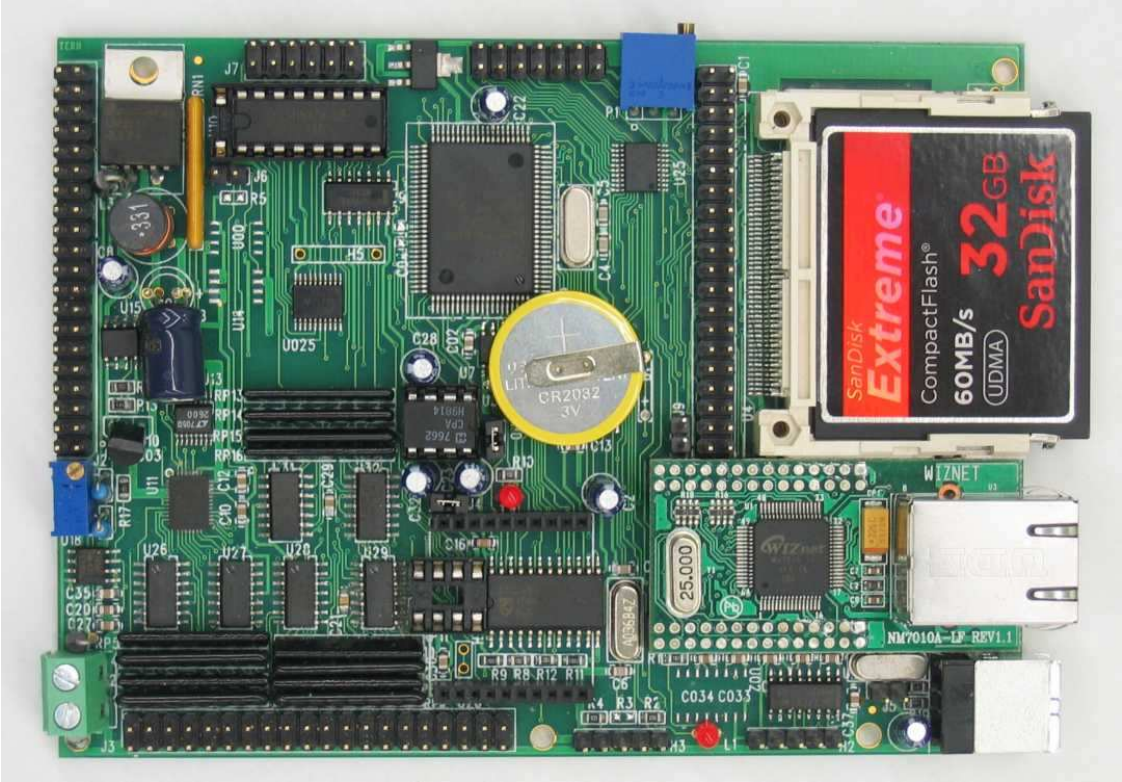
RC™

Industrial Analog Signal Acquisition and Control



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

Version 1.1

August 21, 2014

© 1993-2014 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

The ***RC***™ is a low cost industrial analog data acquisition and control module. It is designed to directly measure up to 16 channels of analog input signals at ±10V, and output up to 8 channels of analog signal at ±5V. It is ideal for an OEM product or applications that require mass local data storage, communication via Ethernet, USB, and RS232/485/wireless. The ***RC***™ supports TTL I/O lines, an LCD interface, solenoid drivers, and an on-board precision temperature sensor with 0.0625°C resolution. Additional I/O, ADCs, UARTs, counters, high voltage I/Os are available by using TERN expansion boards.

The ***RC***™ is C/C++ programmable with TERN's development platform. Ready to use demo firmware can be loaded into the on-board Flash memory.

The ***RC***™ supports a sigma-delta 24-bit ADCs (LTC2448) to provide 16 analog input channels. Each channel is buffered by an OP-Amp circuit to allow ±10V analog inputs. True 16-bit resolution can be achieved. Variable speed/resolution settings (up to 1 KHz) is available. A 16-bit DAC(LTC2600) with output buffers provides 8 channels of analog output (±5V). Mass data stored on a local CF card (up to 2 GB) can be easily transferred to a PC. FAT16 file system libraries are provided with the development system.

A real time clock (RTC72423) can provide clock/calendar for time stamp usage. A UART (SCC2691) supports RS232, or RS485, or ZigBee wireless. On-board sockets and demo software are available to support XBee and XBee-PRO ZB embedded RF modules (www.digi.com).

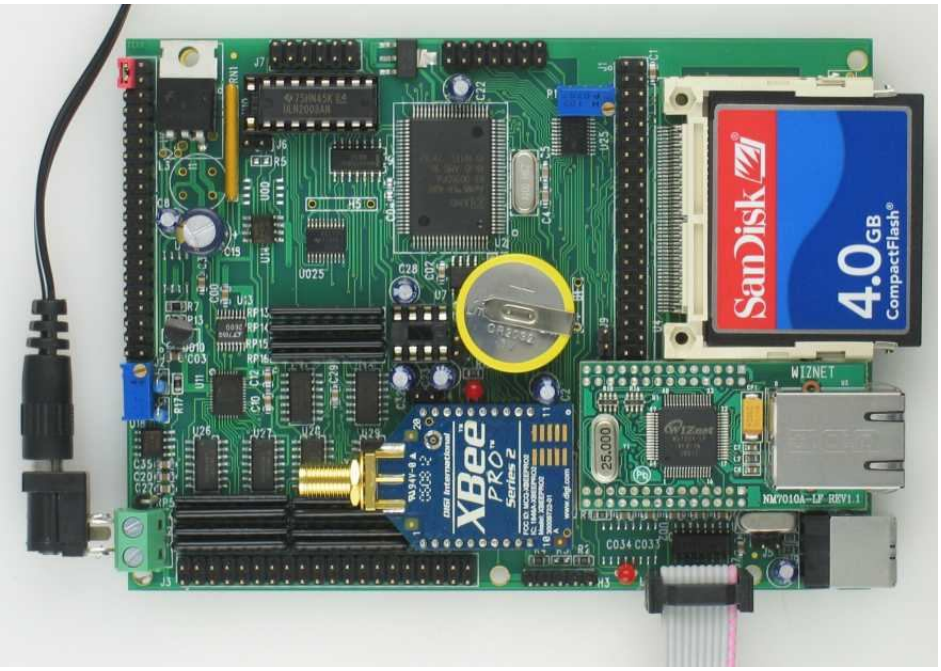
A CPU internal UART is used as RS232 DEBUG port. Three internal timer/counters can be used to count or time external events, or to generate PWM outputs. An optional 10/100-baseT Ethernet module and a slave USB port can be installed.

Over Fourteen PIO lines can be used to drive temperature IC-Sensors. Seven solenoid drivers are capable of sinking 350 mA at 50V per line, and they can directly drive solenoids, relays, or lights.

A high-performance USB stack chip to provides an easy to program USB 1.1/2.0 slave interface. The onboard hardware fully handles USB stack processing, and provides for high-speed bi-directional 8-bit parallel communication. The hardware interface includes a 384 byte FIFO transmit buffer, and a 128 byte FIFO receive buffer, making this an ideal low-overhead solution for all embedded applications. No USB specific firmware programming is required on the controller side.

The ***RC***™ uses 8.5V to 12V DC power supply with default linear regulator or up to 30V DC with switching regulator without generating excessive heat. It can be powered with 5.1V-9V DC while low drop regulators (TPS765) are installed. A 16x2 line character LCD can be installed.

RC™ with CompactFlash, USB, Zigbee and Ethernet.



RC™ with CompactFlash, USB, Ethernet, ZigBee Wireless and LCD module.

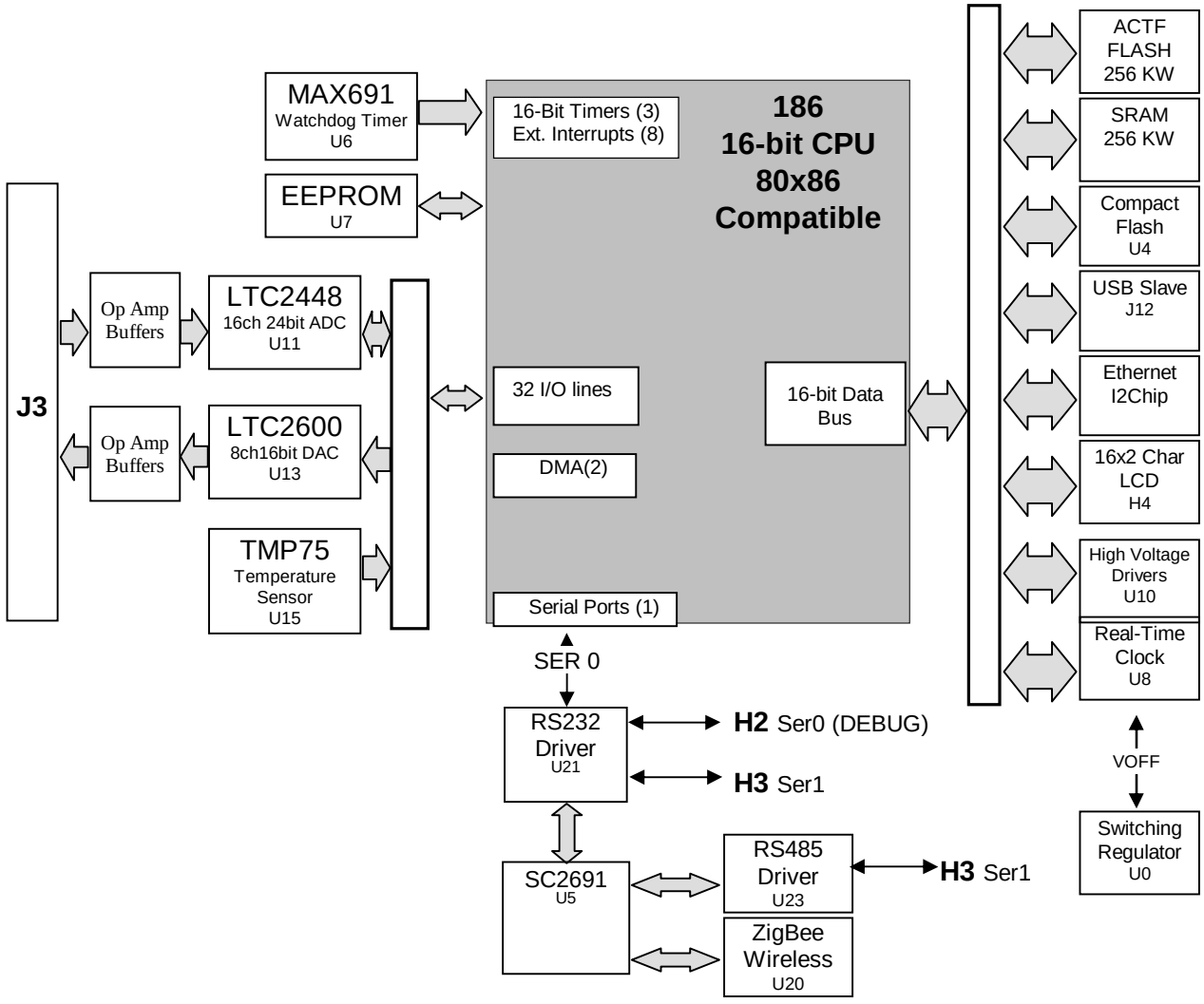


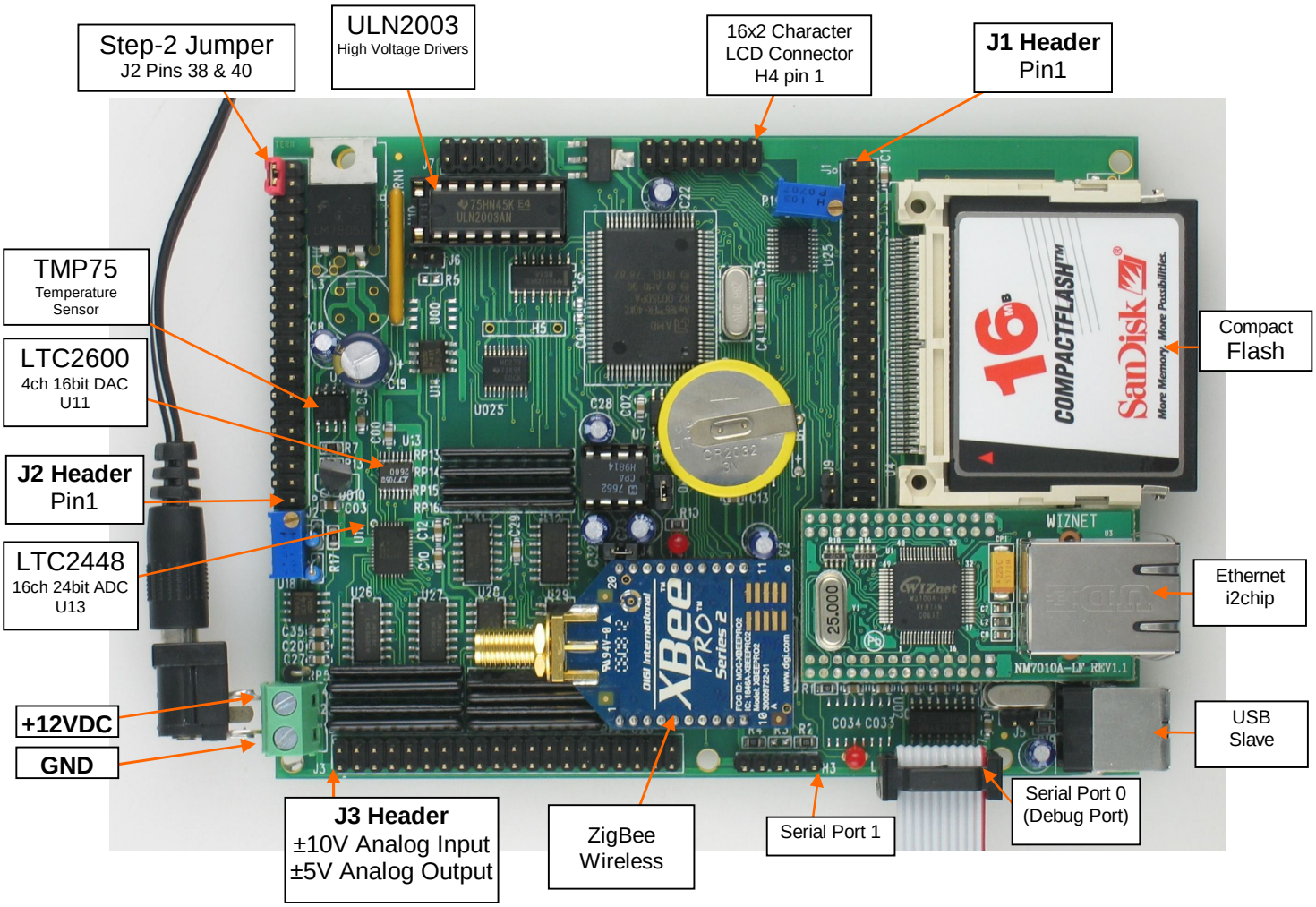
Figure 1.1 Functional block diagram of the RC™

Features:

* 5.0 x 3.7", 50 µA standby, 200 mA, 5-24V DC power
* C/C++ programmable, Supports TERN expansion boards
* On-board Temperature Sensor with 0.0625°C resolution
* 16 ch. 16-bit ADC supporting ±10V analog inputs
* 8 ch. 16-bit DAC supporting ±5V analog outputs
* 15+ TTL I/Os, 7 Solenoid drivers
* CompactFlash with FAT file system
* 16x2 line LCD, Real Time Clock, Timer/Counters, PWM
* UART with RS-232, or RS485, or ZigBee wireless support
* 10/100M-baseT Ethernet module and Slave USB port
* USB slave port

1.2 Physical Description

Below shows the physical description of the RC™.



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 (J1 pins 1 & 2). 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 RC™ for communication with the Paradigm C/C++ IDE for downloading and debugging applications. The following diagrams show the procedure for programming the RC™. Steps include preparing the RC™ for debugging, debugging the RC™, standalone field test, and production.

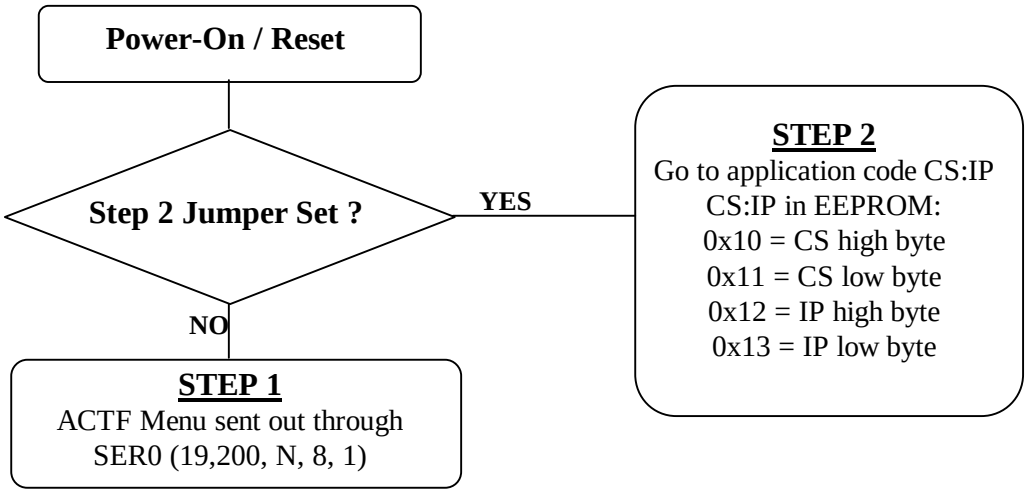


Figure 1.2 Flow Chart of ACTF Operation

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 RC™ (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 RC™ 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. 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 RC™ 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\RC™\RC™.ide.
● Use samples within the "RC™.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.
● Remove STEP 2 jumper and setup Hyper Terminal link with ST. (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>" command to execute application. Set STEP 2 jumper (J3 pins 1&3 or H1 pins 1&2). At every power-on/reset, application at 0x08000 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 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 you application in flash. Set STEP 2 jumper (J1 pins 1 & 2). At power-on/reset application will execute.

There is no ROM socket on the RC™. 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 RC™ System Development

Minimum Hardware Requirements

- PC or PC-compatible computer with serial COMx port that supports 115,200 baud
- RC™ 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 RC™ 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 "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.
- 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 RC to the PC

Fig 2.1, 2.2 and 2.3 show the location of the debug serial port and power jack. The RC is linked to the PC via a serial cable (DB9-IDE) which is supplied with TERN's EV-P / DV-P Kits.

The RC communicates through SER0 by default. Install the 5x2 IDC connector on the SER0 H2 5x1 pin header. ***IMPORTANT:*** *Note that the* ***red*** *side of the cable must point to pin 1 of the H2 header and the pins connect to the top row of the 5x2 IDC connector.* The DB9 connector should be connected to one of your PC's COM Ports (COM1 or COM2).

2.2.2 Powering-on the RC

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 screw terminal at the corner of the board is positive 12V input and the other terminal is GND (see figure for details). A power jack adapter (seen below) 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 RC. 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.

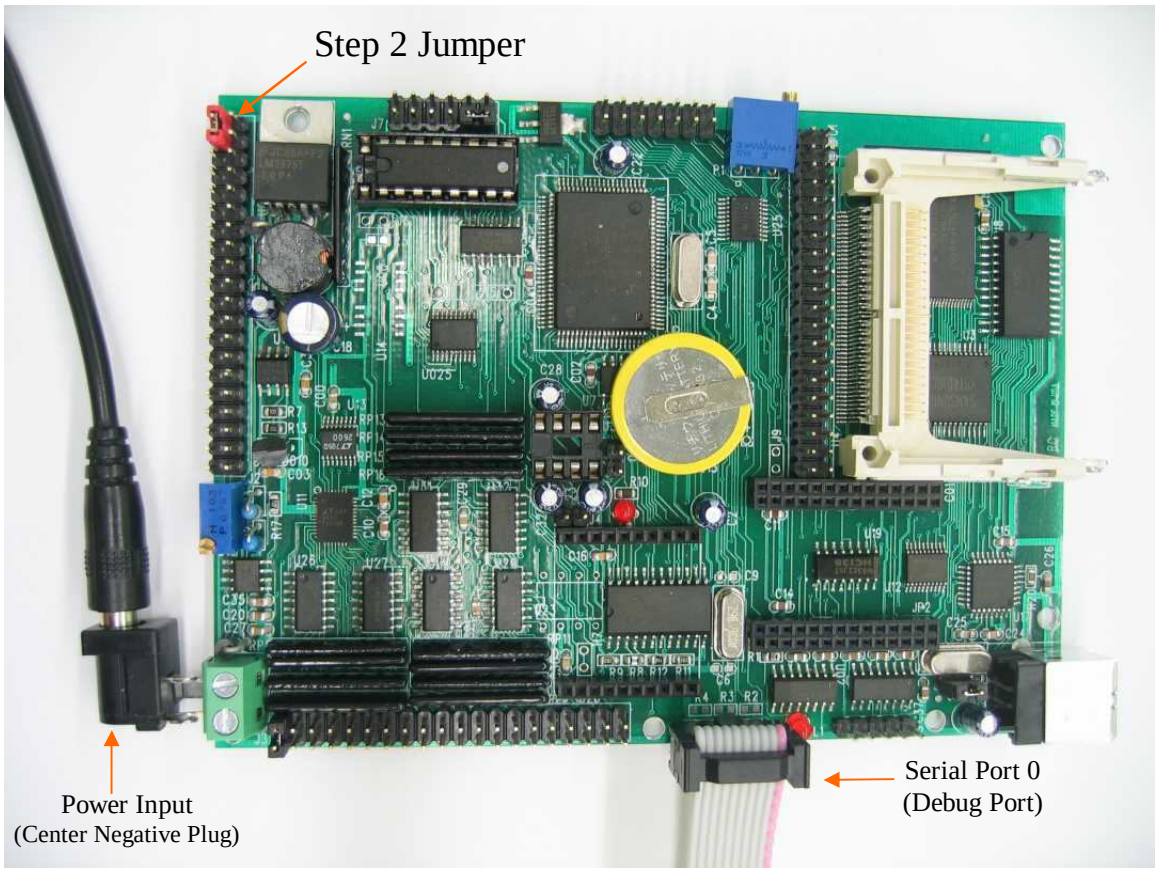


Figure 2.1 Locations of STEP2 Jumper, LED, Power input and DEBUG port

Chapter 3: Hardware

3.1 *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 RC™ without external SRAM, relying only on the Am186ER's internal RAM.

3.2 *Am186ER – Features*

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.

External Interrupts and Schmitt Trigger Input Buffer

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

/INT0, used by SC26C91 UART.
/INT1, J2 pin 6, free for application use
INT2, J2.19, is not inverted and is free for application use
/INT3, J2.21, free for application use
/INT4, J2.33, is used by Ethernet I2Chip and free when Ethernet is not installed
NMI, tied to /PFO of MAX691 supervisor chip through 74HC14 inverter.

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.

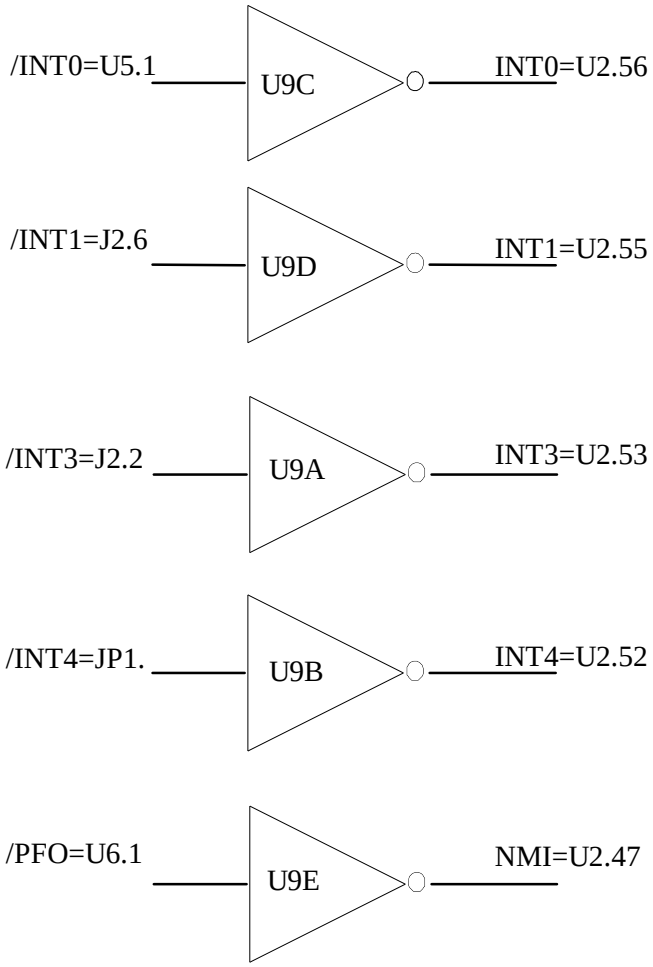


Figure 3.1 External interrupt inputs

Remember that /INT0 is used by the on board UART. /INT0 should not be used by application if the SCC2691 (U5) is installed.

The RC™ 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 Am186ER 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
- 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*

An external SCC26C91 UART is located in position U5. For more information about the external UART SCC26C91, please refer to the section in this manual on the SCC26C91, or the data sheet. From the root directory of the installation CD, \tern_docs\parts\scc26c91.pdf

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	= J2.12
Timer0 input	= P11	= J2.14
Timer1 output	= P1	= J2.29 and H5.1 Beeper
Timer1 input	= P0	= J2.20

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

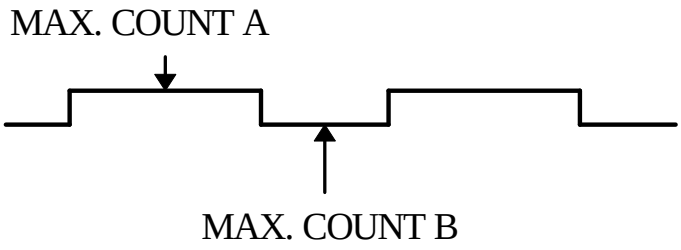
Timer2 is not connected to any external pin. It can be used as an internal timer for real-time coding or time-delay applications. It can also prescale timer 0 and timer 1 or be used as a DMA request source.

The maximum rate at which each timer can operate is 10 MHz for the Am186ER 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 RC™ 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.

The RTC72423 on the RC™ has a VOFF signal routed to J6 pin 2. VOFF is controlled by the battery-backed RTC72423. The VOFF signal can be programmed by software to be in tri-state or to be active low. The RTC72423 can be programmed in interrupt mode to drive the VOFF pin at 1/64 second, 1 second 1 minute, or 1 hour intervals. The user can use the VOFF line to control the switching power supply to turn the power supply on/off. More details are available in the sample file ***re_voff.c*** in the **`186\samples\re`** sub-directory.

3.3 *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*	*RC™ Pin No.*	*RC™ Initial1 after ae_init(); function call*
P0	Timer1 in	Input with pull-up	J2 pin 20	Input with pull-up
P1	Timer1 out	Input with pull-down	J2 pin 29, H5 pin 1 beep	Input with external pull-up
P2	/PCS6/A2	Input with pull-up	J2 pin 27	/PCS6
P3	/PCS5/A1	Input with pull-up	J2 pin 23	/PCS5
P4	DT/R	Normal	J2 pin 38	Input with pull-up: Step 2
P5	/DEN/DS	Normal	J2 pin 30	Input with pull-up
P6	SRDY	Normal	J2 pin 35	Input with external pull-up*
P7	A17	Normal	N/A	A17
P8	A18	Normal	N/A	A18
P9	A19	Normal	J2 pin 10	Input with pull-up
P10	Timer0 out	Input with pull-down	J2 pin 12	Input with external pull-up
P11	Timer0 in	Input with pull-up	J2 pin 14	Input with pull-up
P12	DRQ0	Input with pull-up	J1 pin 26	Output
P13	DRQ1	Input with pull-up	J2 pin 11	Input with pull-up
P14	/MCS0	Input with pull-up	J2 pin 37, JP1 pin 5	Input with pull-up
P15	/MCS1	Input with pull-up	J2 pin 23	Input with pull-up
P16	/PCS0	Input with pull-up	J1 pin 19	/PCS0
P17	/PCS1	Input with pull-up	U19 pin 4	/PCS1 for U19 HC138
P18	/PCS2	Input with pull-up	J2 pin 13, U15 pin 1	Input with external pull-up
P19	/PCS3	Input with pull-up	J2 pin 31	Input with external pull-up
P20	SCLK	Input with pull-up	J2 pin 5, U11.38, U13.8	Input with pull-up
P21	SDATA	Input with pull-up	J2 pin 3, U11.37, U13.9	Input with external pull-up
P22	SDEN0	Input with pull-down	U7 pin 6	Output
P23	SDEN1	Input with pull-down	J2 pin 9	Input with pull-down
P24	/MCS2	Input with pull-up	J2 pin 17, U15.2, U13.7	Input with pull-up
P25	/MCS3	Input with pull-up	J2 pin 18	Input with pull-up
P26	UZI	Input with pull-up	J2 pin 4	Input with external pull-up*
P27	TxD0	Input with pull-up	J2 pin 34	TxD0
P28	RxD0	Input with pull-up	J2 pin 32	RxD0
P29	S6/CLKSEL1	Input with pull-up	LED1, U7.5, J9.2	Output*
P30	INT4	Input with pull-up	J2 pin 33, JP1 pin 2	Input with external pull-up
P31	INT2	Input with pull-up	J2 pin 19	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

RC™ 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 RC™ 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**
P1	J2.29	Beeper H5.1
P4	J2.38	Step Two jumper
P7	A17	Upper address line – **Never use by application**
P8	A18	Upper address line – **Never use by application**
P14	J2.37	Chip select for Ethernet module
P17	U19.4	Chip select for U18 HC138
P18	J2.13	Data pin for U15 TMP75
P20**	J2.5	SCLK - Synchronous Clock for U11 and U13
P21**	J2.3	SDAT - Serial Interface for U11 and U13
P22	U7.6	Clock for EEPROM
P24	J2.17	Clock pin for TMP75 (U15) and /CS for LTC2600 DAC (U13)
P27	J2.34	TxD0 - Transmit line for RS232 debug port
P28	J2.32	RxD0 - Receive line for RS232 debug port
P29*	U7.5, J9.2	Data pin for EEPROM , watch dog timer pin, LED1
P30	J2.33	INT4 - Interrupt for Ethernet module

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

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. 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 TLC2448 ADC and TLC2600 DAC use these lines. The user is free to use the SCLK and SDAT lines for their application only if the ADC and DAC are disabled first. This is needed so as not to have more than one device trying to occupy the SDAT line simultaneously.

3.4 *I/O Mapped Devices*

I/O Space

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

The default I/O access time is 15 wait states. You may use the function void ***io_wait***(char wait) to define the I/O wait states from 0 to 15. The system clock is 100 ns for the Am186ER and the CPU clock is 25ns. Details regarding this can be found in the Software chapter of this manual, 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	USER
0x0100-0x01ff	/PCS1	U19 pin 4 = P17	HC138
0x0100-0x011f	/RTC	U8 pin 2	RTC 72423
0x0120-0x013f	/RTU	U17 pin 4	USB Reset
0x0140-0x015f	/SC	U5 pin 14	SCC2691
0x0160-0x017f	/RD2	U12 pin 1 & 19	HC244
0x0180-0x019f	/LH	U25 pin 11	HC273
0x01a0-0x01bf	/RDU	U17 pin16	USB Read
0x01c0-0x01df	/WRU	U17 pin 15	USB Write
0x01e0-0x01ff	/CF	U4 pin 32	Compact Flash
0x0200-0x02ff	/PCS2	J2 pin 13 = P18	TMP75 U15
0x0300-0x03ff	/PCS3	J2 pin 31 = P19	USER
0x0400-0x04ff	/PCS4		Reserved
0x0500-0x05ff	/PCS5	J2 pin 23 = P3	USER
0x0600-0x06ff	/PCS6	J2 pin 27 = P2	USER

Table 3.3 I/O Mapping

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

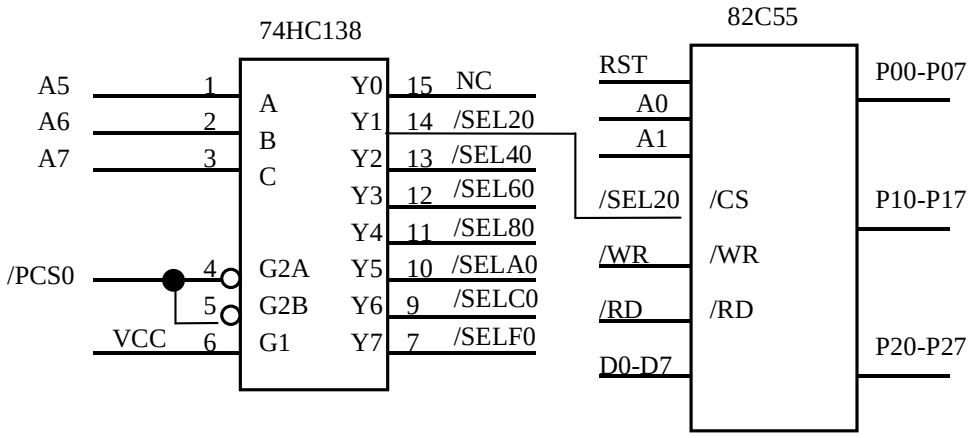


Figure 3.2 Interface the RC™ to external I/O devices

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.

Real-time Clock RTC72423

If installed, the real-time clock RTC72423 (EPSON, U8) 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\186\samples\rc\poweroff.c*

UART SCC2691

The UART SCC2691 (Signetics, U5) is mapped into the I/O address space at 0x0140. The SCC2691 has a full-duplex asynchronous receiver/transmitter, a quadruple buffered receiver data register, an interrupt control mechanism, programmable data format, selectable baud rate for the receiver and transmitter, a multi-functional and programmable 16-bit counter/timer, an on-chip crystal oscillator, and a multi-purpose
input/output including RTS and CTS mechanism.

Depending on the option selected at the time of purchase, the RC™'s SCC2691 can be used as either an RS232 port, an RS485 port or communicate directly with the XBee Pro Zigbee wireless module. When configured as RS232, the SCC2691 TxD and RxD lines are connected to the on-board RS232 (U21) driver. When configured for RS485, the TxD, RxD and MPO lines are connected to the on-board RS485 (U23) driver. If the Zigbee option was purchased for the RC™, the TxD, RxD, MPI and MPO lines communicate with the XBEE Pro module (U20).

3.5 *Other Devices*

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

On-board Supervisor with Watchdog Timer

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

Watchdog Timer

The watchdog timer is activated by setting a jumper on J9 of the RC™. 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 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 RC™ 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 Am186ER has an internal watchdog timer. This is disabled by default with **ae_init()**.

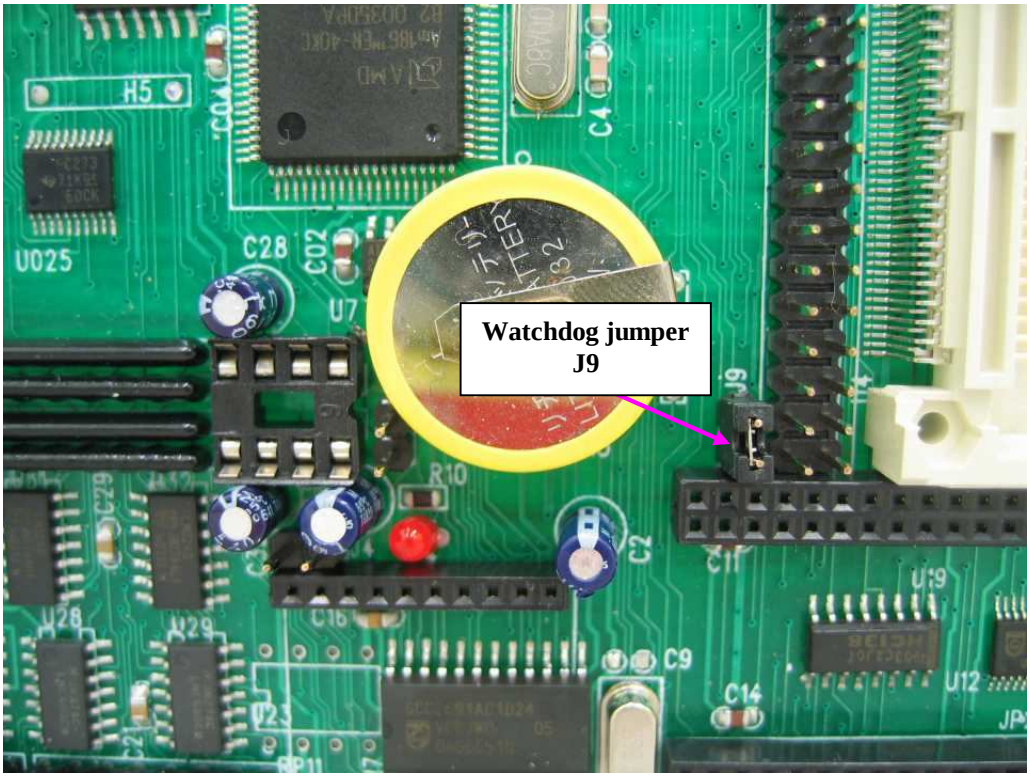


Figure 3.3 Location of watchdog timer enable jumper

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 72423 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.

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.

EEPROM

A serial EEPROM of 512 bytes (24C04) is installed in U7. The RC™ 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.

24-bit, 16-channel ADC (LTC2448)

The RC™ features an optional high-speed LTC2448 sigma-delta 24-bit ADC (U11) to provide 16 analog input channels. LTC2448 is interfaced to a high-speed synchronous serial port for fast data collection. Variable speed/resolution settings (up to 1 KHz) is available and true 16-bit resolution can be achieved. The LTC2448 analog input pins, N0-N15, are located at hearder J3 (see schematics). Sample programs for reading the LTC2448 are found in the ***c:\tern\186\samples\rc\rc_ad24.axe*** project.

The LTC2448 supports a maximum input voltage (Vo) of Reference ÷ 2. With the default 4.095V reference, the maximum input voltage is 2.0475V. Each input channel is buffered by an op amp circuit to allow ±10.24V analog input. By changing resistor values of Ri, Ro, R22 and R23, the op amp buffer can be configured for various input voltage ranges.

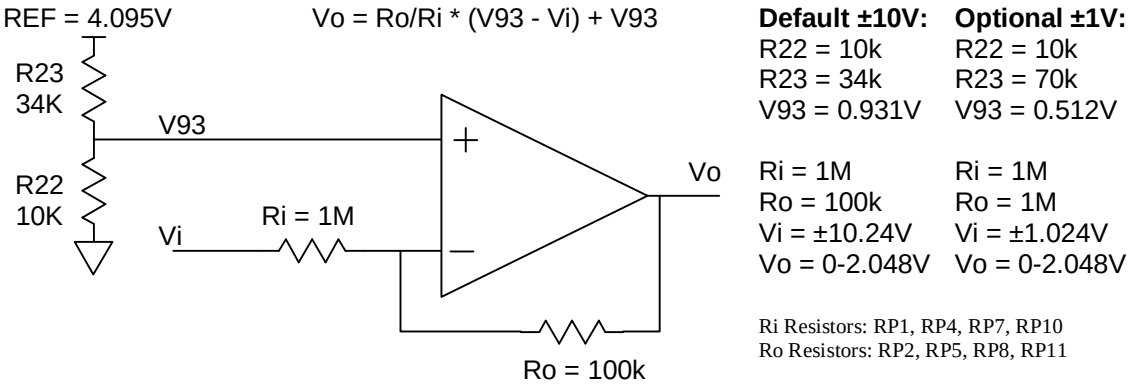


Figure 3.4 24-bit Analog Input Buffer

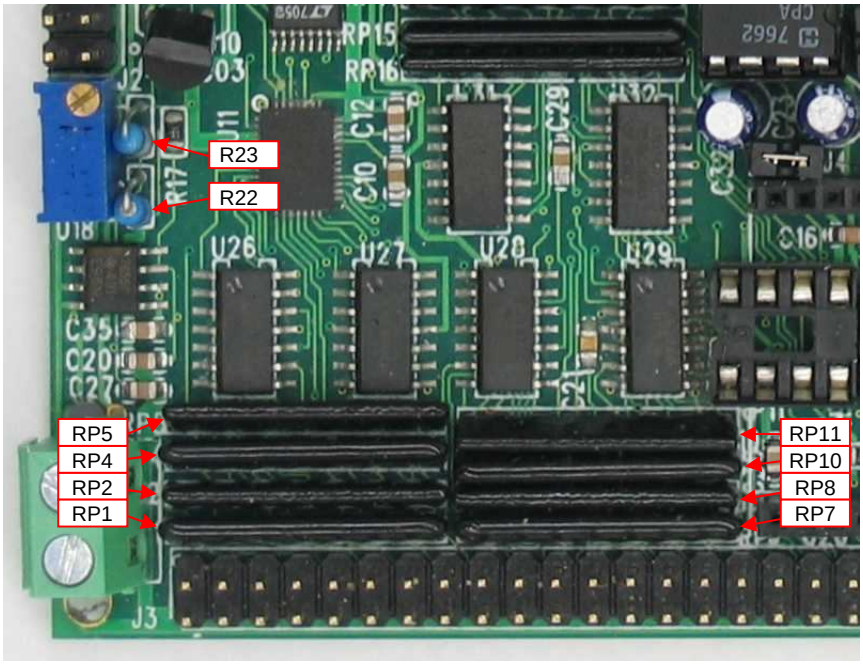


Figure 3.5 Analog Input Resistors

16-bit, 8-channel DAC (LTC2600)

An optional LTC2600 16-bit DAC chip provides 8 channels of analog output. Each channel is buffered by an op amp circuit to allow ±5V analog output. LTC2600 is interfaced to a high-speed synchronous serial port. DAC outputs, V1-V8, are located at header J3, pins 33-40 (see schematics). Sample programs for writing to the LTC2600 are found in the ***c:\tern\186\samples\rc\rc_da.axe*** project.

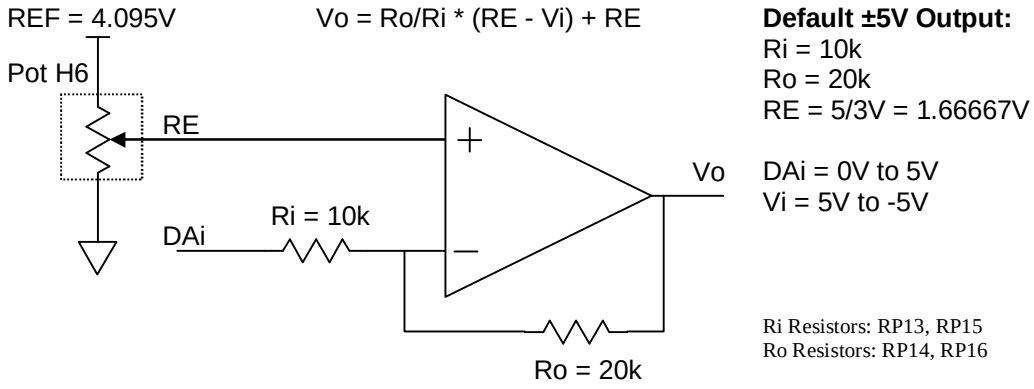


Figure 3.6 16-bit Analog Output Buffer

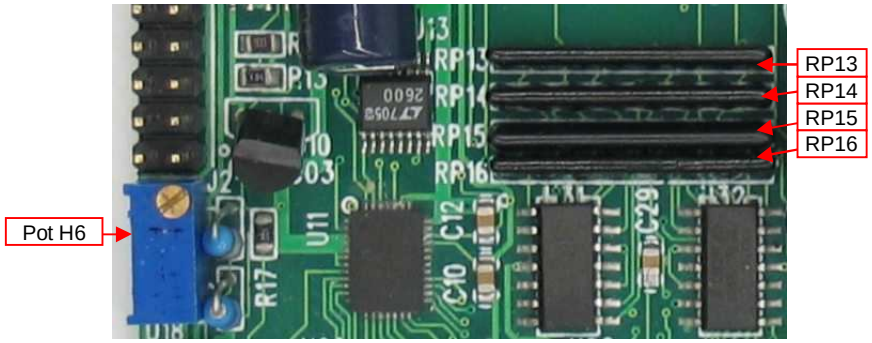


Figure 3.7 Analog Output Resistors

High-Voltage, High-Current Drivers

The RC™ comes with seven channels of high-voltage/high-current drivers. The ULN2003A (U10) has high voltage, high current Darlington transistor array, consisting of seven silicon NPN Darlington pairs on a common monolithic substrate. All channels feature open-collector outputs for sinking 350 mA at 50V, and integral protection diodes for driving inductive loads. Peak inrush currents of up to 500 mA sinking are allowed.

These outputs may be paralleled to achieve high-load capability, although each driver has a maximum continuous collector current rating of 350 mA at 50V. The maximum power dissipation allowed is 2.20 W per chip at 25 degrees C (°C).

ULN2003 is a sinking driver, not a sourcing driver. An optional UDS2982 high-voltage sourcing output chip can be installed.

An example of a typical application wiring is shown below.

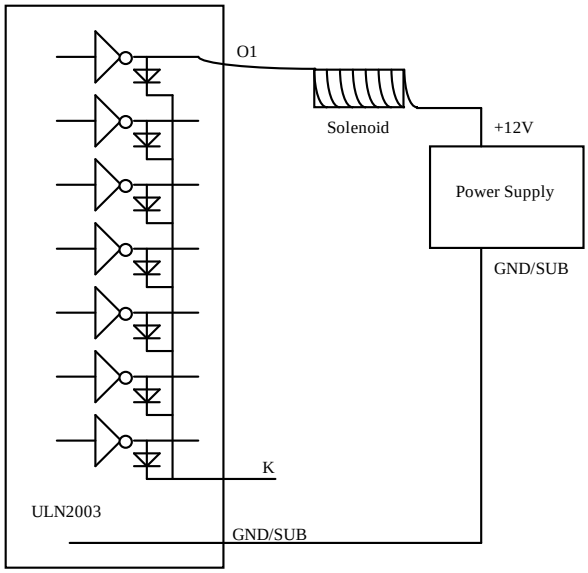


Figure 3.8 Darlington Transistors used as high-voltage/high-current OUTPUTS.

Compact Flash Interface
By utilizing the compact flash interface on the **RC™**, 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\rc** directory includes sample code to show reads and writes of raw data by sector. In addition, **tern\186\samples\rc\rc.ide** includes the file system demo **rc_filesys.axe** in the project

Ethernet Module
The **RC™** supports an i2Chip W3100A which is an LSI of hardware protocol stack that provides an easy, low-cost solution for high-speed internet connectivity for digital devices by allowing simple installation of TCP/IP stack in the hardware. The i2Chip offers a quick and easy way to add Ethernet networking functionality to embedded controllers. It can completely offload internet connectivity and processing standard protocols from the host system, reducing development time and cost. It contains TCP/IP protocol stacks such as TCP, UDP, IP, ARP and ICMP protocols, as well as Ethernet protocols such as Data Link Control and MAC protocol. The full datasheet is provided on the TERN CD:
\tern_docs\parts\w3100a datasheet 3.1.pdf. Also find sample programs for accessing the module in the RC™ sample project, ***c:\tern\186\sampels\rc\rc.ide***. Sample programs include **i2chip.c** and **i2_dma.c**. The
module is accessed by P14 = /MCS0 (midrange chip select). The chip select can be mapped into memory by initializing the MMCS and MPCS registers. See chapter 5 of the Am186ER technical manual (amd_docs\am186er).

USB

The ***RC™*** integrates a high-performance USB stack chip to provide an easy to program USB 1.1/2.0 slave interface. The onboard hardware fully handles USB stack processing, and provides for high-speed bidirectional 8-bit parallel communication. The hardware interface includes 384 bytes of FIFO transmit buffer, and 128 bytes of FIFO for the receiving buffer, making this an ideal low-overhead solution for all embedded applications. The RC™ USB exposes a slave USB interface, and connects to a PC via USB-B connector.

No USB specific firmware programming is required on the controller side. The USB interface is seen as a transparent parallel FIFO buffer tasked with transferring data back and forth with the remote host. The only control signals needed are "ready to transmit" and "data received" signals, readily available to your C/C++ application running on the TERN controller.

Royalty-free software drivers are provided for most Windows environments (XP, 2000, NT, 98). These field proven USB software drivers eliminates the requirement for Windows USB driver development. Two types of USB software drivers are available: VCP and D2xx. The VCP (Virtual Com Port) driver supports up to 300 K bytes per second transfer rate, and allowing the device to be accessed transparently on the PC side through traditional COM port software. The D2xx (USB direct driver and DLL) drivers can support up to 1M bytes per second. Additional utilities available from third-party sources allow the USB interface to be programmed with unique service and product ID numbers, allowing the unit to be transparently integrated into OEM applications.

Temperature Sensor

The ***RC™*** has an optional on-board TMP275 (U15) temperature sensor. The sensor is a 0.5°C accurate, two-wire, serial output temperature sensor. The TMP275 is capable of reading temperatures with a resolution of 0.0625°C over an operating temperature range of −40°C to +125°C.

The two-wire serial bus is driven by P18=Data and P24=Clock. The device protocol allows up to eight devices to be read from on one bus. By default, the on-board chip is set to address 0.

The ***RC™*** is ideal for connecting additional ***Temperature IC Sensors (IS)*** to the J3 header. Each ***RC™*** can read upto 100 sensors. A sample program in ***tern\186\samples\rc\tb_tmp.axe*** demonstrates how to read the on-board and external temperature sensors.

LCD Display

A 16x2 alphanumeric character-based LCD comes as part of the aluminum extrusion enclosure package. It is connected to header H4, and an on-board buffer to drive the LCD. The contrast for the LCD is set by a potentiometer (pot) at P1.

See the sample program ***tern\186\samples\rc\rc_lcd.c*** for programming details.

ZigBee Wireless

The ***RC™*** incorporates an interface for the XBee-Pro wireless module. The XBee-Pro RF module meets the IEEE 802.15.4 standards. The ***RC™*** interfaces the wireless module through the SCC2691 serial port. By default, XBee-PRO RF Module operates in Transparent Mode. When operating in this mode, the module act as a serial line replacement - all UART data received through the DI pin is queued up for RF transmission. When RF data is received, the data is sent out the DO pin. See the XBee-Pro product manual for details on the wireless module.

3.6 *Headers and Connectors*

Expansion Headers J1 and J2
There are two 20x2 0.1 spacing headers for ***RC™*** expansion. Most signals are directly routed to the Am186ER processor.

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

J2 Signal			
GND	40	39	VCC
P4	38	37	P14
	36	35	P6
TxD0	34	33	
RxD0	32	31	P19
P5	30	29	P1
TxDA	28	27	P2
RxDA	26	25	A19
	24	23	P15
I25	22	21	I24
P0	20	19	
P25	18	17	P24
I27	16	15	I26
P11	14	13	P18
P10	12	11	P13
	10	9	P23
/INT0	8	7	NMI
/INT1	6	5	SCLK
P26	4	3	SDAT
GND	2	1	

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 3.4 Signals for J2 and J1, 20x2 expansion ports

Analog Header J3

There is one 20x2 0.1 spacing header for ***RC™*** analog I/O. All analog signals are buffered with Op Amps. V1-V8 are ±5V analog outputs and N0-N15 are ±10V analog inputs.

J3 Signal			
V7	40	39	V8
V5	38	37	V6
V3	36	35	V4
V1	34	33	V2
N15	32	31	GND
N14	30	29	GND
N13	28	27	GND
N12	26	25	GND
N11	24	23	GND
N10	22	21	GND
N9	20	19	GND
N8	18	17	GND
N7	16	15	GND
N6	14	13	GND
N5	12	11	GND
N4	10	9	GND
N3	8	7	GND
N2	6	5	GND
N1	4	3	GND
N0	2	1	GND

Table 3.5 Signals for J3 analog I/O

High-Voltage I/O Header J7

There is one 6x2 0.1 spacing header for ***RC™*** high-voltage I/O. All signals connect directly to the U10 ULN2003 high-voltage driver . Jumper J7.9=J7.11 & J7.10=J7.12 for ULN2003 high-voltage sinking driver. Jumper J7.9=J7.10 & J7.11=J7.12 for UDS2982 high-voltage sourcing driver.

J7 Signal			
HV1	1	2	HV2
HV3	3	4	HV4
HV5	5	6	HV6
HV7	7	8	HV8
GND	9	10	GNK
VS	11	12	9VI

Table 3.6 Signals for J7 high-voltage I/O

Table 3.7

Serial Headers H2 and H3

There are two 5x1 0.1 spacing headers for ***RC*™** serial I/O. H2 is Serial Port 0 and used for software debugging. H3 is Serial Port 1. It can either be an RS232 or RS485 serial line. Pins 1 and 3 on H3 are for RS485 and pins 2, 3 and 5 are for RS232.

H2 Signal	
	1
/TXD0	2
/RXD0	3
	4
GND	5

H3 Signal	
485+	1
/TXD	2
/RXD	3
485-	4
GND	5

Table 3.8 Signals for H2/H3 serial I/O

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\TB.LIB

RE.LIB is a C library for basic RC 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, RTC, Watchdog
SER0.H	Internal serial port 0, from CPU
SER1R.H	External UART SCC26C92
AEEE.H	on-board EEPROM

Not all functions in the above modules will apply to the RC. 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 RC. The user will need to include the header file “re.h” to provide routines for the RC devices. Although “ae.h” was created for a different controller, it will still be needed for a variety of routines used by the RC, such as timers, interrupts, and others. Refer to the actual header file itself to determine which is needed for a certain application.

TB.LIB is a library for RC specific functions. This library must be linked in to your applications as well. It includes drivers for various input/output functions.

Include-file name	Description
RC.H	RTC, 16x2 LCD, Temperature Sensor

4.2 Functions in AE.OBJ

4.2.1 RC Initialization

ae_init

This function should be called at the beginning of every program running on RC 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 RC, 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 available on the TERN CD under the **amd_docs** directory.

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 RC. 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 RC 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 RC, 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 **RC**. 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**.

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 RC 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.

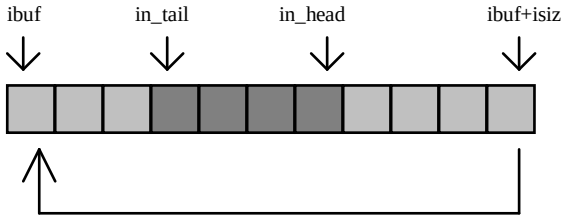


Figure 4.1 Circular ring input buffer

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.

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;
```

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 Functions in TB.LIB

TB.LIB contains drivers for ***RC*** specific hardware. In order to use these functions, RC.LIB must be included in the project tree and TB.H must be included in the C file. See the ***RC*** sample project *tern\186\samples\RC\RC.ide* for details.

RC 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\samples\rc\tb_rtc_init.c*** for a sample program. There is a common data structure used to access and use both interfaces. The structure is defined in the AE.H header file.

```
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 tb_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_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 **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};

SCC2691

The SCC is a component that is used to provide a second asynchronous port. It uses an 3.6864 MHz crystal, different from the system clock speed, for driving serial communications. This means the divisors and function arguments for setting up the baud rate for this second port are different than for SER0. The highest standard baud rate is 19,200, as shown in the table below. A sample file demonstrating how to use the software can be found in **scc_echo.c**, found in the **tern\186\samples\rc** directory.

Function Argument	Baud Rate
1	110
2	134.4
3	150
4	300
5	600
6	1200
7	2000
8	2400
9	4800
10	7200
11	9600
12	19200

Unlike the other serial ports, DMA transfer is not used to fill the input buffer for SCC. Instead, an interrupt-service-routine is used to place characters into the input buffer. If the processor does not respond to the interrupt—because it is masked, for example—the interrupt service routine might never be able to complete this process. Over time, this means data might be lost in the SCC as bytes overflow.
Special control registers are used to define how the SCC operates. For a detailed description of registers **MR1** and **MR2**, please see '*tb.h*'. In most TERN applications, MR1 is set to ***0x57***, and MR2 is set to ***0x07***. This configures the SCC for no flow control (RTS, CTS not used/checked), no parity, 8-bit, normal operation. Other configurations are also possible, providing self-echo, even-odd parity, up to 2 stop bits, 5 bit operation, as well as automatic hardware flow control.
Initialization occurs in a manner otherwise similar to SER0. A **COM** structure is once again used to hold state information for the serial port. The in-bound and out-bound buffers operate as before, and must be provided upon initialization.

scc_init
Arguments: unsigned char m1, unsigned char m2, unsigned char b, unsigned char* ibuf, int isiz, unsigned char* obuf, int osiz, COM *c
Return value: none

This initializes the SCC2691 serial port to baud rate **b**, as defined in the table above. The values in **m1** and **m2** specify the values to be stored in to **MR1** and **MR2**. As discussed above, these values are normally ***0x57*** and ***0x07***, as shown in TERN sample programs.

ibuf and **isiz** define the input buffer characteristics, and **obuf** and **osiz** define the output buffer.

After initializing the serial port, you must also set up the interrupt service routine. The SCC2691 UART takes up external interrupt **/INT0** on the CPU, and you must set up the appropriate interrupt vector to handle this. An interrupt service routine, **scc_isr()**, has been written to handle the interrupt, and it enables/disables the interrupt as needed to transmit and receive data with the data buffers. So, after initialization, you will need to make a call to do this:

```
int0_init(1, scc_isr);
```

By default, the SCC is disabled for both *transmit* and *receive*. Before using the port, you will need to enable these functionalities.
When using RS232 in full-duplex mode, *transmit* and *receive* functions should both be enabled. Once this is done, you can transmit and receive data as needed. If you do need to do limited flow control, the MPO pin on the J1 header can be used for RTS. For a sample file showing RS232 full duplex communications, please see **rc_scc_echo.c** in the directory **`tern\186\samples\RC`**.
RS485 is slightly more complex to use than RS232. RS485 operation is half-duplex only, which means transmission does not occur concurrently with reception. The RS485 driver will echo back bytes sent to the SCC. As a result, assuming you are using the RS485 driver installed on another TERN peripheral board, you will need to disable *receive* while transmitting. While transmitting, you will also need to place the RS485 driver in transmission mode as well. This is done by using **scc_rts(1)**. This uses pin MPO (multi-purpose output) found on the J1 header. While you are receiving data, the RS485 driver will need to be placed in receive mode using **scc_rts(0)**.

en485
Arguments: int i
Return value: none

This function sets the pin MPO either high (i = 1) or low (i = 0). The function scc_rts() actually has a similar function, by pulling the same pin high or low, but is intended for use in flow control.

scc_send_e/scc_rec_e
Arguments: none
Return value: none

This function enables transmission or reception on the SCC2691 UART. After initialization, both of these functions are disabled by default. If you are using RS485, only one of these two functions should be enabled at any one time.

scc_send_reset/scc_rec_reset
Arguments: none
Return value: none

This function resets the state of the send and receive function of the SCC2691. One major use of these functions is to disable *transmit* and *receive*. If you are using RS485, you will need to use this feature when transitioning from transmission to reception, or from reception to transmission.

Transmission and reception of data using the SCC is in most ways identical to SER0. The functions used to transmit and receive data are similar. For details regarding these functions, please refer to the previous section.

putser_scc
See: **putser*n***

putsers_scc
See: **putsers*n***

getser_scc
See: **getser*n***

getsers_scc
See: **getsers*n***

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

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

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

Other SCC functions are similar to those for SER0.

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

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

clean_ser_scc
See: **clean_s*n***

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

RC 16x2 Display

The RC is designed to interface a 16x2 character LCD. RC.lib provides functions to make displaying text on the LCD simple. See sample program *tern\186\samples\RC\RC_lcd.c*

rc_lcd_init
Arguments: none

Return value: none

Intializes the LCD and clears the display. The function set PIO P6 as output to drive UNL2003(U22) C14=J3.20=H0.2 to enable or disable LCD backlighting.

rc_lcd_clr_line
Arguments: unsigned int code
Return value: none

This function clears a line of the display. Use the *code* argument to set the line position. After the line is cleared, the curser is set back to the position set by the *code* argument.

rc_lcdcmd
Arguments: unsigned char cmd
Return Value: none

Write a command byte defined by argument *cmd* to the LCD module.

rc_lcddat
Arguments: unsigned char dat
Return Value: none

Write a data byte defined by argument *cmd* to the LCD module.

rc_lprintf
Arguments: char* str
Return Value: none

Write a null terminated character string to the LCD starting at the current cursor position. Use the rc_lcdcmd to set the cursor position.

Temperature Sensor TMP275

The RC has an on-board temperature sensor TMP275 at U15. The sensor is read via a 2-wire serial bus with P24 = Clock and P18 = Data. See sample program *tern\186\samples\rc\tb_tmp* for details on reading the temperature sensor.

tmp_rd
Arguments: unsigned char sck, unsigned char sda,unsigned char t_add
Return Value: int

This function reads the raw data value from the temperature sensor and converts it to a signed integer. The return value represents 16 times the temperature in degrees Celsius. To calculate the actual temperature, divide the return value by 16.

The sck, sda and t_add arguments are used to select the desired sensor on a temperature sensor network. Arguments sck and sda define the PIO lines connected to the clock (sck) and data (sda) lines on the TMP275. The t_add argument is the address of the desired sensor.

Example: The on-board sensor uses the following settings:
Clock (*sck*) = P24
Data (*sda*) = P18
Address (*t_add*) = 0

The following code reads the current ambient temperature from the on-board sensor:

```
current_temperature = tmp_rd( 24, 18, 0 ) / 16;
```

4.6 Other Sample code

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

4.6.1 Analog to digital conversion

Delta-Sigma ADC LTC2448

The LTC2448 ADC chip (U11) with buffers provides 16 channels of ±10V analog single-ended inputs. For details regarding the hardware configuration, see the Hardware chapter.

The following functions will drive the 24-bit ADCs. The order of functions given here should be followed in actual implementation. The following functions are found in the sample program *tern\186\samples\rc\rc_ad24.c*.

void t_init(void);

void rc_ad_setup(unsigned int dat);

unsigned int rc_ad(void);

The control byte, *control_byte*, used in rd_ac_setup drives the LTC2448 in 16 channel single-ended mode with value 0xb000.

In code, the control byte is calculated this way:

ch=0; //select channel

control_byte=control_byte+speed[10]; //add speed desired to 0xb000

control_byte=control_byte+(chl<<8); //add channel selection w/ 8 bit left shift

NOTE: "*ch*" and the desired channel signal do not match up. Instead use this scheme to select the desired signal on the board:

ch_sel	U11
0	N0
1	N2
2	N4
3	N6
4	N8
5	N10
6	N12
7	N14
8	N1
9	N3
10	N5
11	N7
12	N9
13	N11
14	N13
15	N15

The LTC2448 also supports 8 channel differential mode. This can be achieved by changing the control byte passed to the 'ad24_setup' routine to 0xa0000 (speed and channel selection is added on the same way as in single-ended mode). See the LTC2448 data sheet for details on how to define the control byte, '**LTC2448.pdf**' in the **tern_docs\parts** directory.

For a sample file demonstrating the use of the ADC, please see **rc_ad24.c** in **`tern\186\samples\RC`**.

This sample is also included in the **RC.ide** test project in the **tern\186** directory.

4.6.2 Digital to analog conversion

Serial LTC2600

The serial DAC LTC2600 uses a serial interface with the CPU for operation. Three control lines are used, /CS = P24, SCK = SCLK, and SDI = SDAT. Each PIO lines must be initialized as output (mode 2) for operation. The user defined function *da_16* is provided to give a one statement interface with the device. The function can be found in the sample file, ***rb_da.c***, in the directory **\tern\186\samples\rc**.

void da_16
Arguments: unsigned char mode, unsigned int dac
Return value: none

This function drives the DAC at position U13, outputs are V1-V8. The argument *mode* determines which channels are to be written to. The values for *mode* are:

0x30	DAC A
0x31	DAC B
0x32	DAC C
0x33	DAC D
0x34	DAC E
0x35	DAC F
0x36	DAC G
0x37	DAC H
0x3F	All DACs

See the data sheet. From the root of the installation CD, \tern_docs\parts\ltc2600.pdf.

4.6.3 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 RC 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 **rc.ide** for two samples, rc_cf.c and rc_filesys.c.

4.6.4 Communication

USB Port

The RC supports a single slave USB port. The hardware interface for the USB chip includes 384 bytes of FIFO transmit buffer, and 128 bytes of FIFO for the receiving buffer. Accessing the USB port is simply a matter of reading and writing directly to and from the I/O mapped USB chip. The sample program *tern\186\samples\rc\tb_usb.c* demonstrates how to communice over USB. Other information regarding the USB chip is found in:

Tern\tern_docs\manuals\cusb install.pdf //installation instructions for a USB win32 serial driver

Tern\tern_docs\parts\usb //folder contains all data sheet and application notes for usb chip

Ethernet Module

The RC has an optional i2Chip W3100A Ethernet chip. The i2Chip is selected by /MCS0 (P14) and incoming Ethernet events trigger /INT4. Sample programs for the i2Chip are located in the project ***rc.ide*** in ***tern\186\samples\rc*** directory. r***c_wiz.axe*** in the RC sample project demonstrates how to directly communicate with the i2Chip. Sample program ***http_adc.axe*** in ***rc.ide*** is a simple HTTP server that fills an HTML page with ADC values.

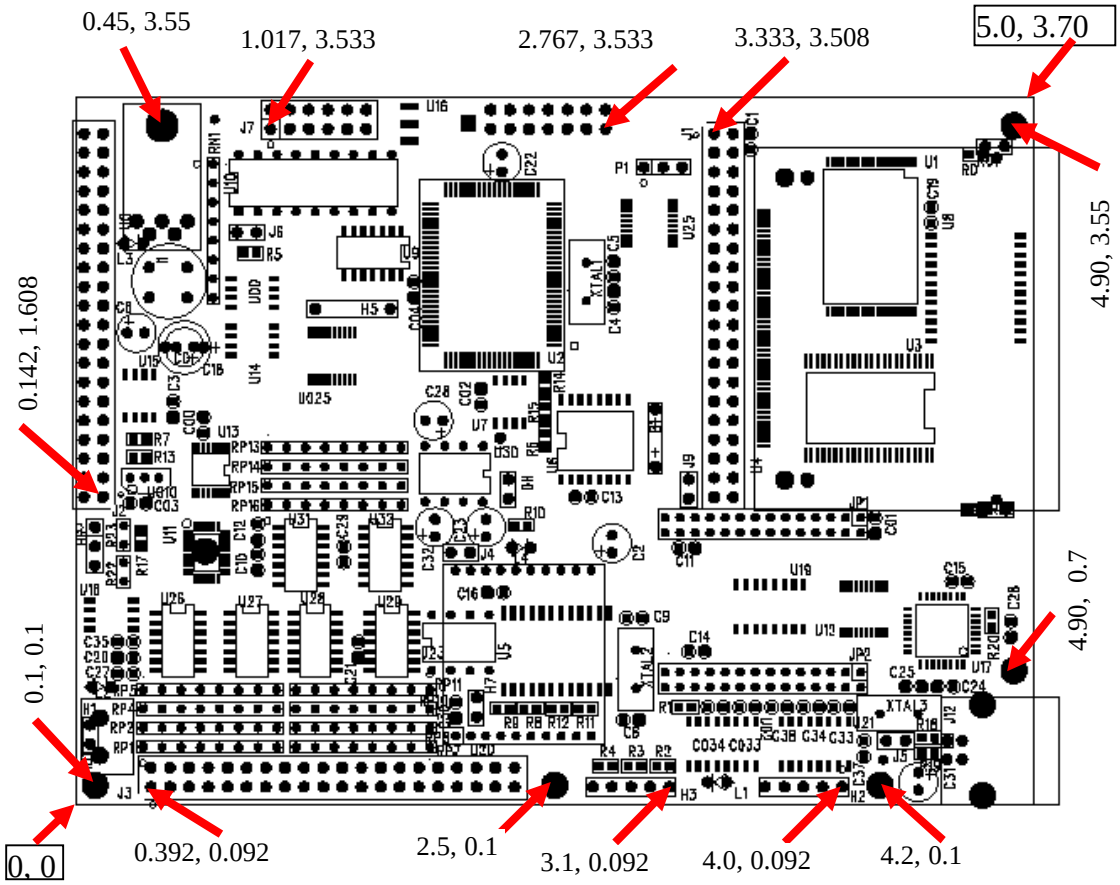
ZigBee Wireless

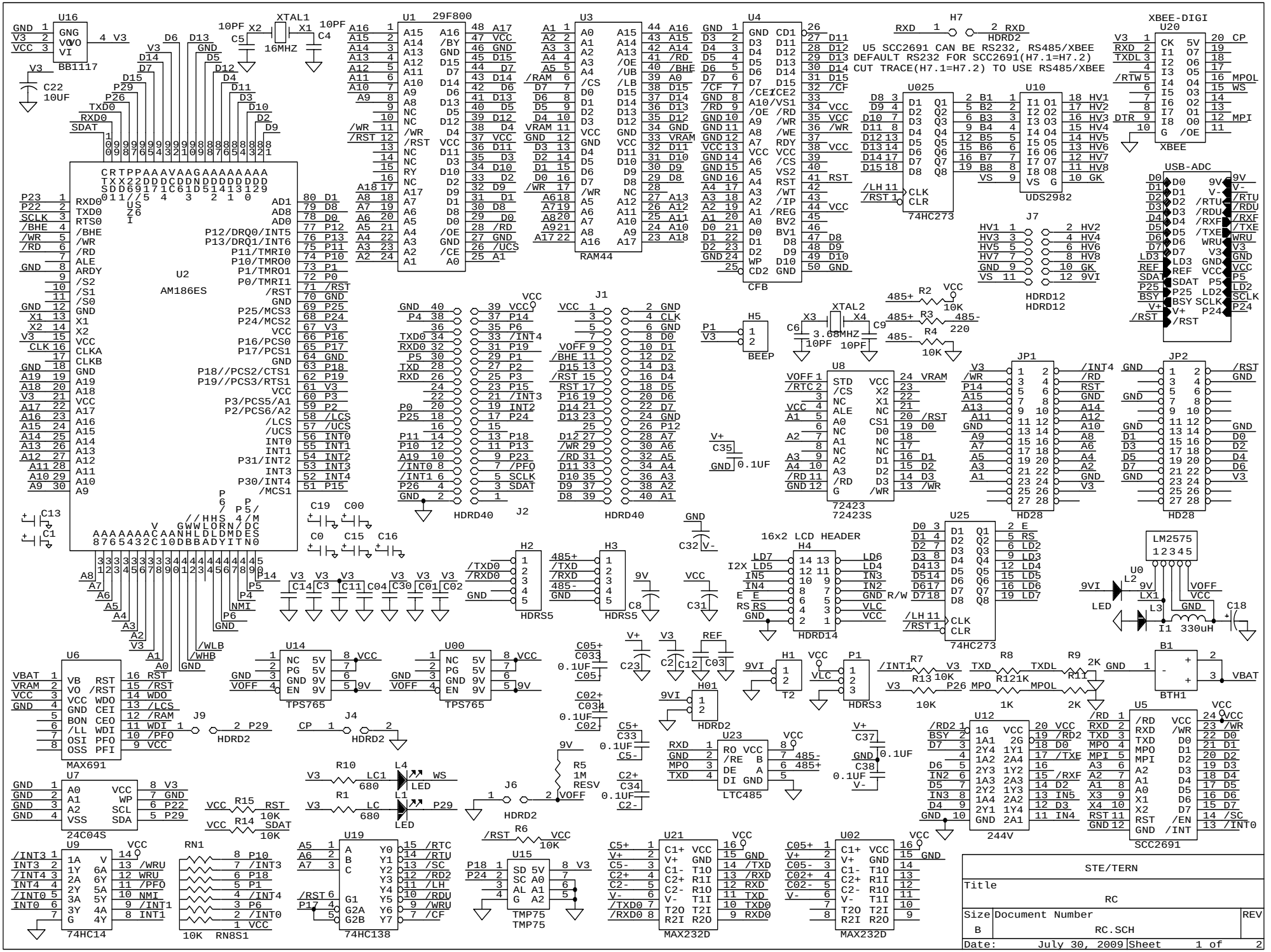
The RC incorporates an interface for the XBee-Pro wireless module. The XBee-Pro RF module meets the IEEE 802.15.4 standards. The RC interfaces the wireless module through the SCC2691 serial port. By default, XBee-PRO RF Module operates in Transparent Mode. When operating in this mode, the module act as a serial line replacement - all UART data received through the DI pin is queued up for RF transmission. When RF data is received, the data is sent out the DO pin. See the XBee-Pro product manual for details on the wireless module. Sample program *tern\186\samples\RC\ser0_scc.c* demonstrates interfacing with the wireless module via RS232.

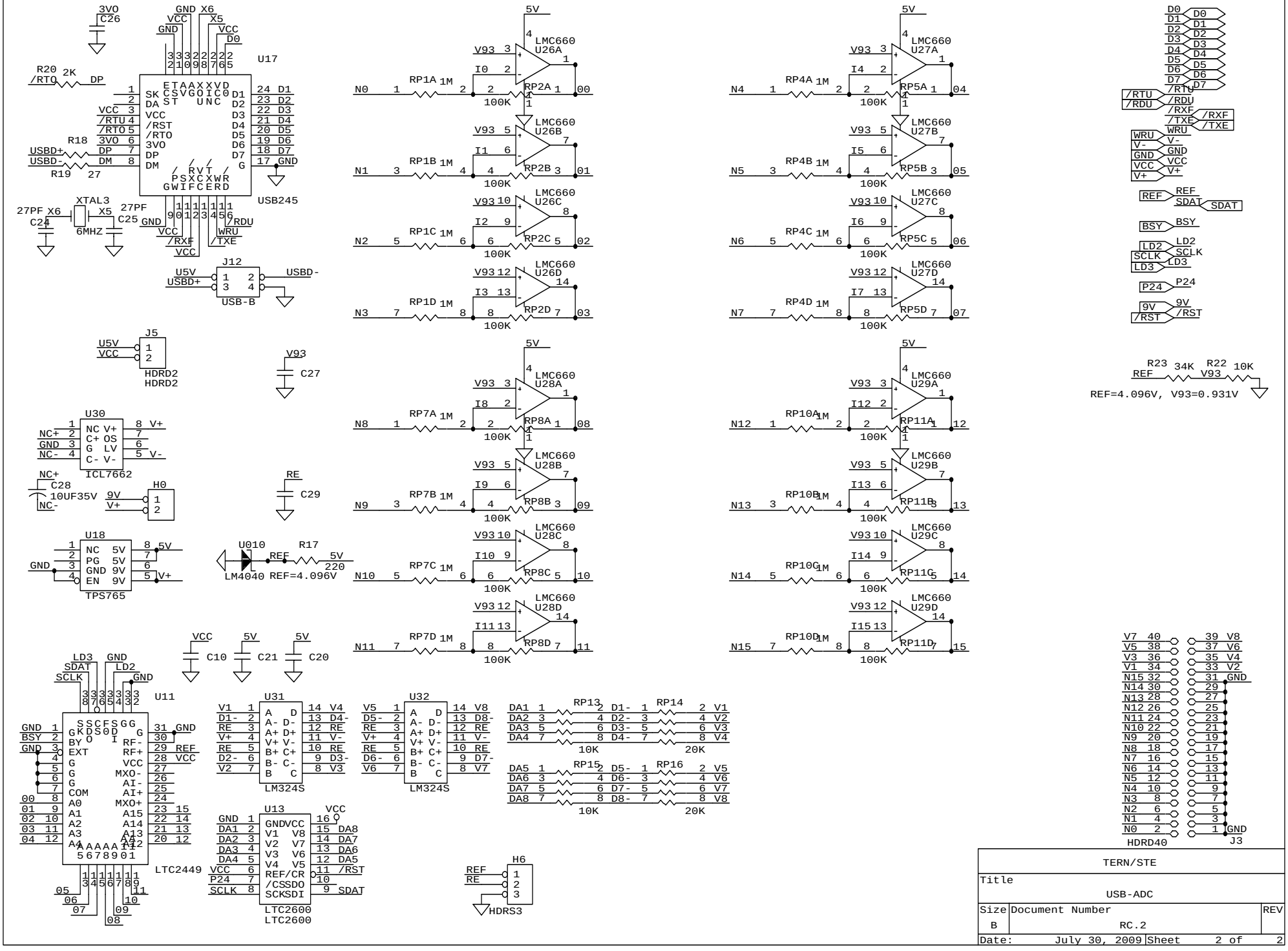
Appendix A: Layout

RC layout mechanical dimensions: (All in inches)

10-1-2009




U2
AM186ES
U1 29F800
U3
RAM44
U4
CFB
U16
BB1117
XTAL1
16MHZ
H7
U5 SCC2691 CAN BE RS232, RS485/XBEE
DEFAULT RS232 FOR SCC2691(H7.1=H7.2)
CUT TRACE(H7.1=H7.2) TO USE RS485/XBEE
XBEE-DIGI
XBEE
U025
74HC273
U10
UDS2982
USB-ADC
J7
HDRD12
U8
72423
72423S
JP1
JP2
HD28
U25
74HC273
16x2 LCD HEADER
H4
HDRD14
U0
LM2575
I1 330uH
BEEP
H5
J1
HDRD40
J2
HDRS5
H2
H3
U14
TPS765
U00
U6
MAX691
J9
J4
HDRD2
U7
24C04S
U23
LTC485
H1
T2
HDRS3
B1
BTH1
U12
244V
U5
SCC2691
U9
74HC14
RN1
10K RN8S1
U19
74HC138
U15
TMP75
U21
MAX232D
U02
MAX232D
STE/TERN
Title
RC
Size B
Document Number
RC.SCH
REV
Date: July 30, 2009
Sheet 1 of 2


3VO
C26
GND X6
VCC X5
GND
VCC
D0
U17
R20 2K
/RTO DP
USB245
R18
USBD+
USBD-
R19 27
XTAL3
27PF X6 X5 27PF
C24 C25
6MHZ
GND VCC /RXF VCC /TXE WRU /RDU
J12
U5V USBD+
USB-B
USBD-
J5
HDRD2
HDRD2
U30
ICL7662
NC+
C28
10UF35V
NC-
H0
9V
V+
U18
TPS765
GND
5V
V+
U010
LM4040
REF
R17
220
REF=4.096V
V93
C27
RE
C29
LMC660
U26A
U26B
U26C
U26D
U27A
U27B
U27C
U27D
U28A
U28B
U28C
U28D
U29A
U29B
U29C
U29D
RP1A 1M
RP1B 1M
RP1C 1M
RP1D 1M
RP4A 1M
RP4B 1M
RP4C 1M
RP4D 1M
RP7A 1M
RP7B 1M
RP7C 1M
RP7D 1M
RP10A 1M
RP10B 1M
RP10C 1M
RP10D 1M
RP2A
RP2B
RP2C
RP2D
RP5A
RP5B
RP5C
RP5D
RP8A
RP8B
RP8C
RP8D
RP11A
RP11B
RP11C
RP11D
100K
R23 34K R22 10K
REF V93
REF=4.096V, V93=0.931V
D0 D1 D2 D3 D4 D5 D6 D7
/RTU /RDU /RXF /TXE WRU V- GND VCC V+
REF SDAT
BSY
LD2 SCLK LD3
P24
9V /RST
C10 C21 C20
VCC 5V 5V
LD3 SDAT SCLK LD2 GND
U11
LTC2449
U31
U32
LM324S
LM324S
RP13 RP14 RP15 RP16
10K 20K
U13
LTC2600
LTC2600
H6
HDRS3
HDRD40
J3
TERN/STE
Title
USB-ADC
Size B
Document Number RC.2
REV
Date: July 30, 2009
Sheet 2 of 2