developing a Windows application interface (see FTDI.com for information on using the D2xx driver). See the appendix at the end of this manual for installing and configuring the FTDI drivers.
Note: USB does not provide power to the U-Box. Removing the USB connection during run time may reset the U-Box. USB connection should be maintained during run time.

3.8 *Headers and Connectors*

There are three primary connectors on the UB which provide expansion, J1, J2 and J3. The tables bellow summarize the signals available on each connector. Most of the signals on J1 are routed directory to the CPU with no buffer protection. This makes the CPU (including SRAM, Flash, and other devices tied to the A/D bus) vulnerable to damage from out of range voltages. The user is therefore responsible for ensuring that out of range voltages are not applied to sensitive lines.

These signals are +3.3V signals, but are +5V tolerant. Any voltages above +5V will certainly damage the board.

Refer to the schematic at the end of this technical manual for complete signal definitions for all headers and connectors.

3.8.1 Expansion Headers J1 and J2

There are two headers for UB expansion. Most signals are directly routed to the Am186ER processor. J1 signals are primarialy address and data lines from the CPU. These can be used for device expansion. J2 contains a mix of processor IO lines and analog IO signals.

J2 Signal			
GND	40	39	VCC
E15	38	37	P14
E13	36	35	E14
E12	34	33	/INT4
E11	32	31	P18
E10	30	29	P1
E9	28	27	E8
E7	26	25	E6
E5	24	23	E4
P19	22	21	/INT3
E3	20	19	E2
E1	18	17	E0
AI2	16	15	AI1
AI4	14	13	AI3
DA4	12	11	DA3
DA2	10	9	DA1
/INT0	8	7	/NMI
/INT1	6	5	SCLK
P0	4	3	SDAT
GND	2	1	P31

J1 Signal			
VCC	1	2	GND
	3	4	CLK
	5	6	GND
	7	8	D0
	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	P12
D12	27	28	A7
/WR	29	30	A6
/RD	31	32	A5
D11	33	34	A4
D10	35	36	A3
D9	37	38	A2
D8	39	40	A1

Table 3.3 Signals for J1, J2 expansion ports

3.8.2 Analog I/O Header J3

J3 provides access to all of the analog IO on the UB. Some singals on J3 are duplicated on J2. Signals E00-E15 map to the 24-bit ADC LTC2448 (U19). Signals AI1-AI8 map to AD7606/7 (U8) and BI1-BI7 to AD7606/7 (U7).

J3 Signal			
GND	40	39	
9VI	38	37	
E15	36	35	E14
E13	34	33	E12
E11	32	31	E10
E9	30	29	E8
E7	28	27	E6
E5	26	25	E4
E3	24	23	E2
E1	22	21	E0
	20	19	
AI2	18	17	AI1
AI4	16	15	AI3
AI6	14	13	AI5
AI8	12	11	AI7
BI2	10	9	BI1
BI4	8	7	BI3
BI6	6	5	BI5
BI8	4	3	BI7
	2	1	

Table 3.4 Signals for J3 expansion ports

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.

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 RE.LIB

RE.LIB is a C library for basic UB operations. It includes the following modules: AE.OBJ, SER0.OBJ, SER1R.OBJ, and AEEE.OBJ. You need to link to RE.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, Watchdog
SER0.H	Internal serial port 0, from CPU
AEEE.H	on-board EEPROM
TB.H	RTC

Not all functions in the above modules will apply to the UB. For example, “ae.h” was originally created for the A-Engine. Therefore, “ae.h” will include routines for the TLC2543 (for example), not installed on the UB. The user will need to include the header file “re.h” to provide routines for the UB devices. Although “ae.h” was created for a different controller, it will still be needed for a variety of routines used by the UB, such as timers, interrupts, and others. Refer to the actual header file itself to determine which is needed for a certain application.

4.2 Functions in AE.OBJ

4.2.1 UB Initialization

ae_init

This function should be called at the beginning of every program running on UB 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 Am186ER Microcontroller User's manual.

- 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 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 six external interrupt sources on the UB, 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. For a detailed discussion involving the ICUs, the user should refer to Chapter 9 of the AMD Am186ER Microcontroller User's Manual - or the R1100 user's manual, both available on the CD under the **amd_docs** directory. (**Remember, DMA channels to and from the serial port not available on the R1100.)**

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);
```

See Chapter 9 of Am186ER technical manual (tern_docs) for additional details. 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 the UB. 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. 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. (Example, if using the ADS8344, P15 is needed as the chip select, so it will be unavailable for any other purpose while the ADC is being used).

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 Am186ER 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 **'re_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 UB 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 10 of the AMD AM186ER User's Manual.

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.

The specific behavior that you might want to implement is described in detail in chapter 10 of the AMD AM186ER 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 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 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 library ***tb.lib*** must be included into the application project to use the real-time clock. See \tern\186\samples\ub\ub_rtc_init.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 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 tb_rtc_rd
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 **tb_rtc_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 tb_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 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 Am186ER only provides one asynchronous serial port. The UB comes standard with the SC26C92, providing two additional asynchronous ports. The serial port on the Am186ER will be called SER0, and the two UARTs from the SC26C92 will be referred to as SER1 and SER2.

This section will discuss functions in **ser0.h** only, as SER0 pertains to the Am186ER.

By default, SER0 is used by the DEBUG kernel (re80_115.hex) for application download/debugging in STEP 1 and STEP 2. **The following examples that will be used, show functions for SER0, but since it is used by the debugger, you cannot directly debug SER0.** This section will describe its operation and software drivers. The following section will discuss, SER1 and SER2, which pertain to the external SC26C92 UART. SER1 and SER2 will be easier to implement in applications, as they can be directly debugged in the Paradigm C/C++ environment.

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 ONLY**. SER1 and SER2 have baud rated based upon different arguments. These are based on a 40 MHz CPU clock (80MHz boards will have all baud rates doubled).

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

As of January 25, 2004, the function argument "16" was added for initializing SER0. This new rate provides a baud rate of 28,000 for 40MHz boards, and 57,600 for 80MHz boards.

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.

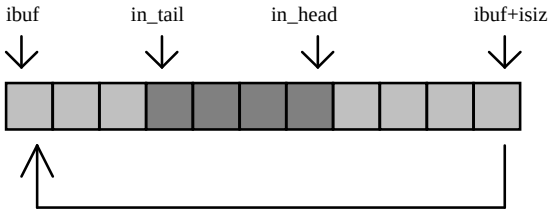


Figure 4.1 Circular ring input buffer

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.

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;
```

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 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 '**putser*n***', 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.

4.5 Other Sample code

The following is a list of other sample code available for the UB. Each will show an example implementation of the specific hardware and are located in the tern\186\samples\ub directory. The following are some of the sample applications that can be found in the **ub.ide** sample project.

Sample	Description
ub_usb.axe	Echo's characters via USB
ub_ad16.axe	Reads from both 16-bit ADC's AD7606
ub_ad16_USB.axe	Reads 8 ch. 16-bit ADC and dumps the results to USB at 100 KHz
ub_ad24.axe	Read from the 24-bit ADC LTC2448
ub_da.axe	Writes to the 16-bit DAC DA8544
ub_rtc.axe	Initializes and read from the real-time clock
ub_cf.axe	Reads and writes raw data to the compact flash card
ub_fs_cmd_usb.axe	Text interface to the file system over the USB virtual COM port
act_ub.axe	Text interface to the UB I/O functions over the USB virtual COM port

4.5.1 File system support

TERN libraries support FAT file system for the Compact Flash interface. Refer to Chapter 4 of the FlashCore technical manual (tern_docs\manuals\flashcore.pdf) for a summary of the available routines. The libraries and header files are as follows:

fileio.h

filegio.h

filesy16.lib

mm16.lib

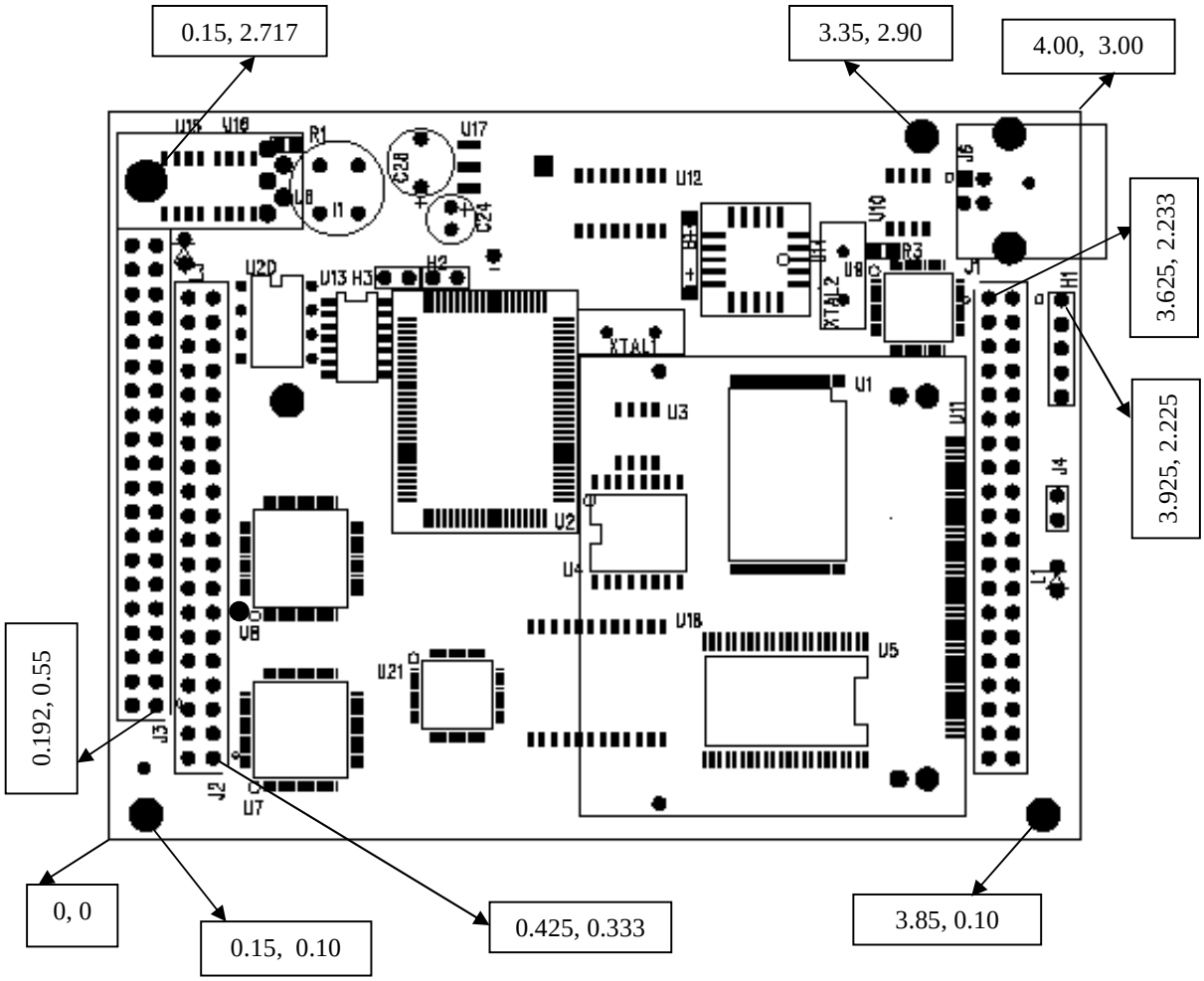
The UB uses a 16-bit external A/D bus. The user must then link to the libraries for 16-bit external busses, filesy16.lib and mm16.lib. In addition, if using the fs_cmds1 sample, you must define 'TERN_186' and 'TERN_16_BIT' in the ROM node's local options.

Libraries are found in the tern\186\lib directory and header files in the tern\186\include directory. Refer to **ub.ide** for two samples, ub_cf.c and ub_fs_cmd_usb.c.

Appendix A: Layout

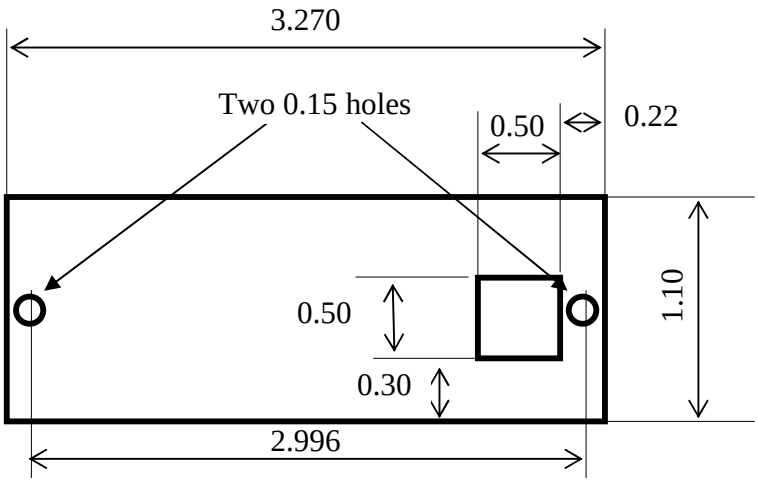
UB layout mechanical dimensions: (All in inches)

11/17/2011



Appendix B: Enclosure Dimensions

UB End Panel Dimensions: (All in inches) 08-02-2011



Appendix C: FT232H Driver Installation

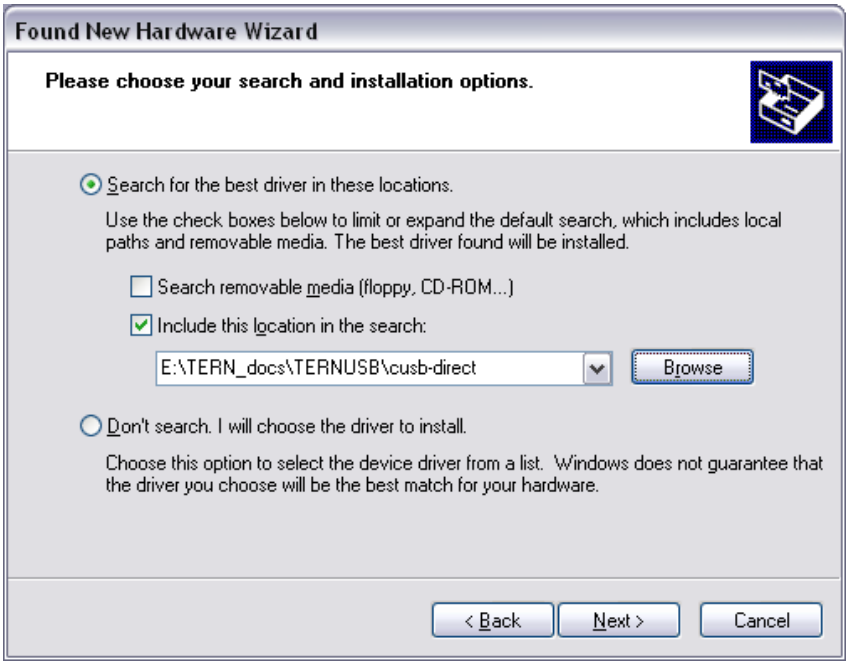
Installing FTDI's D2XX Driver

As stated earlier, the FT232H on the UB is configured by TERN to interface the D2XX Windows driver. The D2XX driver can be used with application software to directly access the FT232H through a DLL. The D2XX driver can be found on the TERN CD ***TERN_Docs\TERNUSB\cusb-direct*** or at FTDI's website **http://www.ftdichip.com**. This section describes the process of installing the D2XX driver to work with the UB.

The following instructions were performed on a Windows XP machine. Other version of Windows may vary.

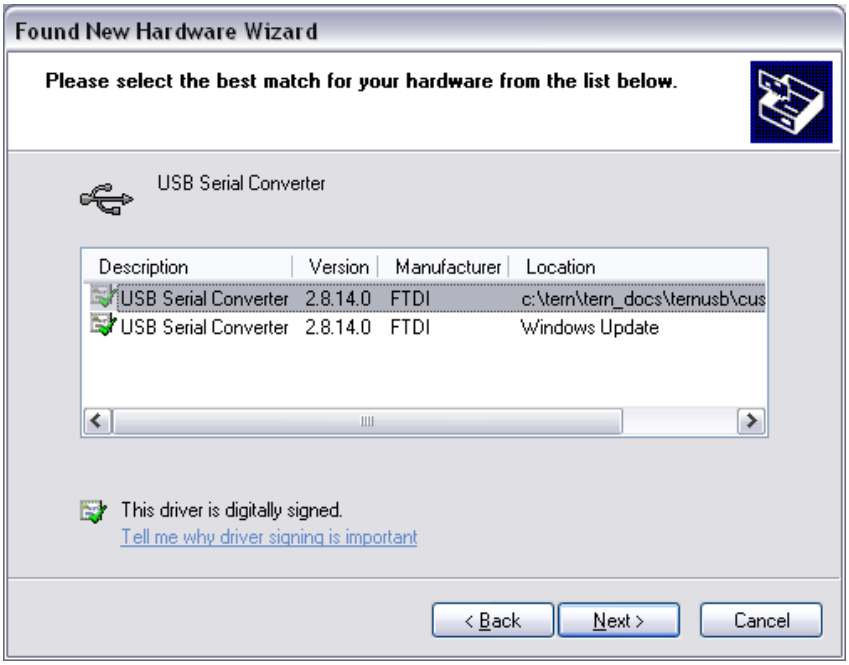
1. Connect the USB to your computer using USB cable. No other power connection is required. Once it's plugged in, Windows should detect the new hardware and prompt with the Found New Hardware Wizard.

2. Set the search location to the ***TERN_docs\TERNUSB\cusb-direct*** folder on the TERN Development Kit CD. Click “Next”.

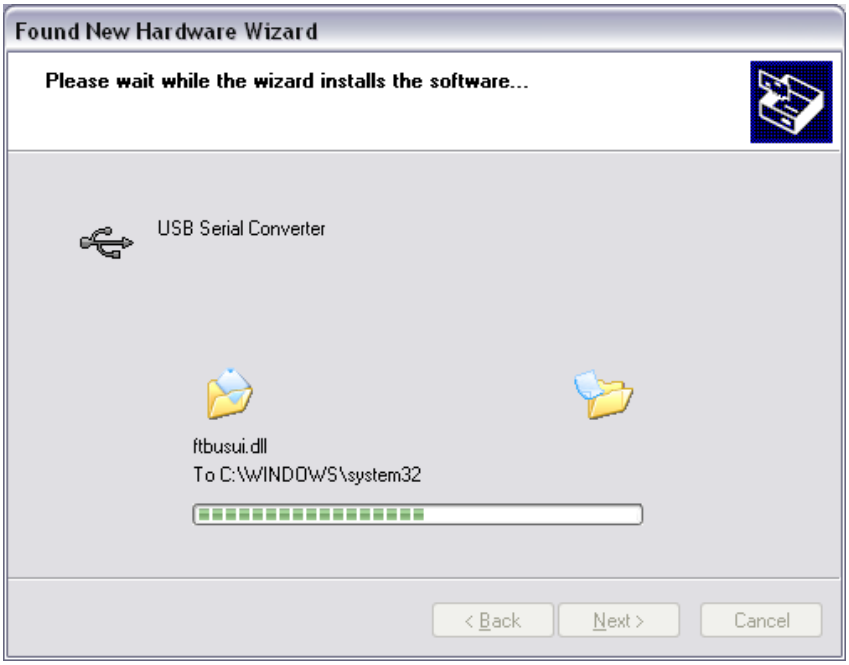


Windows should begin searching for the correct driver.

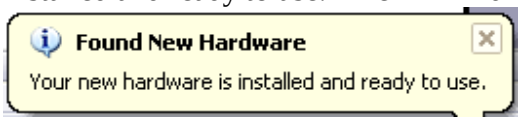
3. Select the **USB Serial Converter** driver. This example shows the same driver in two locations. This will vary on different computers, but at least one driver should be located.



4. Click “Next” and Windows will install the USB Serial Converter driver.

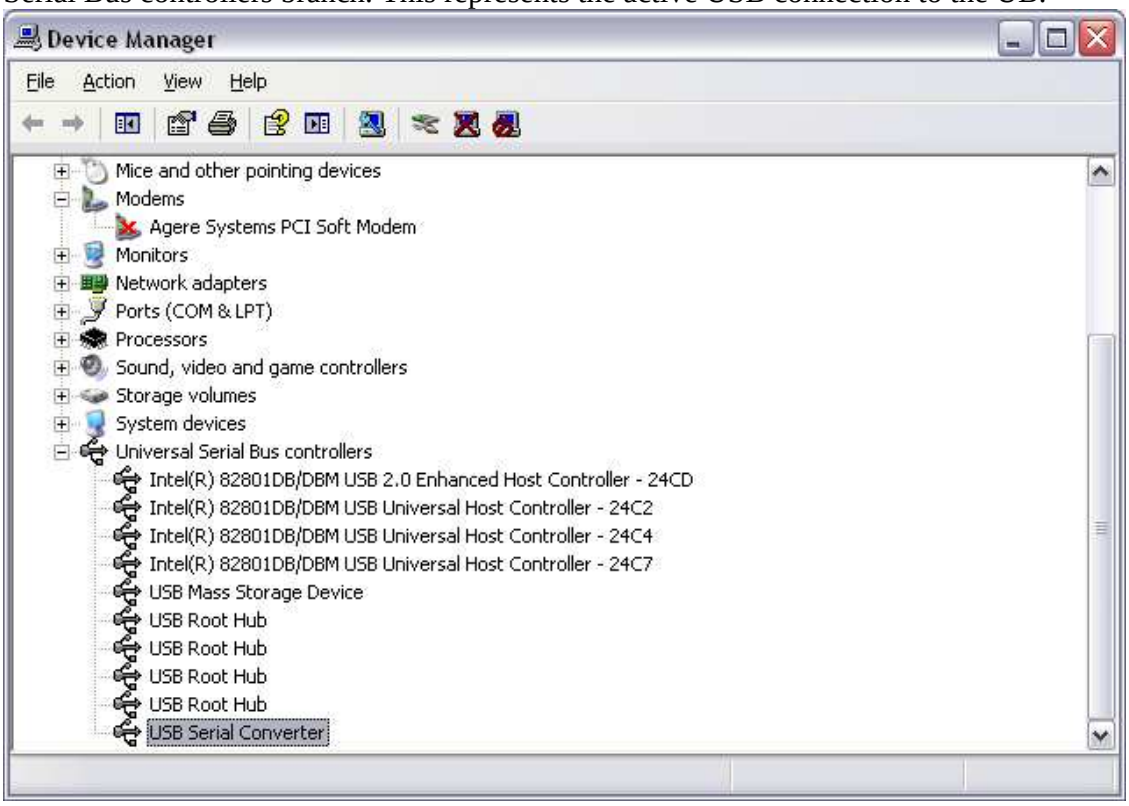


5. When the installation is complete, Windows will prompt that the new hardware is installed and ready to use. The D2XX driver is now accessible to application programs.



6. To verify the driver is installed, open System Properties by right-clicking the My Computer desktop icon and selecting Properties. Click the Device Manager button on the Hardware tab.

The Device Manager should show a “USB Serial Converter” node under the Universal Serial Bus controllers branch. This represents the active USB connection to the UB.



Installing FTDI's VCP Driver

The VCP version of the driver creates a Virtual COM Port allowing legacy serial port applications to operate over USB e.g. HyperTerminal. In order to use the VCP driver, the D2XX driver must be configured to load the VCP driver when connected. This section describes the steps to setup the VCP driver.

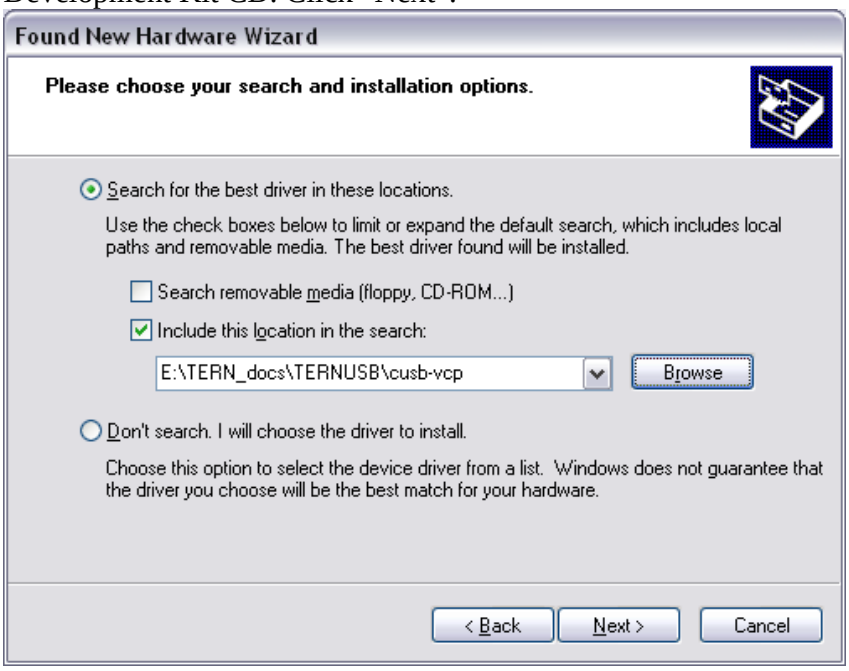
1. After installing the D2XX driver described in the last section, open the Properties tab of the "USB Serial Converter" node from the Device Manager. The previous section describes finding the "USB Serial Converter" node in the Device Manager tree.

2. Select the Advanced tab in the Properties page and check the "Load VCP" box. Click OK and exit Device Manager. This will force D2XX to use the VCP driver instead.

3. Disconnect the USB connection to the UB and then reconnect it. When reconnected, the D2XX driver will attempt to load the VCP driver and treat the UB as a new device. Windows should prompt with “Found New Hardware”.

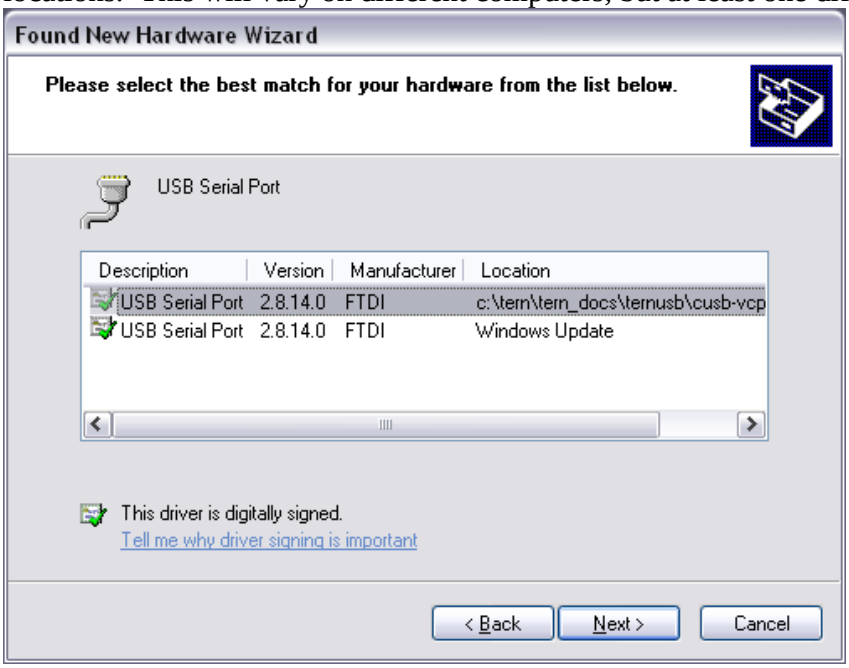
4. When the Found New Hardware Wizard appears, select to install from a specific location.

5. Set the search location to the ***TERN_docs\TERNUSB\cusb-vcp*** folder on the TERN Development Kit CD. Click “Next”.

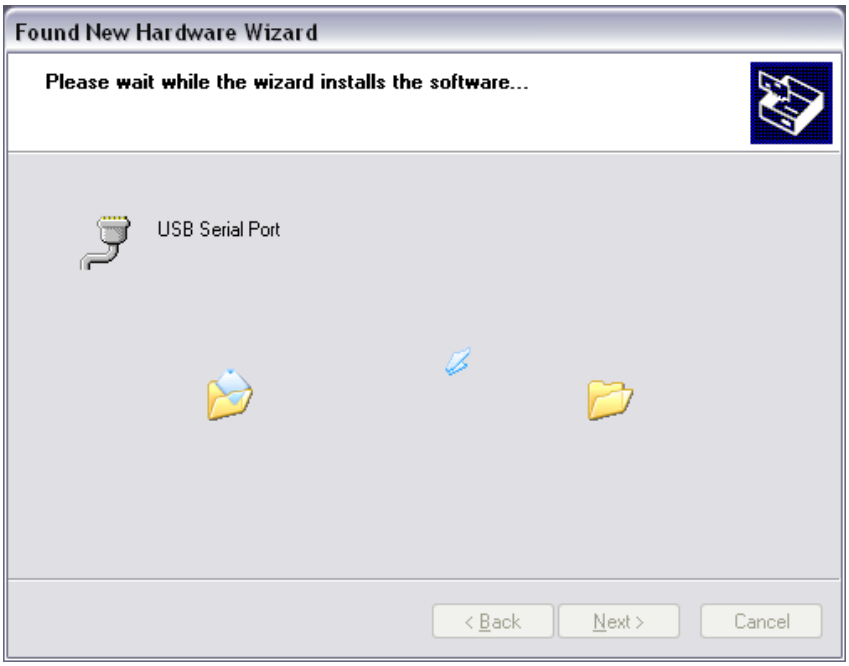


Windows should begin searching for the correct driver.

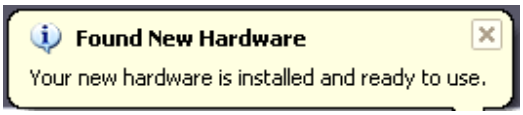
6. Select the **USB Serial Converter** driver. This example shows the same driver in two locations. This will vary on different computers, but at least one driver should be located.



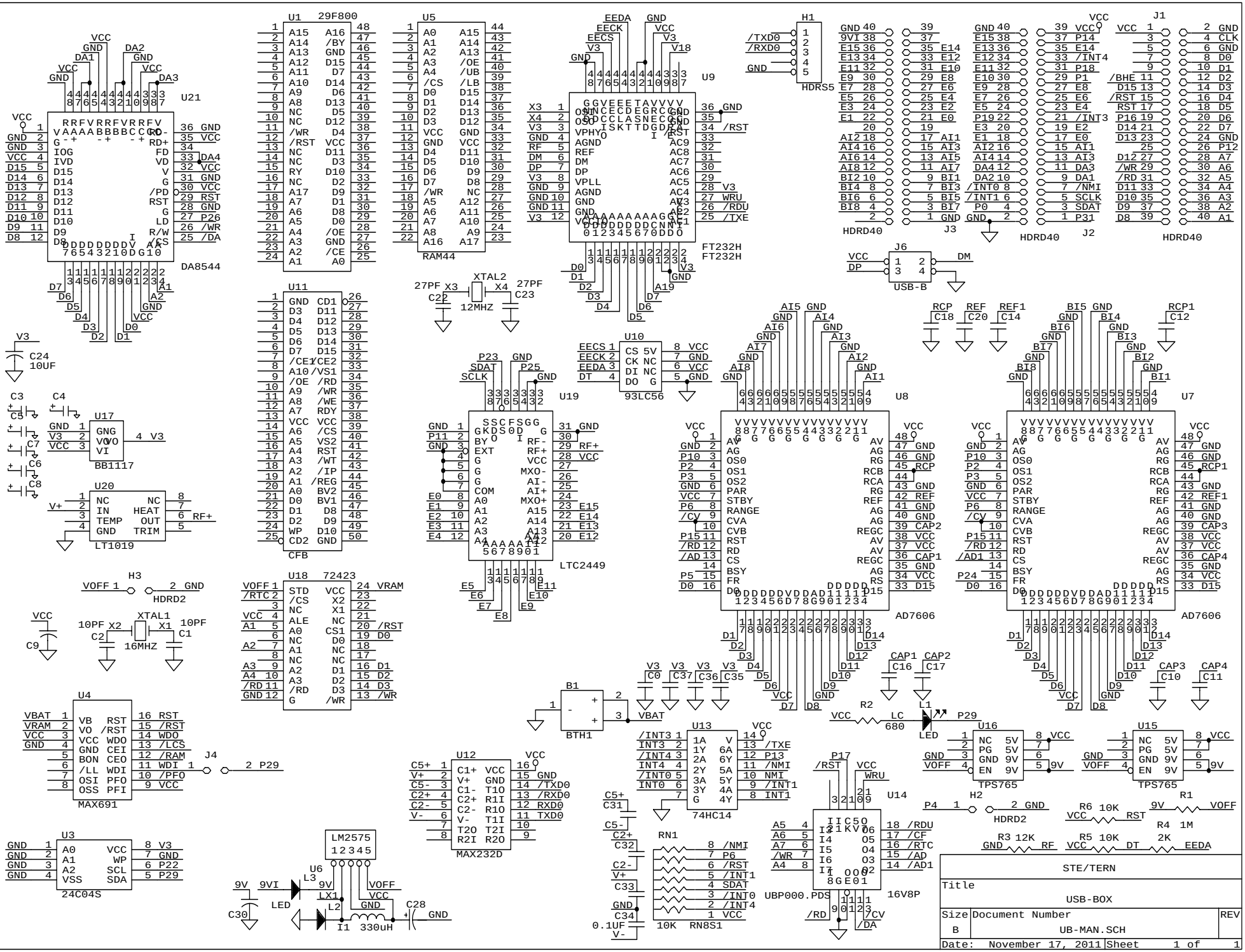
7. Click “Next” and Windows will install the USB Serial Converter driver.



8. Windows will prompt when the New Hardware installation is complete.



9. Open the Device Manager as described previously to verify the VCP driver is installed correctly. The UB should show up on the Device Manager as a USB Serial Port. The UB can now communicate with the PC like a serial COM Port.


U1 29F800
U5
RAM44
U9
FT232H
FT232H
H1
HDRS5
HDRD40
J3
HDRD40
J2
HDRD40
J1
U21
DA8544
J6
USB-B
XTAL2
27PF X3 X4 27PF
C22
12MHZ
C23
U11
CFB
U10
93LC56
U19
U8
U7
U17
BB1117
U20
LT1019
U18 72423
LTC2449
AD7606
AD7606
H3
HDRD2
XTAL1
10PF X2 X1 10PF
C2
16MHZ
C1
U4
MAX691
J4
U3
24C04S
B1
BTH1
U13
74HC14
U12
MAX232D
RN1
10K RN8S1
U14
UBP000.PDS 16V8P
U16
TPS765
U15
TPS765
H2
HDRD2
LM2575
12345
U6
L3
LED
L2
I1 330uH
C28
C30
R6 10K
R4 1M
R1
R3 12K
R5 10K
2K
STE/TERN
Title
USB-BOX
Size B
Document Number
UB-MAN.SCH
REV
Date: November 17, 2011
Sheet 1 of 1