i386-Engine™

C/C++ Programmable, 32-bit Microprocessor Module
Based on the Intel386EX

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*
Internet Email: sales@tern.com ***http://www.tern.com***

Version 2.00

October 28, 2010

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 ***i386-Engine*** (IE) is a 32-bit microprocessor core module based on the 33 MHz Intel386EX. It is designed for embedded systems that require high performance, PC-compatibility, compactness, low power consumption, and high reliability at a low cost. The ***i386-Engine*** is not only easy to program in C/C++, but also offers PC-compatibility, and thus a vast software library.

Users of TERN's A-Engine, C-Engine, or V25-Engine will find the ***i386-Engine*** to be an ideal upgrade, with increased functionality and performance provided within the same mechanical dimensions (3.6 x 2.3 x 0.3 inches), as well as compatible pin outs and software drivers. Low-cost C/C++ Evaluation (EV) and Development (DV) Kits are available, as well as a 386 Upgrade Kit for V25/186 users.

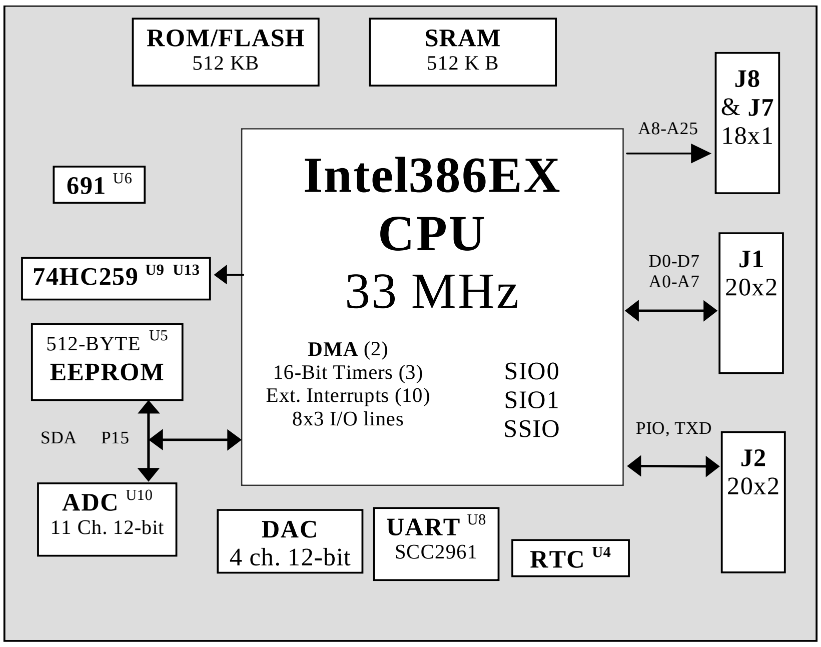


Figure 1.1 Functional block diagram of the i386-Engine

The ***i386-Engine*** operates with regulated 5V-only power input, and consumes 270 mA at 33 MHz. In software-programmable power-down mode, power consumption is 28 mA. The real-time clock can switch an external power supply via the "VOFF" pin on the J1 header to achieve µA-level power consumption. The ***i386-Engine*** can operate in either 8-bit or 16-bit external data mode. Up to 512KB 8-bit ROM/Flash and up to 512KB 8-bit battery-backed SRAM can be installed on-board. 64MB of memory space is supported, with 26 address lines and 16 data lines. The on-board 512-byte serial EEPROM, which does not require a battery backup, can be used as an additional memory device to store important data such as user IDs, calibration coefficients, etc. An optional real-time clock (RTC) provides information on the year, month, date, hour, minute, second, 1/64 second, and an interrupt signal. A PCB-mounting lithium coin battery can be installed to back up both the SRAM and RTC. Under normal conditions, the battery can last at least three years.

Two asynchronous serial ports from the i386EX support reliable DMA-driven serial communication at up to 115,200 baud and provide complete handshaking and modem control signals. One synchronous serial port from the i386EX can interface to external synchronous serial devices at a rate of up to 8 MHz. An optional UART SCC2691 can be added for a third asynchronous serial port that supports 8-bit or 9-bit networking communication.

Three PC-compatible 16-bit programmable timers/counters can operate in one of six modes. They can be used to generate interrupts or count external events, at a rate of up to 8 MHz, or to generate pulse outputs. Three 8-bit multifunctional, user-programmable I/O ports support up to 10 external interrupts. Four external interrupts are buffered by Schmitt-trigger inverters and provide active low inputs. The other six interrupts provide active high inputs. A supervisor chip (LTC691) with power-failure detection, a watchdog timer, and a red LED are on-board.

The optional 12-bit ADC has 11 channels of analog inputs with sample-and-hold and a high-impedance reference input. With an analog input range of single ended 0-5V (or 0 to REF), the ADC supports conversion up to a sample rate of 10KHz. Up to four channels of 12-bit DAC that provide 0-4.095V analog voltage outputs capable of sinking or sourcing 5mA are also available.

On-board expansion headers provide data lines, address lines, control signals, and pre-decoded chip select lines for user expansion. The optional VE232 add-on board provides regulated 5V power and RS232/RS485 drivers.

The i386-Engine can be installed on the MotionC, BirdBox, PowerDrive, SensorWatch, PC-Co, or LittleDrive (see Figure 1.2).

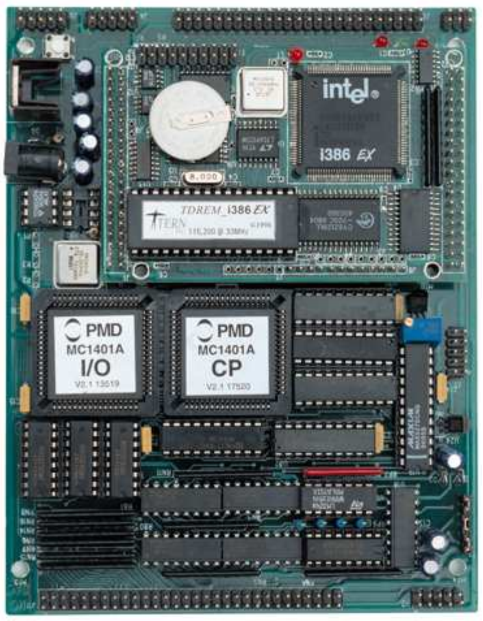
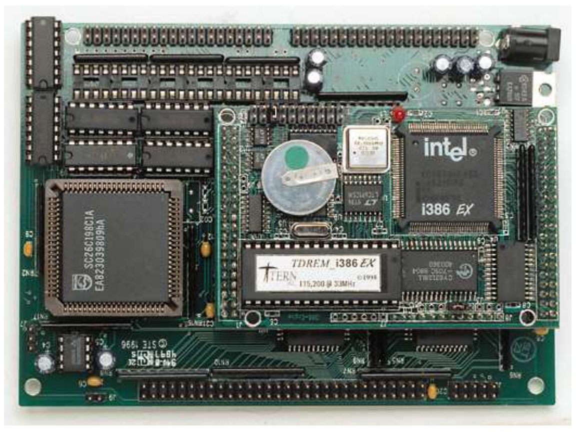



Figure 1.2 i386-Engine installed on the MotionC (left) and on the LittleDrive (right)

1.2 Features

Standard Features

- Dimensions: 3.6 x 2.3 x 0.3 inches
- 32-bit CPU (Intel i386EX), PC-compatible, C/C++ programmable
- Power consumption: 270 mA normal, 28 mA in power-down mode
- Power input: +5V regulated DC
- On-board 8-bit configuration of up to 512KB RAM and up to 512KB ROM
- A total of 64MB memory space, with 16 data lines and 26 address lines
- Three 16-bit timer/counters and a watchdog timer
- Two PC-compatible asynchronous serial ports and one synchronous serial port
- Three 8-bit I/O ports with multiplexed functions from i386EX
- Up to 10 external interrupts and 8 internal interrupts.
- 512-byte EE and supervisor chip for power failure, reset and watchdog
- Two DMA channels for data transfer between memory and I/O

Optional Features (* surface-mounted components):

- 32KB, 128KB, or 512KB SRAM*
- 11 channels of 12-bit ADC, sample rate up to 10 KHz*
- 4 channels of 12-bit DAC, 0-4.095V output *
- SCC2691 UART (on-board) supports 8-bit or 9-bit networking
- Real-time clock RTC72423, lithium coin battery*
- VE232 add-on board for regulated 5V power & RS232/RS485 drivers

1.3 Physical Description

The physical layout of the i386-Engine is shown in Figure 1.3. For a more detailed layout with dimensions, please see Appendix A.

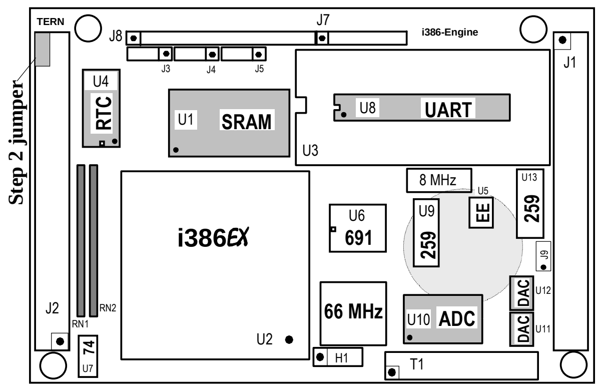


Figure 1.3 Layout of the i386-Engine

Shaded components (such as the ADC in U10) are optional and therefore may not be present on your board. First-time users should have a DEBUG ROM, labeled *TDREM_i386EX*, in the U3 socket.

1.4 i386-Engine Programming Overview

You can program the i386-Engine from your PC via serial link with an RS232 interface. Your C/C++ program can be remotely debugged over the serial link at a rate of 115,000 baud. The C/C++ Evaluation Kit (EV) or Development Kit (DV) from TERN provides a Borland C/C++ compiler, TASM, LOC31, Turbo Remote Debugger, I/O driver libraries, sample programs, and batch files. These kits also include a DEBUG ROM (*TDREM_i386EX)* to communicate with Turbo Debugger, a PC-V25 cable to the connect the controller to the PC, and a 9-volt wall transformer. *See your Evaluation/Development Kit Technical Manual for more information on these kits.*

After you debug your program, you can test run the i386-Engine in the field, away from the PC, by changing a single jumper, with the application program residing in the battery-backed SRAM. When the field test is complete, application ROMs can be produced to replace the DEBUG ROM. The .HEX or .BIN file can be easily generated with the makefile provided. You may also use the DV Kit or ACTF Kit to download your application code to the on-board Flash.

Development of application software for the i386-Engine consists of three easy steps, as shown in the block diagram below.

STEP 1 Serial link PC and i386-Engine (IE), program in C/C++.
Debug C/C++ program on the IE with Remote Debugger

$\updownarrow$

STEP 2 Test IE in the field, away from PC.
Application program resides in the battery-backed SRAM.

$\downarrow$

STEP 3 Make application ROM or Download to Flash.
Replace DEBUG ROM, project is complete.

The three steps in the development of a C/C++ application program are explained in detail below.

STEP 1: Debugging

- Write your C/C++ application program in C/C++.
- Connect your controller to your PC via the PC-V25 serial link cable.
- Use the batch file `m.bat` to compile, link, and locate, or use `t.bat` to compile, link locate, download, and debug your C/C++ application program.

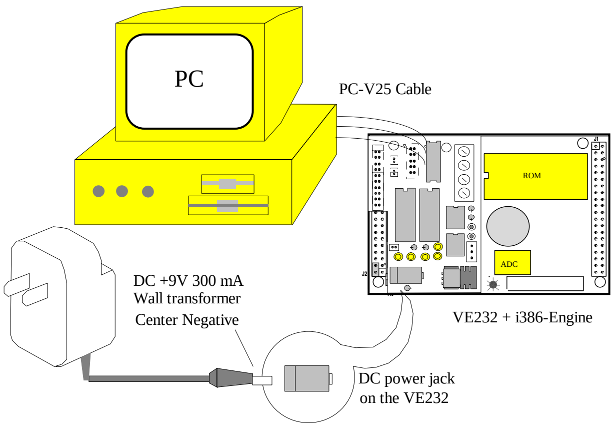


Figure 1.4 STEP 1 connections for the i386-Engine

STEP 2: Standalone Field Test.

- Set the Step 2 jumper on H2 pins 1-2 on the VE232 (Figure 1.6),

or, if you are not using the VE232,

- Set the jumper on J2 pins 38-40 on the i386-Engine (Figure 1.5).
- At power-on or reset, if J2 pin 38 (P4) is low, the CPU will run the code that resides in the battery-backed SRAM.
- If a jumper is on J2 pins 38-40 at power-on or reset, the i386-Engine will operate in Step Two mode. If the jumper is off J2 pins 38-40 at power-on or reset, the i386-Engine will operate in Step One. The status of J2 pin 38 (signal RI1) of the i386EX is only checked at power-on or at reset.

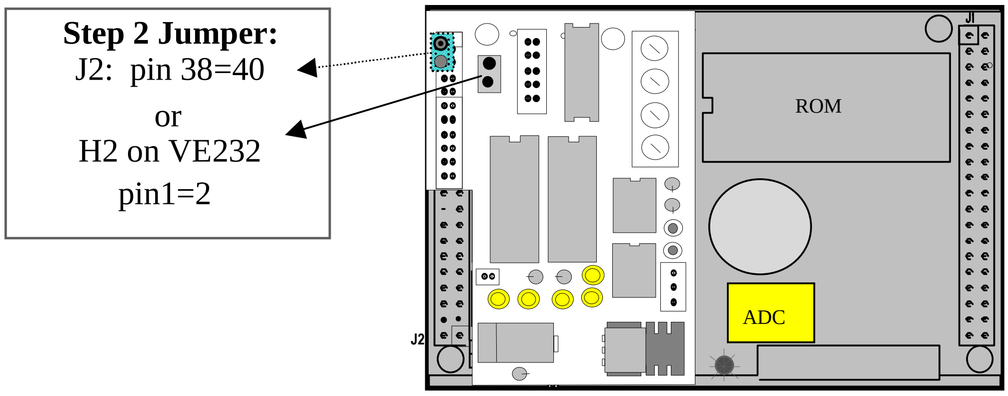


Figure 1.5 Locations for Step 2 jumpers

STEP 3: Generate the application .BIN or .HEX file, make production ROMs or download your program to FLASH via ACTF.

- If you are happy with your Step Two test, you can go back to your PC to generate your application ROM to replace the DEBUG ROM (*TDREM_i386EX*). You need to change ***DEBUG=1*** to ***DEBUG=0*** in the makefile.

You need to have the DV Kit to complete Step 3.

Please refer to the Tutorial of the Technical Manual of the EV/DV Kit for further details on programming the i386-Engine.

1.5 VE232 add-on board

The VE232 is an interface add-on board for the i386-Engine that provides regulated +5V DC power and RS232/485 drivers. It converts TTL signals to and from RS232 signals. You do not need the VE232 if you are using the i386-Engine installed on another TERN controller such as the BirdBox, LittleDrive, MotionC, or PowerDrive.

The VE232, shown in Figure 1.6, measures 2.3 x 1.57 inches. A wall transformer (9V, 300 mA) with a center negative DC plug (Ø=2.0 mm) should be used to power the i386-Engine via the VE232. The VE232 connects to the i386-Engine via H1 (2x10 header). SER0 (J2) and SER1 (J3) on the VE232 are 2x5-pin headers for serial ports SER0 and SER1. SER0 is the default programming port.

While the VE232 is installed, J2 pins 38-40 of the i386-Engine are connected to H2 of the V232. You may use H2 for the Step 2 jumper.

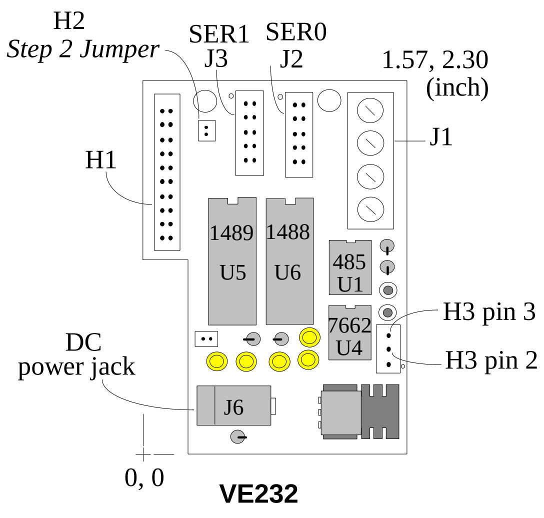


Figure 1.6 The VE232, an interface card for the i386-Engine

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

1.6 Minimum Requirements for i386-Engine System Development

1.6.1 Minimum Hardware Requirements

- PC or PC-compatible computer with serial COMx port that supports 115,200 baud
- i386-Engine controller with DEBUG ROM *TDREM_i386EX*
- VE232 interface board (provides regulated +5V DC power and RS232/RS485 drivers)*
- PC-V25 serial cable (RS232; DB9 connector for PC COM port and IDC 2x5 connector for controller)
- center negative wall transformer (+9V DC, 500mA)

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

1.6.2 Minimum Software Requirements

- TERN EV/DV Kit installation diskettes
- PC software environment: DOS, Windows 3.1, Windows95, or Windows98

The C/C++ Evaluation Kit (EV) and C/C++ Development Kit (DV) are available from TERN. The EV Kit is a limited-functionality version of the DV Kit. With the EV Kit, you can program and debug the i386-Engine in Step One and Step Two, but you cannot run Step Three. In order to generate an application ROM/Flash file, make production version ROMs, and complete the project, you will need the Development Kit (DV).

Chapter 2: Installation

2.1 Software Installation

Please refer to the Technical Manual for the **"C/C++ Development Kit for TERN 16-bit Embedded Microcontrollers"** for information on installing software.

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

2.2 Hardware Installation

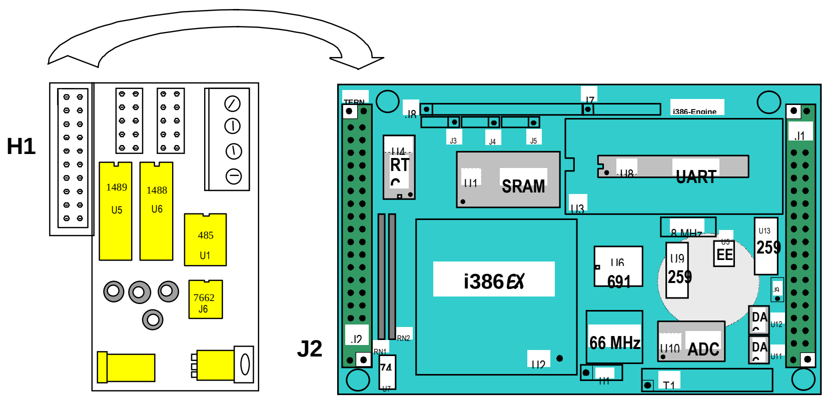
Overview

- Install VE232:
  H1 connector of VE232 installs on J2 of the i386-Engine
- Connect PC-V25 cable:
  For debugging (Step One), place ICD connector on J2 (SER0) with red edge of cable at pin 1
- Connect wall transformer:
  Connect 9V wall transformer to power and plug into power jack on VE232

Hardware installation for the i386-Engine consists primarily of connecting the microcontroller to your PC. The VE232 must be used to supply regulated power and RS232 drivers to the i386-Engine. If you are using the i386-Engine installed on another controller, please refer to the technical manual for that controller for installation information.

2.2.1 Connecting the VE232 to the i386-Engine

Install the VE232 interface with the H1 (10x2) socket connector on the upper half of the J2 (dual row header) of the i386-Engine. Figure 2.1 and Figure 2.2 show the VE232 and the i386-Engine before and after



installation.

Figure 2.1 Before installing the VE232 on the i386-Engine

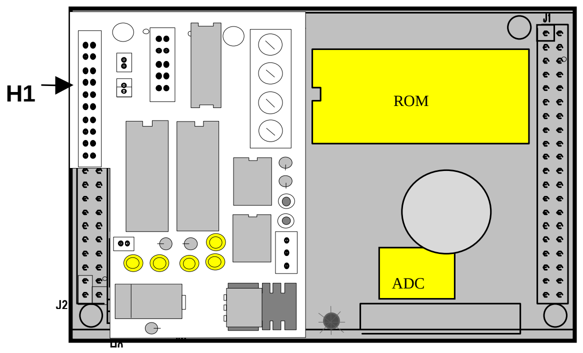


Figure 2.2 After installing the VE232 on the i386-Engine

2.2.2 Connecting the i386-Engine to the PC

The following diagram (Figure 2.3) illustrates the connection between the i386-Engine, VE232 and the PC. The i386-Engine is linked to the PC via a serial cable (PC-V25).

The *TDREM_i386EX* DEBUG ROM communicates through SER0 by default. Install the 5x2 IDC connector to the V232 on the SER0 header. ***IMPORTANT:*** *Note that the* ***red*** *side of the cable must point to pin 1 of the VE232 J2 header.* The DB9 connector should be connected to one of your PC's COM Ports (COM1 or COM2).

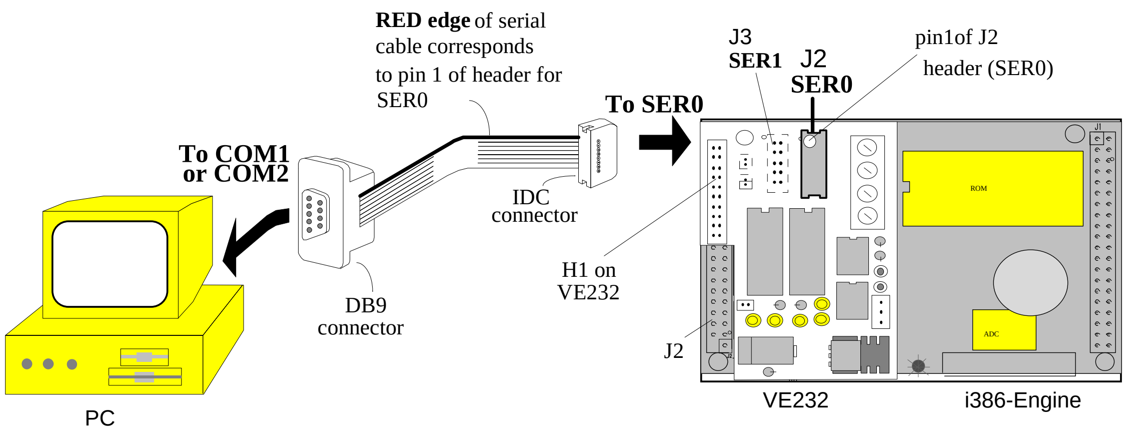


Figure 2.3 Serial connection between the i386-Engine and the PC for debugging (Step One)

2.2.3 Powering-on the i386-Engine

Connect a wall transformer +9V DC output to the VE232 DC power jack. The power supply from the wall transformer is connected to a power jack adapter.

The on-board LED should blink twice and remain on after the i386-Engine is powered on or reset (Figure 2.4).

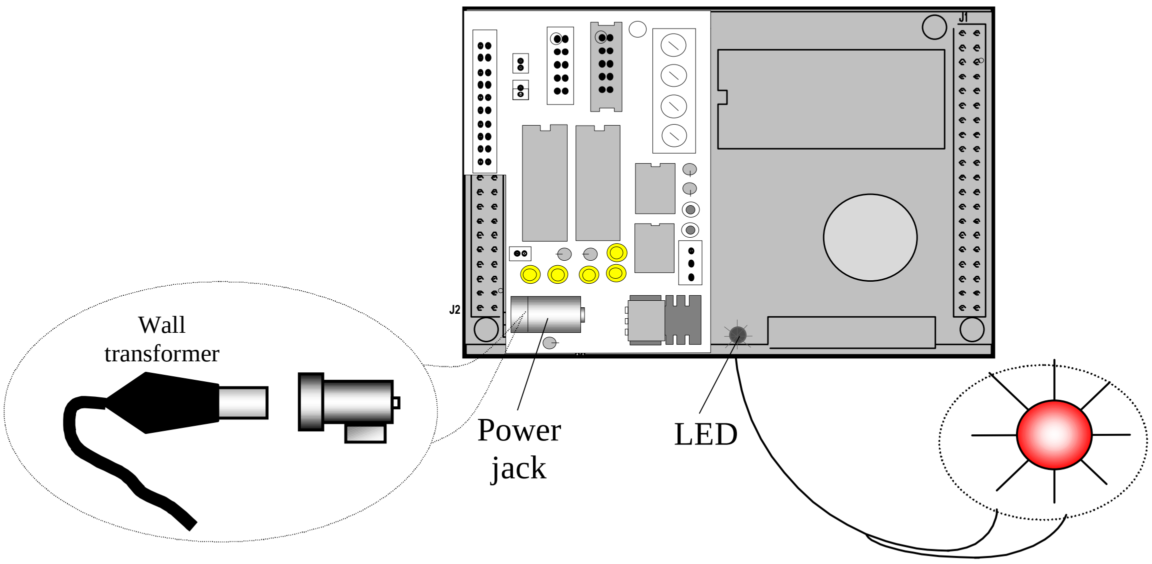


Figure 2.4 The LED blinks twice after the i386-Engine is powered on or reset.

CAUTION: The CPU on the i386-Engine and the power regulator on the VE232 can become **very hot** while the power is connected.

Chapter 3: Hardware

3.1 Intel386EX Processor

The Intel386EX is based on the Intel386SX. This highly integrated device retains PC functions that are useful in embedded applications and adds peripherals that are typically needed in embedded systems. The Intel386EX has new peripherals and an on-chip system interface logic that can minimize total system cost. The Intel386EX has two asynchronous serial ports, one synchronous serial port, 24 I/Os, a watchdog timer, interrupt pins, three 16-bit timers, DMA to and from serial ports, and enhanced chip-select functionality. The i386-Engine provides a PC-compatible development platform optimized for embedded applications.

3.2 Intel386EX I/O Lines

The Intel386EX has 24 I/O lines in three 8-bit I/O ports: P1, P2, and P3. The 24 I/O pins on the Intel386EX are multiplexed with peripheral pin functions, such as serial ports, timer outputs, and chip-select lines. Each of these pins can be used as a user-programmable input or output signal if the normal shared peripheral pin function is not needed. Any I/O line can be configured to operate as a high-impedance input, open-drain output, or complementary output.

After power-on or reset, the I/O 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 in Table 3.1.

PIO	*Peripheral*	*Power-On/Reset*	*i386-Engine Pin No.*	*i386-Engine Initial*
P10	DCD0#	weak pullup	J2 pin 14	Input with pullup
P11	RTS0#	weak pullup	J2 pin 27	Output
P12	DTR0#	weak pullup	J2 pin 18	Input with pullup
P13	DSR0#	weak pullup	J2 pin 20	Input with pullup
P14	RI0#	weak pullup	J2 pin 12	Input with pullup
P15	LOCK#	weak pullup	EE U5.5&ADC U10.16	I/O with pullup
P16	HOLD	Input with pulldown	J2 pin 11	Input with pulldown
P17	HLDA	Output with pulldown	J2 pin 13	Input with pulldown
P20	CS0#	Output with pullup	LT691 U6.13	SRAM select
P21	CS1#	Output with pullup	J2 pin 37	Input with pullup, MM select
P22	CS2#	Output with pullup	J2 pin 5	SCC & RTC I/O select
P23	CS3#	Output with pullup	J2 pin 10	Input with pullup
P24	CS4#	Output with pullup	J2 pin 3	Input with pullup
P25	RXD0	Input with pulldown	J2 pin 32	RXD0
P26	TXD0	Output with pulldown	J2 pin 34	TXD0
P27	CTS0#	Input with pullup	J2 pin 36	Input with pullup
P30	TOUT0	Output with pulldown	J2 pin 17	Input with pulldown
P31	TOUT1	Output with pulldown	J2 pin 19	Input with pulldown
P32	INT0	Input with pulldown	J2 pin 21	Input with pulldown
P33	INT1	Input with pulldown	J2 pin 23	Input with pulldown
P34	INT2	Input with pulldown	J2 pin 24	Input with pulldown
P35	INT3	Input with pulldown	J2 pin 29	Input with pulldown
P36	PWDOWN	Input with pulldown	J2 pin 30	Input with pulldown
P37	COMCLK	Input with pulldown	J2 pin 35	Input with pulldown

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

The 24 PIO lines, P10-P17, P20-P27, and P30-P37 are configurable via 8-bit registers, PnDIR and PnLTC. The value settings are listed as follows:

Pin Configuration	**Desired Pin State**	**PnDIR**	**PnLTC**
High-impedance input	high impedance	1	1
Open-drain output	0	1	0
Complementary Output	1	0	1
Complementary Output	0	0	0

Table 3.2 Value settings for PIO lines

TERN libraries can be used to manipulate these IO pins for you. C functions provided in the library **`ie.lib`** and found in the header file **`ie.h`** can be used to initialize these PIO pins at run-time. Details for these can be found in the Software chapter.

Some of the I/O lines are used by the i386-Engine system for on-board components (Table 3.3). We suggest that you do 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**
P22 = /CS2	J2.5	U4 RTC72423 chip select /RTC
P22 = /CS2	J2.5	U8 SCC2691 chip select /SCC
/CS5	(N/A)	U9 74HC259 (decoder) chip for internal signals T0 to T7
RI1	J2.38	STEP 2 jumper
P15	U5.5	EEPROM SDA = U10.16 ADC DOUT Shared with U10 TLC2543 ADC and U5 24C04 EE data input The ADC and EE data output can be tri-state, while disabled
P20 = /CS0	(N/A)	U6.13 for SRAM chip select, base memory address 0x0000
P26 = TxD0	J2.34	SER0 transmit for default debug ROM
P25 = RxD0	J2.32	SER0 receive for default debug ROM
/INT5	J2.8	i386-Engine U8 SCC2691 UART interrupt.

Table 3.3 Functions of reserved I/O lines on the i386-Engine

At reset, the internal PC/AT-compatible peripherals are mapped into DOS I/O space, of which only 1 Kbyte is used. The DEBUG ROM and **`ie_init()`** enables Expanded I/O space. The registers associated with the integrated peripherals are mapped in the address range of 0f000 to 0f8ffh.

There are four additional external interrupt lines (/INT4, /INT5, /INT6, /INT7) which are not shared with PIO pins. These active-low-only lines are all buffered by Schmitt-triggers. For further details regarding these external interrupt pins, refer to the External Interrupt section below (3.3).

The specifications for these I/O pins state that they can sink up to 8 mA.

If you need further details regarding the Input/Output Ports, please refer to Chapter 16 of the Intel386EX Embedded Microprocessor User’s Manual.

3.3 External Interrupts and Schmitt Trigger Input Buffer

There are 10 external interrupt inputs that the user can adapt for his/her own use.

The master interrupt controller 82C59A supports six ACTIVE HIGH pins on the header **J2**:

INT0 = P32 = J2.21, vector=0x41
INT1 = P33 = J2.23, vector=0x45

INT2 = P34 = J2.24, vector=0x46
INT3 = P35 = J2.29, vector=0x47, IR7 share with Spurious Interrupts
INT8 = P31 = J2.19, vector=0x43 share with SIO1
INT9 = P30 = J2.17, vector=0x44 share with SIO0

The slave interrupt controller 82C59A has six pins, ACTIVE LOW at J2 header:

/INT4 = J2.33, vector=0x48
/INT5 = J2.8, vector=0x49
/INT6 = J2.6, vector=0x4c
/INT7 = J2.15, vector=0x4e

The WDTOUT (Watchdog Timer) interrupt uses vector=0x4f, and the NMI (Non-Maskable Interrupt) at pin J2.7 uses vector=0x2. The NMI interrupt can not be disabled by software, and is raised on a rising edge. /INT5, J2 pin 8, is used by the on-board optional SCC2691 UART if installed.

You must provide a low-to-high (rising) edge to generate an interrupt for the ACTIVE HIGH interrupt inputs and a high-to-low (falling) edge to generate an interrupt for the ACTIVE LOW interrupt inputs.

A spurious interrupt is defined as an interrupt that is "Not Valid." A spurious interrupt on any IR line generates the same vector number as an IR7 request. The spurious interrupt, however, does not set the in-service bit for IR7. Therefore, an IR7 interrupt service routine must check the interrupt service routine register to determine if the interrupt source is either a valid IR7 (the in-service bit is set) or a spurious interrupt (the in-service bit is cleared).

Four external interrupt inputs, /INT4-7, are buffered by Schmitt-trigger inverters (U7) in order to increase noise immunity and transform slowly-changing input signals to fast-changing and jitter-free signals.

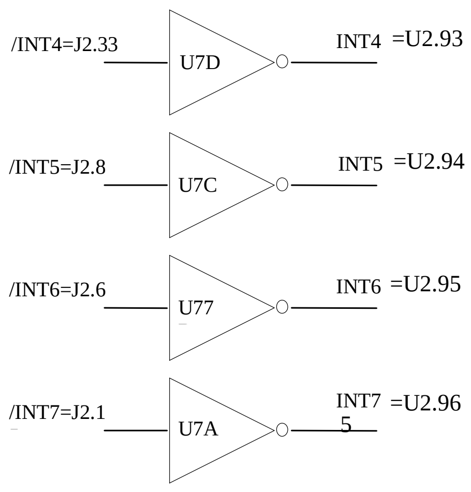


Figure 3.1 External interrupt inputs

The i386-Engine uses vector interrupt functions to response to external interrupts. Please refer to the Intel386EX User's Manual for detailed information about interrupt vectors, and to the Software chapter of this manual (Chapter 4) on how to associate these interrupt vectors with your own interrupt service routine.

3.4 Timer Control Unit

The timer/counter unit has three 16-bit programmable counters: timer0, timer1, and timer2. They can be driven by a pre-scaled value of the processor clock or by external timers. The counters support six different operating modes. Only mode2 and mode3 are periodic modes, in which the counters are reloaded with the user-selected count value when they reach terminal count. For details regarding the modes in which the timers operate, please refer to Chapter 10 of the Intel386EX manual.

The timers provided can be used in several applications. They can be used to act as counters, generate interrupts, and to output repeating pulses with user-specified widths.

Timers can generate pulse outputs at the J1/J2 headers:

Timer 0 output=TOUT0=P30=J2 pin 17
Timer 1 output=TOUT1=P31=J2 pin 19
Timer 2 output=TOUT2=J1 pin 4

Timers can use internal or external clock as clock inputs.

To count external events, the timer clock inputs are routed to the J2 headers:

Timer 0 clock in=/INT4=J2 pin 33
Timer 1 clock in=/INT6=J2 pin 6
Timer 2 clock in=TCLK2=J2 pin 9

These timers can be used to count or time external events.

To use the timers to generate interrupts, a few different options are available. Timer 1 has its output signal, **OUT1**, connected to IR2 of the slave 82C59. The Timer 2 output, **OUT2**, is connected to IR3 of the slave 82C59. The Timer 0 output, **OUT0**, is connected to IR0 of the master 82C59.

The maximum external pulses input rate is 8.25 MHz. Please see the sample program **`timer.c`** and **counter0.c** in **`tern\386\samples\ie`** for details regarding the timers, counters, and their applications.

3.5 Clock

With an on-board 66 MHz oscillator, the i386-Engine operates at 33 MHz system processor clock speed. The 66 MHz clock signal is routed to a 4-pin header H1 pin 1, next to the oscillator. The processor clock is used by serial ports and timers. The default SERCLK for serial ports is 16.5 MHz, and the default pre-scaled PSCLK for the timers is 16.5 MHz. The maximum timer output is 8.25 MHz. For details regarding how to change the PSCLK pre-scale register, see the sample programs **`timer.c`** and **`counter0.c`** in **`\386\samples\ie`**.

3.6 Serial Ports

The i386-Engine has three asynchronous serial channels. Two are Intel386EX-internal: SER0, SER1. One external UART SCC2691 is located underneath the ROM. They can operate in full-duplex communication mode. The SER0 and SER1 use DMA for receiving and for interrupt-driven transmit. The UART SCC2691 is interrupt-driven for both transmitting and receiving. For more information about the external UART SCC2691, refer to Appendix C.

With the DEBUG ROM (*TDREM_i386EX*) installed, the internal serial port SER0 is used by the i386-Engine for DEBUG programming with the PC. It uses 115,000 Baud rate, as default, for programming. It is possible to use both SER0 and SER1 in applications. The user can use SER0 to debug an application program for SER1, and then convert the SER1 code to SER0, since they are identical. The application programs can be combined and downloaded via SER0 in STEP1, and then run in STEP2. Application programs can use both SER0 and SER1 at the same time, but it cannot be debugged over SER0 at the same time.

Complete interrupt/DMA-driven software serial port drivers are included in the EV/DV Kit. Please refer to Chapter 4 (Software) for more details regarding the implementation of the serial port drivers, as well as their application.

3.7 Power-Save-Mode

The i386-Engine is an ideal core module for low power consumption applications. The power-save mode of the Intel386EX processor 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 rate. When an interrupt occurs, it automatically returns to its normal operating rate.

The RTC72423 on the i386-Engine has a VOFF signal routed to J1 pin 9. The VOFF is controlled by the battery-backed RTC72423. It will be in tri-state for the external power-off and become active-low at the programmed time interrupt. The user may use the VOFF line to control an external switching power supply that turns the power supply on/off.

3.8 Memory Map for RAM/ROM

The Intel386EX supports a memory space of up to 64 MB with 26 address lines (A0-A25).

At power-on, the i386-Engine operates in Real-mode, which offers only 1 MB of memory space using segmentation. The DEBUG ROM operates in Real-mode as well, and does not use A20-A25.

The lower memory chip select /CS0 is mapped into memory space of 0x00000 to 0x7ffff. This is used for up to 512K of SRAM. The default wait state on the RAM is set to 3 cycles, but can be shortened if desired.

The upper memory chip select /UCS is mapped into memory space of 0x80000 to 0xfffff and is used for up to 512K of ROM. Both ROMs and Flash chips can be placed in the ROM socket. The default wait state for this component is two cycles, to allow use with ROM components with speed up to 120ns. The preferred ROM speed is 70ns, and if your environment is relatively noise free, you can reduce the wait state to one cycle using this component.

For details regarding how these components are initialized in **`ie_init()`** with these specifications, please refer to the chapter on Software.

In certain applications, you might also choose to re-map the memory address space differently to other chip select lines. This might become useful if you have off-board memory components you also wish to access using ***poke/peek***. Please see the sample file ***ie_cs16.c*** in **tern/386/samples/ie/** for an example of this application.

During development, your code and data segments will be mapped to specific locations within this memory space. Details regarding how this is done during product development can be found in the Technical Manual of the Evaluation/Development Kit.

3.9 I/O Mapped Devices

3.9.1 I/O Space

External I/O devices can use I/O mapping for access. You can access such I/O devices with ***inportb(port)*** or ***outportb(port,dat).*** These functions will transfer one byte of data to the specified I/O address.

The external I/O space size is 64KB, ranging from 0x0000 to 0xffff.

The default I/O access time is 15 wait states. You may modify the wait states by re-programming the Chip-select Low Address register from 0-15 cycles. The system clock speed is 33 MHz. Details regarding this can be found in the Software chapter, and in the Intel386EX Embedded Microprocessor User's Manual. Slower components, such as most LCD interfaces, might find the maximum programmable wait state of 15 cycles still insufficient.

For details regarding the chip select unit, please see Chapter 14 of the Intel386EX Embedded Microprocessor User's Manual.

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

I/O space	Select Signal	Location	Usage
0x8000-0x80ff	/CS6	J1 pin 19 = /CS6	User

I/O space	Select Signal	Location	Usage
0xa090-0xa09f	/CS2	J2 pin 5 = P22	UART, SCC2691
0xa0a0-0xa0af	/CS2	J2 pin 5 = P22	RTC 72423
0xb000-0xb0ff	/CS5	None (**U9**-74HC259)	Internal Usage (T0-T7)
Not mapped	/CS0	N/A	SRAM
Not mapped	/CS1, /CS3	J2 pin 37 = P21, J2 pin 10 = P23	Reserved for future **TERN** use
Not mapped	/CS4	J2 pin 3 = P24	User

A total of eight pre-decoded chip-select lines are available on the IE. These include the UCS (upper chip select), and signals CS0-6. The upper chip select is dedicated for boot-up ROM use. Some others are used for on-board internal usage and not available via I/O mappings, but there are several available for user expansion components.

Chip select lines 1 and 3 (/CS1, /CS3) are used in some special versions of the IE, as well as in some peripheral boards (such as the **MemCard**). If you are sure you are not using TERN controllers that use these chip-select lines, you could also use them for other user external I/O peripherals.

To use one of the chip select lines, you must map the appropriate line to a free base I/O address. After configuring the PIO pin appropriately for this peripheral function (normal-mode operation), you can directly **outport** to that address with appropriate data. The address bus and data bus should then be connected to your I/O component if needed.

To illustrate how to interface the i386-Engine with external I/O boards, a simple decoding circuit for interfacing to an external 82C55 I/O chip is shown.

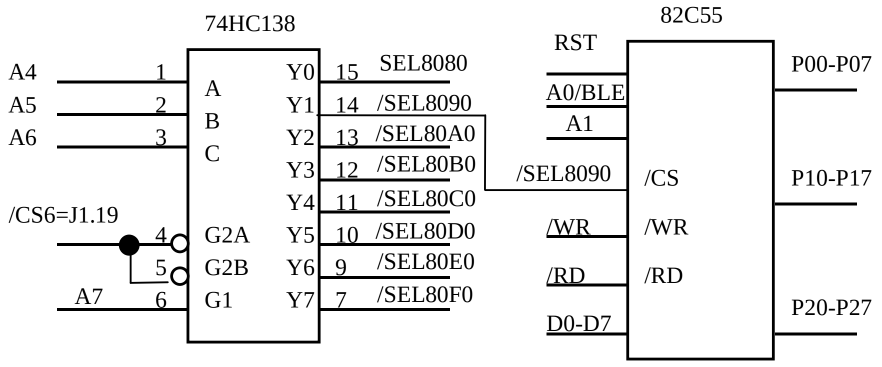


Figure 3.2 Interface i386-Engine to external I/O devices

The function **`ie_init()`** by default initializes the /CS6 line at base I/O address starting at 0x8000. You can read from the 82C55 with ***inportb(0x8090)*** or write to the 82C55 with ***outportb(0x8090,dat).*** The call to **inportb** will activate /CS6, as well as putting the address 0x8090 over the address bus. The decoder will select the 82C55 based on address lines A4-6, and the data bus will be used to read the appropriate data from the off-board component.

3.9.2 Real-time Clock RTC72423

If installed, a real-time clock RTC72423 (EPSON, U4) is mapped in the I/O address space 0xa0a0. It must be backed up with a lithium coin battery. The RTC may be accessed via software drivers ***rtc_init()*** or ***rtc_rd()***; (see Chapter 5, Software for details).

3.9.3 UART SCC2691

The UART SCC2691 (Signetics, U8) is mapped into the I/O address space at **0xa090**. 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. The MPO is routed to J1 pin 3. The MPI is not connected.

For more detailed information, refer to Appendix C. The SCC2691 on the i386-Engine may be used as a network 9-bit UART (for the TERN **NT-Kit**).

The RxD (J1 pin 5), TxD (J1 pin 7), MPO (J1 pin 3), and MPI (J1 pin 21) are TTL-level signals. You must provide RS-232 or RS-485 drivers off of the i386-Engine board if you choose to use either RS-232 or RS-485. This can be achieved by using either the VE232, or a number of other TERN peripheral controllers that are driven by the i386-Engine and offer RS232/RS485 drivers.

The VE232 provides an RS485 driver. You may connect the RxD and TxD signals on J1 of the i386-Engine to H3 pins 3 and 2 on the VE232 and the 485+ and 485- signals at the screw terminal of the VE232 to join a multi-drop RS485 twisted-pair network.

The RS-485 driver on the VE232 was designed to use J2 pin 24 to control the half-duplex RS-485 driver direction for the V25-Engine. However, the i386-Engine J2 pin 24 (signal P34) is the RTC chip select signal. P34 should not be used as RS-485 driver direction control. You can use MPO of the SCC2691 (J1 pin 3) as the RS-485 driver direction control and connect the i386-Engine J1 pin 3 to the VE232's H1 pin, which maps to i386-Engine J2 pin 24. You then should cut off pin 24 of the J2 on the i386-Engine connection to the VE232. Please refer to Appendix B for the layout of the VE232 and to Chapter 1 for further information.

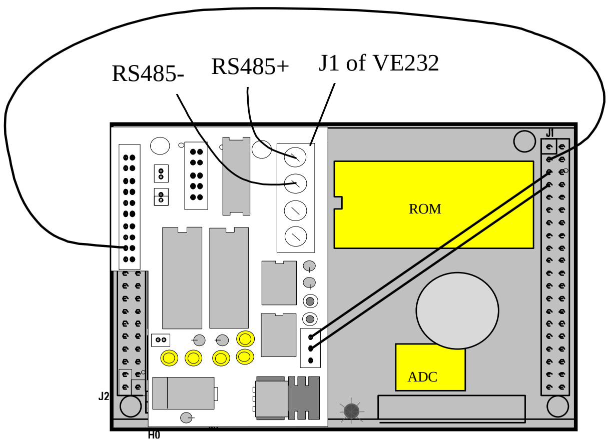


Figure 3.3 Settings and connections for networking with SCC2691

3.10 Other Devices

A number of other devices are also available on the i386-Engine. Some of these are optional, and might not be installed on the particular controller you are using. For a discussion regarding the software interface for these components, please see the Software chapter.

3.10.1 On-board Supervisor with Watchdog Timer

The MAX691/LTC691 (U6) is a supervisor chip. With it installed, the i386-Engine has several functions: watchdog timer, battery backup, power-on-reset delay, power-supply monitoring, and power-failure warning. These will significantly improve the system reliability.

Watchdog Timer

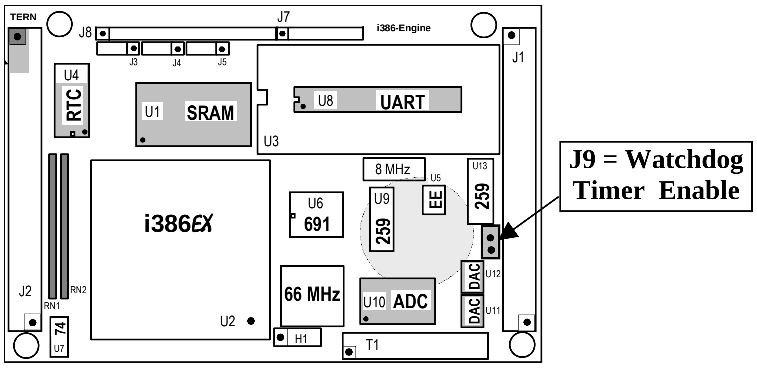


Figure 3.4 Location of watchdog timer enable jumper

The watchdog timer is activated by setting a jumper on J9 of the i386-Engine. 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 T6=HWD pin of the 691) should be arranged so that the HWD pin is accessed at least once every 1.6 seconds. If the J9 jumper is on and the HWD pin is not accessed within this time-out period, the watchdog timer pulls the WDO pin low, which asserts /RESET. This automatic assertion of /RESET may recover the application program if something is wrong. After the i386-Engine is reset, the WDO remains low until a transition occurs at the WDI pin of 691. When controllers are shipped from the factory the J9 jumper is off, which disables the watchdog timer.

The Intel386EX has an internal watchdog timer. This is disabled by default with **ie_init**().

Power-failure Warning and Battery Backup

When power failure is sensed by the on-board supervisor chip 691, it will reset the board if the VCC is less than 4.5V. The battery-switchover circuit compares VCC to VBAT (+3 V lithium battery positive pin), and connects whichever is higher to the VRAM (power for SRAM and RTC). Thus, the SRAM and the real-time clock RTC72423 are backed up. In normal use, the lithium battery should last about 3-5 years without the external power being supplied. When the external power is on, the battery-switch-over circuit will select the VCC to connect to the VRAM.

3.10.2 EEPROM

A serial EEPROM of 128 bytes (24C01), 512 bytes (24C04), or 2Kbytes (24C16) can be installed in U5. The i386-Engine uses the T7=SCL (serial clock) and P15=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 has typically 1,000,000 erase/write cycles. The data retention is more than 40 years. EEPROM can be read and written by simply calling functions **ee_rd**() and **ee_wr**().

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

3.10.3 12-bit ADC (TLC2543)

The TLC2543 is a 12-bit, switched-capacitor, successive-approximation, 11-channel, serial interface, analog-to-digital converter. Three output lines from U9 74HC259 are used to handle the ADC, with /CS=T2; CLK=T0; and DIN=T1.

The ADC digital data output communicates with a host through a serial tri-state output (DOUT=P15). If T2=/CS is low, the TLC2543 will have output on P15. If T2=/CS is high, the TLC2543 is disabled and P15 is free. The TLC2543 has an on-chip 14-channel multiplexer that can select any one of 11 inputs or any one of three internal self-test voltages. The sample-and-hold function is automatic. At the end of conversion, the end-of-conversion (EOC) output goes high to indicate that conversion is complete. On the i386-Engine, this output is not connected.

TLC2543 features differential high-impedance inputs that facilitate ratiometric conversion, scaling, and isolation of analog circuitry from logic and supply noise. A switched-capacitor design allows low-error conversion over the full operating temperature range. The analog input signal source impedance should be less than 50Ω and capable of slewing the analog input voltage into a 60 pF capacitor.

A reference voltage less than VCC (+5V) can be provided for the TLC2543 if additional precision is required. A voltage above 2.5V and less than +5V can be used for this purpose, and connected to the **REF+** pin.

The CLK signal to the ADC is toggled through an I/O pin, and serial access allows a conversion rate of up to approximately 10 KHz.

In order to operate the TLC2543, five I/O lines are used, as listed below:

Pin Label	Description
AIN0-AIN10	11 analog signal inputs. The signal source impedance should be less than 50Ω, and capable of slewing the analog input voltage into a 60pF capacitor.
/CS	Chip select = T2, high to low transition enables DOUT, DIN and CLK, low to high transition disables DOUT, DIN and CLK.
DIN	T1, serial data input
DOUT	P15 of Intel386EX, 3-state serial data output.
EOC	Not Connected, End of Conversion, high indicates conversion complete and data is ready
CLK	I/O clock = T0
REF+	Upper reference voltage (normally VCC)
REF-	Lower reference voltage (normally GND)
VCC	Power supply, +5 V input
GND	Ground

The analog inputs AD0 to AD9 are available at header H0, and can be connected to your signal sources from there. AD10, REF+, GND, VCC are available at header J6.

3.10.4 Dual 12-bit DAC

The LTC1446 is a dual 12-bit digital-to-analog converters (DACs) in an SO-8 package. They are complete with a rail-to-rail voltage output amplifier, an internal reference and a 3-wire serial interface. The LTC1446 outputs a full-scale of 4.096V, making 1 LSB equal to 1 mV.

The buffered outputs can source or sink 5 mA. The outputs swing to within a few millivolts of supply rail when unloaded. They have an equivalent output resistance of 40Ω when driving a load to the rails. The buffer amplifiers can drive 1000pf without going into oscillation.

The DAC is installed in U11 and U12 on the i386-Engine. The outputs are routed to header **T1** pins 17 and 18 for U11 DAC channels A and B. The second DAC installed in U12 has for outputs header **T1** pins 19 and 20.

The DAC uses T0 as CLK, T1 as DI, and T3/4 as LD/CS. Please contact Linear Technology (tel. 408-432-1900) for LT1446 technical data sheets.

3.11 Headers and Connectors

3.11.1 Expansion Headers J1 and J2

There are two 20x2, 0.1 spacing headers for i386-Engine expansion. Most signals are directly routed to the Intel386EX processor. These signals are 5V only, and any out-of-range voltages will most likely damage the board.

J1 Signal			
VCC	1	2	GND
MPO	3	4	TOUT2
RxD	5	6	GND
TxD	7	8	D0
VOFF	9	10	D1
BHE	11	12	D2
D15	13	14	D3
/RST	15	16	D4
RST	17	18	D5
/CS6	19	20	D6
D14	21	22	D7
D13	23	24	GND
M/IO	25	26	A7
D12	27	28	A6
/WR	29	30	A5
/RD	31	32	A4
D11	33	34	A3
D10	35	36	A2
D9	37	38	A1
D8	39	40	BLE

J2 Signal			
GND	40	39	VCC
RI1	38	37	P21
P27	36	35	P37
TxD0	34	33	/INT4
RxD0	32	31	/RTS1
P36	30	29	P35
TxD1	28	27	P11
RxD1	26	25	DTR1
P34	24	23	P33
/CTS1	22	21	P32
P13	20	19	P31
P12	18	17	P30
R/W	16	15	/INT7
P10	14	13	P17
P14	12	11	P16
P23	10	9	TCLK2
/INT5	8	7	NMI
/INT6	6	5	P22
DSR1	4	3	P24
GND	2	1	DCD1

Table 3.4 J1 and J2, 20x2 expansion ports

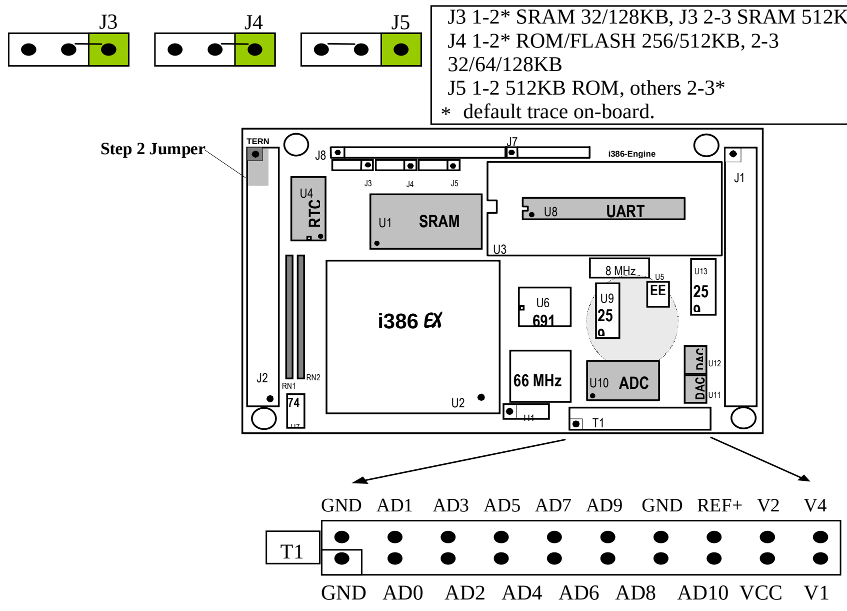
Signal definitions for J1:

VCC	+5V power supply
GND	Ground
TOUT2	Intel386EX pin 91, timer2 output, 8.25 MHz maximum
RxD	data receive of UART SCC2691, U8
TxD	data transmit of UART SCC2691, U8
MPO	Multi-Purpose Output of SCC2691, U8
MPI	Multi-Purpose Input of SCC2691, U8
VOFF	real-time clock output of RTC72423 U4, open collector
D0-D15	Intel386EX 16-bit external data lines
A1-A7	Intel386EX lower address lines
/RST	reset signal, active low
RST	reset signal, active high
/CS6	/CS6, Intel386EX pin 2, ie_init(); set it up as I/O chip select line at address 0x8000
M/IO	Intel386EX pin 27, high for memory, low for I/O operation
BHE	Intel386EX pin 39, high byte enable
/WR	Intel386EX pin 35, active low when write operation
/RD	Intel386EX pin 34, active low when read operation

Signal definitions for J2:

VCC	+5V power supply, < 300 mA
GND	ground
Pxx	Intel386EX PIO pins
R/W	inverted from Intel386EX pin 30, W/R
TxD0	Intel386EX pin 131, transmit data of serial channel 0
RxD0	Intel386EX pin 129, receive data of serial channel 0
TxD1	Intel386EX pin 112, transmit data of serial channel 1
RxD1	Intel386EX pin 118, receive data of serial channel 1
P27=/CTS0	Intel386EX pin 132, Clear-to-Send signal for SER0
/CTS1	Intel386EX pin 113, Clear-to-Send signal for SER1
P11=/RTS0	Intel386EX pin 102, Request-to-Send signal for SER0
/RTS1	Intel386EX pin 110, Request-to-Send signal for SER1
/INT4-7	Schmitt-trigger buffered active low interrupt inputs
P32-35=INT0-3	active high interrupt inputs
TCLK2	timer2 clock input
NMI	Non-mask interrupt
DSR1, DCD1, RI1, DTR1	Serial port 1 handshake lines
RI1	J2 pin 38 Used as Step Two jumper

3.11.2 Expansion Header T1



J3 1-2* SRAM 32/128KB, J3 2-3 SRAM 512KB
J4 1-2* ROM/FLASH 256/512KB, 2-3 32/64/128KB
J5 1-2 512KB ROM, others 2-3*
* default trace on-board.

Figure 3.5 ADC inputs and DAC outputs on the T1 header

The pin layout of the 10x2 pin header T1 on the i386-Engine is as follows:

T1	
Pin 1 = GND	Pin 2 = GND
Pin 3 = AD0	Pin 4 = AD1
Pin 5 = AD2	Pin 6 = AD3
Pin 7 = AD4	Pin 8 = AD5
Pin 9 = AD6	Pin 10 = AD7
Pin 11 = AD8	Pin 12 = AD9
Pin 13 = AD10	Pin 14 = GND
Pin 15 = VCC	Pin 16 = REF+
Pin 17 = V1(da)	Pin 18 = V2 (da)
Pin 19 = V3(da1)	Pin 20 = V4 (da1)

3.11.3 Jumpers and Headers

There are 10 jumpers and connectors on the i386-Engine, as listed below.

Name	**Size**	**Function**	**Possible Configuration**
J1	20x2	main expansion port	
J2	20x2	main expansion port	Pins 38=40, Step 2 jumper
J3	3x1	SRAM selection:	pin 2-3 256K-512KB pin 1-2, 32K-128KB default
J4	3x1	ROM/Flash size selection:	pin 1-2, 32K-128K, default pin 2-3, 256K-512K
J5	3x1	512K ROM selection:	pin 1-2, 512KB ROM pin 2-3, all others, default
J7	6x1	Address lines A20 to A25, pin 1=A25	
J8	12x1	High address lines, A8-A19, pin1=A19	
J9	2x1	Watchdog timer	Enabled if jumper is on Disabled is jumper is off
T1	20x2	11 channels of 12-bit ADC analog inputs and 4 ch. 12-bit DAC outputs.	
H1	4x1	pin1=CLK2, pin2=WDTO, pin3=NC, pin4=GND.	

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 a 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.

inport/inport
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 IE.LIB

IE.LIB is a C library for basic i386-Engine operations. It includes the following modules: IE.OBJ, SER0.OBJ, SER1.OBJ, SCC.OBJ, and IEEE.OBJ. You need to link IE.LIB in your applications and include the corresponding header files. The following is a list of the header files:

Include-file name	Description
IE.H	PIO, timer/counter, ADC, DAC, RTC, Watchdog,
SER0.H	internal serial port 0
SER1.H	internal serial port 1
SCC.H	external UART SCC2691
IEEE.H	on-board EEPROM

4.2 Functions in IE.OBJ

4.2.1 i386-Engine Initialization

ie_init

This function should be called at the beginning of every program running on i386-Engine core controllers. It provides default initialization and configuration of the various I/O pins, interrupt vectors, sets up expanded DOS I/O, and provides other processor-specific updates needed at the beginning of every program.

There are certain default pin modes and interrupt settings you might wish to change. With that in mind, the basic effects of **ie_init** are described below. For details regarding register use, you will want to refer to the Intel386EX Embedded Processor User's manual.

Initialize the upper chip select to support the default ROM. The CPU registers are configured such that:

Address space for the ROM is from 0x80000-0xfffff.

512K ROM operation (this works for the 32K ROM provided, also)

Two 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.

```
outport(0xf43a, 0x0008); // UCSADH, 0x80000-0xfffff, 512K ROM
outport(0xf438, 0x0102); // UCSADL, bs8, 2 wait states
outport(0xf43e, 0x0007); // UCSMSKH
outport(0xf43c, 0xfc01); // UCSMSKL, enable UCS
```

Initialize CS0 for use with the SRAM. It is configured so that:

Address space starts 0x00000, with a maximum of 512K RAM.
8 bit operation with 3 wait states. Once again, you can set the same register to a lower wait state if you desire faster operation.

```
outport(0xf402, 0x0000); // CS0ADH, base Mem address 0x0000
outport(0xf400, 0x0103); // CS0ADL, bs8, 3 wait states
outport(0xf406, 0x0007); // CS0MSKH
outport(0xf404, 0xfc01); // CS0MSKL, 512K, enable CS0 for RAM
```

Initialize the chip select used for RTC and SCC (UART).

The I/O Address for the RTC is at 0xa0a0. (See samples\ie\rtc_init.c and rtc.c for RTC usage.
The I/O Address for the SCC is at 0xa090. (See samples\ie\ie_scc.c).
These are initialized to 16 wait states.

```
outport(0xf412, 0x0280); // CS2ADH, RTC/SCC I/O addr=0xa0a0/0xa090
outport(0xf410, 0x000f); // CS2ADL, 0x000f=16 wait
outport(0xf416, 0x0003); // CS2MSKH
outport(0xf414, 0xfc01); // CS2MSKL, 32 enable CS2=RTC/SCC
```

Initialize chip select U9, which is used for internal signals T0-T7.

I/O address is 0xb000.

```
outport(0xf42A, 0x02c0); // CS5ADH, 259 base I/O address 0xb000
outport(0xf428, 0x0001); // CS5ADL, 0x0001=1 wait
outport(0xf42E, 0x0003); // CS5MSKH
outport(0xf42C, 0xfc01); // CS5MSKL, 256 enable CS5=259
```

This chip select line, CS6, is provided for the user’s use. Many users choose to attach peripheral boards to the headers provided on the controllers. It is possible to attach a 74HC259 decoder, for example, which could then be used to select a number of off-board user components. This line is at pin 19 of header J1. For details regarding this and the other chip select line, refer to the Hardware chapter of this manual.

I/O address for this is 0x8000. A wait-state of 32 has been set initially for easier interface with slower devices. This value can be decreased as well by changing the value of the register.

```
outport(0xf432, 0x0200); // CS6ADH, base I/O address 0x8000
outport(0xf430, 0x001f); // CS6ADL, 0x001f=32 wait
outport(0xf436, 0x0003); // CS6MSKH
outport(0xf434, 0xfc01); // CS6MSKL, 256 enable CS6
```

Configure the three PIO ports for default operation.

```
outportb(0xf820, 0x00); // P1CFG
outportb(0xf822, 0x65); // P2CFG,TXD0,RXD0,CS2=P22=RTC/SCC, 0=RAM
outportb(0xf824, 0x00); // P3CFG
```

Configure serial port 1, DMA, interrupts, timers.

```
outportb(0xf826, 0x1f); // PINCFG,CS5,CTS1,TXD1,DTR1,RTS1
outportb(0xf830, 0x00); // DMACFG
outportb(0xf832, 0x00); // INTCFG
outportb(0xf834, 0x00); // TMRCFG
outportb(0xf836, 0x01); // SIOCFG,SIO0 use SERCLK
```

Configure PIO ports as input

```
outportb(0xf862, 0xff); // P1LTC
outportb(0xf864, 0xff); // P1DIR
outportb(0xf86a, 0xff); // P2LTC
outportb(0xf86c, 0xff); // P2DIR
outportb(0xf872, 0xff); // P3LTC
outportb(0xf874, 0xff); // P3DIR
```

4.2.2 External Interrupt Initialization

The i386-Engine offers two cascaded interrupt controllers to handle internal and external interrupts. Each interrupt controller is functionally identical to a 82C59A. Combined, the cascaded interrupt controllers can handle up to 10 external interrupts, and eight internal interrupts. For a detailed discussion involving the ICUs, the user should refer to Chapter 9 of the Intel386EX Embedded Microprocessor User's Manual. **Figure 9-1**, in particular, shows interrupts that share the same IR and thus cannot be used at the same time.

You should note that if an IR on the slave 82C59 is activated, IR2 on the master must also be activated before the interrupt handler is called.

TERN provides functions to enable/disable all of the 10 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.

If you are dealing with external interrupts, you might need to disable the particular interrupt being handled while processing within the interrupt service routine. The interrupt control unit is sensitive to certain non-qualified external interrupts that come from sources such as mechanical switches. In such a situation, repeated interrupts (in the thousands) might be generated, crashing the system. Disabling such an interrupt for a length of time will make sure that you isolate such interrupts.

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. Thus, 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.

On the i386-Engine, the overhead of executing the interrupt service routine is approximately 30 µs using a 33 MHz controller.

To send the nonspecific EOI command, you need to write the **OCW2** word with 0x20 (see **Figure 9-14** in the Intel386EX manual for details regarding this command word).

To clear the master 82C59, you will need to do:

```
outportb(0xf020, 0x20);
```

If the IR that has just been handled is on the slave 82C59, you must clear its in-service bit first. After this, you must also send another Nonspecific EOI command to the master 82C59, since the slave interrupt was only transmitted to the core after IR2 on the master 82C59 was raised. So, you will need to have code similar to:

```
outportb(0xf0a0, 0x20) ;
outportb(0xf020, 0x20) ;
```

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.
By default, the interrupts are all disabled after initialization. To disable them again, you can repeat the call but pass in 0 as the first argument.

The NMI (Non-Maskable Interrupt) is special in that it can not be masked (disabled). The default ISR will return on interrupt.

```
void int0_init( unsigned char i, void interrupt far(* int0_isr)() );
void int1_init( unsigned char i, void interrupt far(* int1_isr)() );
void int2_init( unsigned char i, void interrupt far(* int2_isr)() );
void int3_init( unsigned char i, void interrupt far(* int3_isr)() );
void int4_init( unsigned char i, void interrupt far(* int4_isr)() );
void int5_init( unsigned char i, void interrupt far(* int5_isr)() );
void int6_init( unsigned char i, void interrupt far(* int6_isr)() );
void int7_init( unsigned char i, void interrupt far(* int7_isr)() );
void int8_init( unsigned char i, void interrupt far(* int8_isr)() );
void int9_init( unsigned char i, void interrupt far(* int9_isr)() );
void nmi_init(void interrupt far (* nmi_isr)());
```

4.2.3 I/O Initialization

There are three ports of 8 I/O pins available on the i386-Engine. Hardware details regarding these PIO lines can be found in the Hardware chapter.

There are several functions provided for access to the PIO lines. At the beginning of any application where you choose to use the PIO pins as input/output, you will probably need to initialize these pins in one of the four available modes. Before selecting pins for this purpose, make sure that the peripheral mode operation of the pin is not needed for a different use within the same application.

You should also confirm the PIO usage that is described above within **ie_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 16 of the Intel386EX Embedded Processor User's Manual.

Please see the sample program **ie_pio.c** in **`tern\386\samples\ie`**. 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 slower when accessing the PIO pins. 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.

void pio_init
Arguments: char port, char bit, char mode
Return value: none

Port and bit refer to the specific PIO line you are dealing with. P10-P17 are in port 1, P20-P27 are in port 2, and P30-P37 are in port 3. Bit 0 refers to Pn0 in each port, while bit 7 is used for Pn7.
Mode refers to one of four modes of operation.

- 0, High-impedance Input operation
- 1, Open-drain output operation
- 2, output
- 3, peripheral mode

unsigned char pio_rd:
Arguments: char port
Return value: byte indicating PIO status

Each bit of the returned byte value indicates the current I/O value for the PIO pins in the selected port.

void pio_wr:
Arguments: char port, char bit, char dat
Return value: none

Writes the passed in dat value (either 1/0) to the selected PIO.

4.2.4 Analog-to-Digital Conversion

The ADC unit provides 11 channels of analog inputs based on the reference voltage supplied to **REF+**. For details regarding the hardware configuration, see the Hardware chapter.

The ADC shares a common data line (**P15**) with the EEPROM. As a result, before using the ADC for this purpose, the EEPROM is placed in stop mode. This is done within the function interface to the ADC. This means that if you are developing an interrupt-driven application, you must be careful of situations where the ADC is in use and the EEPROM is used simultaneously through an interrupt service routine. If this occurs, the calls will block and the application will deadlock. You should also make sure that you do not re-program **P15** for any other use if you are using the ADC.

For a sample file demonstrating the use of the ADC, please see **ie_ad12.c** in **`tern\386\samples\ie`**.

int ie_ad12
Arguments: char c
Return values: int ad_value

The argument **c** selects the channel from which to do the next Analog to Digital conversion. A value of 0 corresponds to channel **AD0**, 1 corresponds to channel **AD1**, and so on.

The return value **ad_value** is the latched-in conversion value from the previous call to this function. This means each call to this function actually returns the value latched-in from the previous analog-to-digital conversion.

For example, this means the first analog-to-digital conversion done in an application will be similar to the following:

```
ie_ad12(0); // Read from channel 0
chn_0_data = ie_ad12(0); // Start the next conversion, retrieve value.
```

4.2.5 Digital-to-Analog Conversion

Up to two LTC 1446 chips are available on the i386-Engine in positions **U11** and **U12**. Each chip offers two channels, A and B, for digital-to-analog conversion. Details regarding hardware, such as pin-outs and performance specifications, can be found in the Hardware chapter.

A sample program demonstrating the DAC can be found in **ie_da12.c** in the directory **`tern\386\samples\ie`**.

void ie_da
void ie_da1
Arguments: int dat1, int dat2
Return value: none

These functions are identical in functionality. **Ie_da()** is used for the DAC chip installed in position **U11** and **ie_da1()** can be used for the one installed in position **U12.**

Argument **dat1** is the current value to drive to channel A of either chip, while argument **dat2** is the value to drive channel B of each chip.

These argument values should range from 0-4095, with units of millivolts. This makes it possible to drive a maximum of 4.906 volts to each channel.

4.2.6 Other library functions

On-board supervisor MAX691 or LTC691

The watchdog timer offered by the MAX691 or LTC691 offers an excellent way to monitor improper program execution. If the watchdog timer (**J9**) is connected, the function **hitwd()** must be called every 1.6 seconds of program execution. If this is not executed because of a run-time error, such as an infinite loop or stalled interrupt service routine, a hardware reset will occur.

void hitwd
Arguments: none
Return value: none

Resets the supervisor timer for another 1.6 seconds.

void led
Arguments: int ledd
Return value: none

Turns the on-board LED on or off according to the value of **ledd**.

Real-Time Clock

The real-time clock can be used to keep track of real time. Backed up by a lithium-coin battery, the real time clock can be accessed and programmed using two interface functions.

The real time clock only allows storage of two digits of the year code, as reflected below. As a result, application developers should be careful to account for a roll-over in digits in the year 2000. One solution might be to store an offset value in non-volatile storage such as the EEPROM.

There is a common data structure used to access and use both interfaces.

```
typedef struct{
  unsigned char sec1; One second digit.
  unsigned char sec10; Ten second digit.
  unsigned char min1; One minute digit.
  unsigned char min10; Ten minute digit.
  unsigned char hour1; One hour digit.
  unsigned char hour10; Ten hour digit.
  unsigned char day1; One day digit.
  unsigned char day10; Ten day digit.
  unsigned char mon1; One month digit.
  unsigned char mon10; Ten month digit.
  unsigned char year1; One year digit.
  unsigned char year10; Ten year digit.
  unsigned char wk; Day of the week.
} TIM;
```

int rtc_rd
Arguments: TIM *r
Return value: int error_code

This function places the current value of the real time clock within the argument **r** structure. The structure should be allocated by the user. This function returns 0 on success and returns 1 in case of error, such as the clock failing to respond.

Void rtc_init
Arguments: char* t
Return value: none

This function is used to initialize and set a value into the real-time clock. The argument **t** should be a null-terminated byte array that contains the new time value to be used.

The byte array should correspond to { ***weekday, year10, year1, month10, month1, day10, day1, hour10, hour1, minute10, minute1, second10, second1,*** 0 }.
If, for example, the time to be initialized into the real time clock is June 5, 1998, Friday, 13:55:30, the byte array would be initialized to:

```
unsigned char t[14] = { 5, 9, 8, 0, 6, 0, 5, 1, 3, 5, 5, 3, 0 };
```

Delay
In many applications it becomes useful to pause before executing any further code. There are functions provided to make this process easy. For applications that require precision timing, you should use hardware timers provided on-board for this purpose.

void delay0
Arguments: unsigned int t
Return value: none

This function is just a simple software loop. The actual time that it waits depends on processor speed as well as interrupt latency. The code is functionally identical to:

```
While(t) { t--; }
```

Passing in a **t** value of 600 causes a delay of approximately 1 ms.

void delay_ms
Arguments: unsigned int
Return value: none

This function is similar to delay0, but the passed in argument is in units of milliseconds instead of loop iterations. Again, this function is highly dependent upon the processor speed.

unsigned int crc16
Arguments: unsigned char *wptr, unsigned int count
Return value: unsigned int value

This function returns a simple 16-bit CRC on a byte-array of **count** size pointed to by **wptr**.

void ie_reset
Arguments: none
Return value: none

This function is similar to a hardware reset, and can be used if your program needs to re-start the board for any reason. Depending on the current hardware configuration, this might either start executing code from the DEBUG ROM or from some other address.

4.3 Functions in SER0.OBJ/SER1.OBJ

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

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

Two asynchronous serial ports are integrated in the i386EX CPU: SER0 and SER1. Both ports by default use the signal **SERCLK** to drive communication, which is based on the 66 MHz system clock signal **CLK2**. By default, SER0 is used by the DEBUG ROM for application download/debugging in STEP 1 and

STEP 2. We will use SER1 as the example in the following discussion; any of the interface functions which are specific to SER1 can be easily changed into function calls for SER0. While selecting a serial port for use, please realize that some pins might be shared with other peripheral functions. This means that in certain limited cases, it might not be possible to use a certain serial port with other on-board controller functions. For details, you should see both chapter 11 of the Intel 386EX Embedded Microprocessor User's Manual and the schematic of the i386-Engine provided at the end of this manual.

TERN interface functions make it possible to use one of a number of predetermined baud rates. These baud rates are achieved by specifying a divisor for **SERCLK (***1,031,250 hz***)**.

The following table shows the function arguments that express each baud rate, to be used in TERN functions. These are based on a 33 MHz system clock.

Function Argument	Divisor Value	Baud Rate
1	6875	150
2	3438	300
3	1719	600
4	859	1200
5	430	2400
6	215	4800
7	107	9600
8	72	14,400
9	54	19,200 (default)
10	27	38,400
11	18	57,600
12	9	115,200
13	4	275,812
14	2	515,625
15	1	1,031,250

Table 4.1 Baud rate values

After initialization by calling **`s1_init()`**, SER1 is configured as a full-duplex serial port and is ready to transmit/receive serial data at one of the specified 15 baud rates.

An input buffer, **`ser1_in_buf`** (whose size is specified by the user), will automatically store the receiving serial data stream into the memory by DMA1 operation. In terms of receiving, there is no software overhead or interrupt latency for user application programs even at the highest baud rate. DMA transfer allows efficient handling of incoming data. The user only has to check the buffer status with **`serhit1()`** and take out the data from the buffer with **`getser1()`**, if any. The input buffer is used as a circular ring buffer, as shown in Figure 4.1. However, the transmit operation is interrupt-driven.

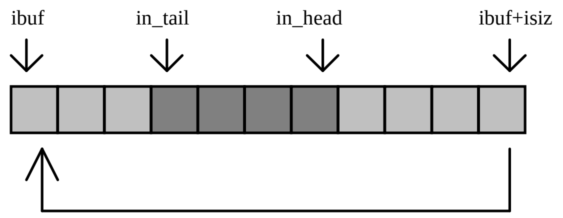


Figure 4.1 Circular ring input buffer

The input buffer (**ibuf**), buffer size (**isiz**), and baud rate (**baud**) are specified by the user with **`s1_init()`** with a default mode of 8-bit, 1 stop bit, no parity. After **`s1_init()`** you can set up a new mode with different numbers for data-bit, stop bit, or parity by directly accessing the Serial Line Control Register (LCR1) if necessary, as described in the Intel386EX manual for asynchronous serial ports.

Due to the nature of high-speed baud rates and possible effects from the external environment, serial input data will automatically fill in the buffer circularly without stopping, regardless of overwrite. If the user does not take out the data from the ring buffer with **`getser1()`** before the ring buffer is full, new data will overwrite the old data without warning or control. Thus it is important to provide a sufficiently large buffer if large amounts of data are transferred. For example, if you are receiving data at 9600 baud, a 4 KB buffer will be able to store data for approximately four seconds.

However, it is always important to take out data early from the input buffer, before the ring buffer rolls over. You may designate a higher baud rate for transmitting data out and a slower baud rate for receiving data. This will give you more time to do other things, without overrunning the input buffer. You can use **`serhit1()`** to check the status of the input buffer and return the offset of the in_head pointer from the in_tail pointer. A return value of 0 indicates no data is available in the buffer.

You can use **`getser1()`** to get the serial input data byte by byte using FIFO from the buffer. The in_tail pointer will automatically increment after every **`getser1()`** call. It is not necessary to suspend external devices from sending in serial data with /RTS. Only a hardware reset or **`s1_close()`** can stop this receiving operation.

For transmission, you can use **`putser1()`** to send out a byte, or use **`putsers1()`** to transmit a character string. You can put data into the transmit ring buffer, **`s1_out_buf`**, at any time using this method. The transmit ring buffer address (**obuf**) and buffer length (**osiz)** are also specified at the time of initialization. The transmit interrupt service will check the availability of data in the transmit buffer. If there is no more data (the head and tail pointers are equal), it will disable the transmit interrupt. Otherwise, it will continue to take out the data from the out buffer, and transmit. After you call **`putser1()`** and transmit functions, you are free to do other tasks with no additional software overhead on the transmitting operation. It will automatically send out all the data you specify. After all data has been sent, it will clear the busy flag and be ready for the next transmission.

The sample program **`ser1_0.c`** demonstrates how a protocol translator works. It would receive an input HEX file from SER1 and translate every ':' character to '?.' The translated HEX file is then transmitted out of SER0. This sample program can be found in **`tern\386\samples\ie`**.

Software Interface

Before using the serial ports, they must be initialized.

There is a data structure containing important serial port state information that is passed as argument to the TERN library interface functions. The **COM** structure should normally be manipulated only by TERN libraries. It is provided to make debugging of the serial communication ports more practical. Since it allows you to monitor the current value of the buffer and associated pointer values, you can watch the transmission process.

The two serial ports have similar software interfaces. Any interface that makes reference to either **s0** or **ser0** can be replaced with **s1** or **ser1**, for example. Each serial port should use its own **COM** structure, as defined in **ie.h**.

```
typedef struct  {
  unsigned char ready;          /* TRUE when ready */
  unsigned char baud;
  unsigned char mode;
  unsigned char iflag;     /* interrupt status     */
  unsigned char            *in_buf;        /* Input buffer */
  int  in_tail;        /* Input buffer TAIL ptr */
  int  in_head;        /* Input buffer HEAD ptr */
  int  in_size;        /* Input buffer size */
  int  in_crcnt;       /* Input <CR> count */
  unsigned char   in_mt;          /* Input buffer FLAG */
  unsigned char   in_full;        /* input buffer full */
  unsigned char   *out_buf;       /* Output buffer */
  int  out_tail;       /* Output buffer TAIL ptr */
  int  out_head;       /* Output buffer HEAD ptr */
  int  out_size;       /* Output buffer size */
  unsigned char  out_full;       /* Output buffer FLAG */
  unsigned char  out_mt;         /* Output buffer MT */
  unsigned char tmso;   // transmit macro service operation
  unsigned char rts;
  unsigned char dtr;
  unsigned char en485;
  unsigned char err;
  unsigned char node;
  unsigned char cr; /* scc CR register    */
  unsigned char slave;
  unsigned int in_segm;       /* input buffer segment */
  unsigned int in_offs;       /* input buffer offset */
  unsigned int out_segm;       /* output buffer segment */
  unsigned int out_offs;       /* output buffer offset */
  unsigned char byte_delay;   /* V25 macro service byte delay */
} COM;
```

s*n*_init
Arguments: unsigned char b, unsigned char* ibuf, int isiz, unsigned char* obuf, int osiz, COM* c
Return value: none

This function initializes either SER0 or SER1 with the specified parameters. **b** is the baud rate value shown in Table 4.1. Arguments **ibuf** and **isiz** specify the input-data buffer, and **obuf** and **osiz** specify the location and size of the transmit ring buffer.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Miscellaneous Serial Communication Functions

One thing to be aware of in both transmission and receiving of data through the serial port is that TERN drivers only use the basic serial-port communication lines for transmitting and receiving data. Hardware flow control in the form of **CTS** (Clear-To-Send) and **RTS** (Ready-To-Send) is not implemented. There are, however, functions available that allow you to check and set the value of these I/O pins appropriate for whatever form of flow control you wish to implement. Before using these functions, you should once again be aware that the peripheral pin function you are using might not be selected as needed. For details, please refer to chapter 11 of the Intel386EX Embedded Microprocessor User's Manual.

For an example on implementing your own flow control, please see **s0_rts.c** in **`tern\samples\ie`**.

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

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

void s*n*_dtr(char b)
Sets the value of **DTR** to **b**.

Completing Serial Communications

After completing your serial communications, there are a few functions that can be used 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.

clean_ser*n*
Arguments: COM *c
Return value: none

This flushes the input buffer by resetting the tail and header buffer pointers.

The asynchronous serial I/O ports available on the Intel386EX Embedded Processor have many other features that might be useful for your application. If you are truly interested in having more control, please read Chapter 11 of the manual for a detailed discussion of other features available to you.

4.4 Functions in SCC.OBJ

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

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

Table 4.2 Function Arguments for Baud Rate

Function Argument	Baud Rate
1	110
2	150
3	300
4	600
5	1200
6	2400
7	4800
8	9600 (default)
9	19,200
10	31,250

11	62,500
12	125,000
13	250,000

Unlike the other serial ports, DMA transfer is not used to fill the input buffer for SCC. Instead, an interrupt-service-routine is used to place characters into the input buffer. If the processor does not respond to the interrupt—because it is masked, for example—the interrupt service routine might never be able to complete this process. Over time, this means data might be lost in the SCC as bytes overflow.

Special control registers are used to define how the SCC operates. For a detailed description of registers **MR1** and **MR2**, please see Appendix C of this manual. In most TERN applications, MR1 is set to ***0x57***, and MR2 is set to ***0x07***. This configures the SCC for no flow control (RTS, CTS not used/checked), no parity, 8-bit, normal operation. Other configurations are also possible, providing self-echo, even-odd parity, up to 2 stop bits, 5 bit operation, as well as automatic hardware flow control.

Initialization occurs in a manner otherwise similar to SER0 and SER1. A **COM** structure is once again used to hold state information for the serial port. The in-bound and out-bound buffers operate as before, and must be provided upon initialization.

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

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

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

After initializing the serial port, you must also set up the interrupt service routine. The SCC2691 UART takes up external interrupt **/INT5** 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:

```
int5_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 **ie_scc.c** in the directory `tern\samples\ie`.

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 **en485(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 **en485(0)**. For a sample file showing RS485 communication, please see **ie_rs485.c** in the directory `tern\samples\ie`.

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_recv_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_recv_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 send and receive. If you are using RS485, you will need to use this feature when transitioning from transmission to reception, or from reception to transmission.

Transmission and reception of data using the SCC is in most ways identical to SER0 and SER1. The functions used to transmit and receive data are similar. For details regarding these functions, please refer to the previous section.

putser_scc
`See:` **`putser`*`n`***

putsers_scc
`See:` **`putsers`*`n`***

getser_scc
`See:` **`getser`*`n`***

getsers_scc
`See:` **`getsers`*`n`***

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

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

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

Other SCC functions are similar to those for SER0 and SER1.
ser_close
`See:` **`s`*`n`*`_close`**

ser_hit
`See:` **`s`*`n`*`_hit`**

clean_ser_scc
`See: clean_sn`

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

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

4.5 Functions in IEEE.OBJ

The 512-byte serial EEPROM (***24C04***) provided on-board provides 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 2, and other data that is of a more permanent nature.

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

The EEPROM also shares line **P15** with the ADC on the i386-Engine, if installed. As described above, when the ADC is in use, the EEPROM is placed in stop mode. When using the EEPROM, be careful when trying to use the ADC concurrently.

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

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

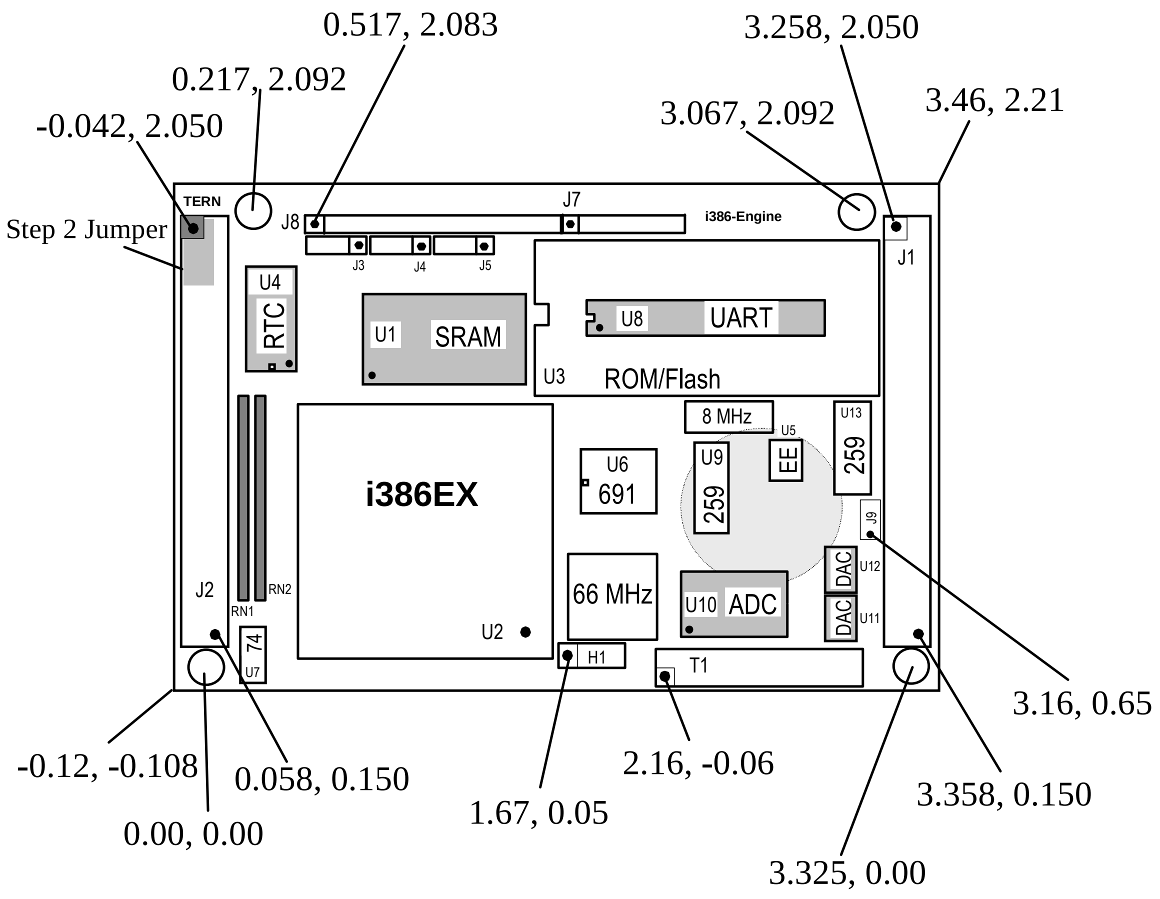
ee_rd
Arguments: int addr
Return value: int data

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

Appendix A: i386-Engine Layout

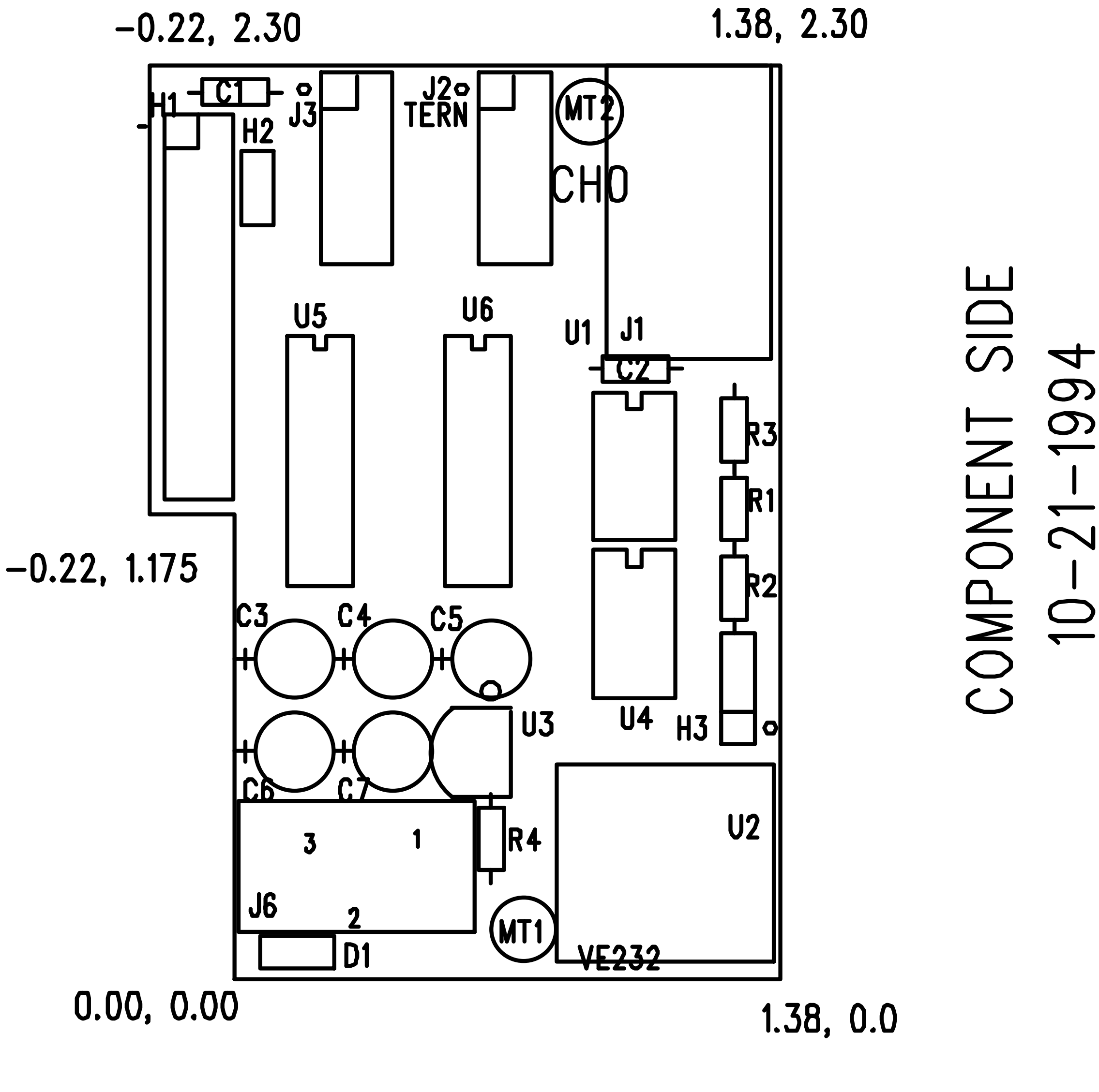
The i386-Engine measures 3.6 by 2.3 inches. Its layout is show below.

All dimensions are in inches.



Appendix B: VE232 Pin Layout

All dimensions are in inches.



Appendix C: UART SCC2691

1. Pin Description

D0-D7	Data bus, active high, bi-directional, and having 3-State
/CEN	Chip enable, active-low input
/WRN	Write strobe, active-low input
/RDN	Read strobe, active-low input
A0-A2	Address input, active-high address input to select the UART registers
RESET	Reset, active-high input
INTRN	Interrupt request, active-low output
X1/CLK	Crystal 1, crystal or external clock input
X2	Crystal 2, the other side of crystal
RxD	Receive serial data input
TxD	Transmit serial data output
MPO	Multi-purpose output
MPI	Multi-purpose input
Vcc	Power supply, +5 V input
GND	Ground

2. Register Addressing

A2	A1	A0	READ (RDN=0)	WRITE (WRN=0)
0	0	0	MR1,MR2	MR1, MR2
0	0	1	SR	CSR
0	1	0	BRG Test	CR
0	1	1	RHR	THR
1	0	0	1x/16x Test	ACR
1	0	1	ISR	IMR
1	1	0	CTU	CTUR
1	1	1	CTL	CTLR

Note:

ACR = Auxiliary control register
BRG = Baud rate generator
CR = Command register
CSR = Clock select register
CTL = Counter/timer lower
CTLR = Counter/timer lower register
CTU = Counter/timer upper
CTUR = Counter/timer upper register
MR = Mode register
SR = Status register
RHR = Rx holding register
THR = Tx holding register

3. Register Bit Formats

MR1 (Mode Register 1):

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RxRTS 0 = no 1 = yes	RxINT 0=RxRDY 1=FFULL	Error 0 = char 1 = block	___Parity Mode___ 00 = with parity 01 = Force parity 10 = No parity 11 = Special mode		Parity Type 0 = Even 1 = Odd In Special mode: 0 = Data 1 = Addr	Bits per Character 00 = 5 01 = 6 10 = 7 11 = 8	

MR2 (Mode Register 2):

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

Channel Mode	TxRTS	CTS Enable Tx	Stop Bit Length (add 0.5 to cases 0-7 if channel is 5 bits/character)
00 = Normal 01 = Auto echo 10 = Local loop 11 = Remote loop	0 = no 1 = yes	0 = no 1 = yes	0 = 0.563 4 = 0.813 8 = 1.563 C = 1.813 1 = 0.625 5 = 0.875 9 = 1.625 D = 1.875 2 = 0.688 6 = 0.938 A = 1.688 E = 1.938 3 = 0.750 7 = 1.000 B = 1.750 F = 2.000

CSR (Clock Select Register):

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

Receiver Clock Select	Transmitter Clock Select
when ACR[7] = 0: 0 = 50 1 = 110 2 = 134.5 3 = 200 4 = 300 5 = 600 6 = 1200 7 = 1050 8 = 2400 9 = 4800 A = 7200 B = 9600 C = 38.4k D = Timer E = MPI-16x F = MPI-1x	when ACR[7] = 0: 0 = 50 1 = 110 2 = 134.5 3 = 200 4 = 300 5 = 600 6 = 1200 7 = 1050 8 = 2400 9 = 4800 A = 7200 B = 9600 C = 38.4k D = Timer E = MPI-16x F = MPI-1x
when ACR[7] = 1: 0 = 75 1 = 110 2 = 134.5 3 = 150 4 = 300 5 = 600 6 = 1200 7 = 2000 8 = 2400 9 = 4800 A = 7200 B = 1800 C = 19.2k D = Timer E = MPI-16x F = MPI-1x	when ACR[7] = 1: 0 = 75 1 = 110 2 = 134.5 3 = 150 4 = 300 5 = 600 6 = 1200 7 = 2000 8 = 2400 9 = 4800 A = 7200 B = 1800 C = 19.2k D = Timer E = MPI-16x F = MPI-1x

CR (Command Register):

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

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

SR (Channel Status Register):

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

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

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

ACR (Auxiliary Control Register):

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

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

ISR (Interrupt Status Register):

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

MPI Pin Change	MPI Pin Current State	Not Used	Counter Ready	Delta Break	RxRDY/ FFULL	TxEMT	TxRDY
0 = no 1 = yes	0 = low 1 = high		0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes

IMR (Interrupt Mask Register):

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

MPI Change Interrupt	MPI Level Interrupt	Not Used	Counter Ready Interrupt	Delta Break Interrupt	RxRDY/ FFULL Interrupt	TxEMT Interrupt	TxRDY Interrupt
0 = off 1 = 0n	0 = off 1 = 0n		0 = off 1 = 0n	0 = off 1 = 0n	0 = off 1 = 0n	0 = off 1 = 0n	0 = off 1 = 0n

CTUR (Counter/Timer Upper Register):

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

CTLR (Counter/Timer Lower Register):

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

Appendix D: RTC72421/72423

Function Table

Address					Data					
A_3	A_2	A_1	A_0	Register	D_3	D_2	D_1	D_0	Count Value	Remarks
0	0	0	0	S_1	s_8	s_4	s_2	s_1	0~9	1-second digit register
0	0	0	1	S_{10}		s_{40}	s_{20}	s_{10}	0~5	10-second digit register
0	0	1	0	MI_1	mi_8	mi_4	mi_2	mi_1	0~9	1-minite digit register
0	0	1	1	MI_{10}		mi_{40}	mi_{20}	mi_{10}	0~5	10-minute digit register
0	1	0	0	H_1	h_8	h_4	h_2	h_1	0~9	1-hour digit register
0	1	0	1	H_{10}		PM/AM	h_{20}	h_{10}	0~2 or 0~1	PM/AM, 10-hour digit register
0	1	1	0	D_1	d_8	d_4	d_2	d_1	0~9	1-day digit register
0	1	1	1	D_{10}			d_{20}	d_{10}	0~3	10-day digit register
1	0	0	0	MO_1	mo_8	mo_4	mo_2	mo_1	0~9	1-month digit register
1	0	0	1	MO_{10}				mo_{10}	0~1	10-month digit register
1	0	1	0	Y_1	y_8	y_4	y_2	y_1	0~9	1-year digit register
1	0	1	1	Y_{10}	y_{80}	y_{40}	y_{20}	y_{10}	0~9	10-year digit register
1	1	0	0	W		w_4	w_2	w_1	0~6	Week register
1	1	0	1	Reg D	30s Adj	IRQ Flag	Busy	Hold		Control register D
1	1	1	0	Reg E	t_1	t_0	INT/ STD	Mask		Control register E
1	1	1	1	Reg F	Test	24/ 12	Stop	Rest		Control register F

Note: 1) INT/STD = Interrupt/Standard, Rest = Reset;

2) Mask AM/PM bit with 10's of hours operations;

3) Busy is read only, IRQ can only be set low ("0");

4)

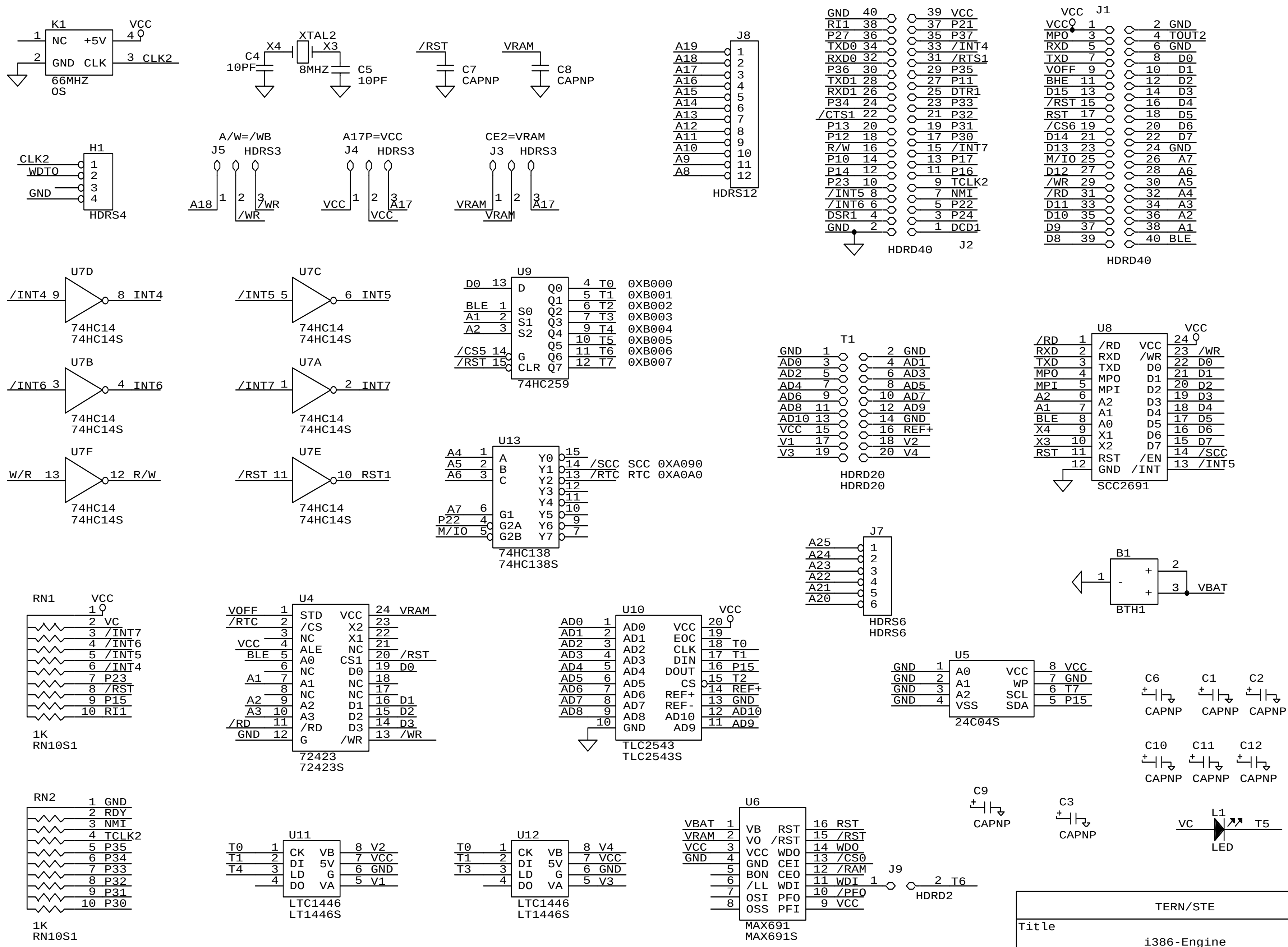
Data bit	PM/AM	INT/STD	24/12
1	PM	INT	24
0	AM	STD	12

5) Test bit should be "0".

Appendix E: Serial EEPROM Map

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

0x00	Node Address, for networking		
0x01	Board Type	00	VE
		10	CE
		01	BB
		02	PD
		03	SW
		04	TD
		05	MC
0x02			
0x03			
0x04	SER0_receive, used by ser0.c		
0x05	SER0_transmit, used by ser0.c		
0x06	SER1_receive, used by ser1.c		
0x07	SER1_transmit, used by ser1.c		
0x10	CS high byte, used by STEP 2		
0x11	CS low byte, used by STEP 2		
0x12	IP high byte, used by STEP 2		
0x13	IP low byte, used by STEP 2		
0x18	MM page register 0		
0x19	MM page register 1		
0x1a	MM page register 2		
0x1b	MM page register 3		



K1 66MHZ OS NC +5V GND CLK VCC CLK2
XTAL2 8MHZ X4 X3 C4 10PF C5 10PF
/RST C7 CAPNP
VRAM C8 CAPNP
J8 HDRS12 A19 A18 A17 A16 A15 A14 A13 A12 A11 A10 A9 A8
J2 HDRD40 GND 40 RI1 38 P27 36 TXD0 34 RXD0 32 P36 30 TXD1 28 RXD1 26 P34 24 /CTS1 22 P13 20 P12 18 R/W 16 P10 14 P14 12 P23 10 /INT5 8 /INT6 6 DSR1 4 GND 2
39 VCC 37 P21 35 P37 33 /INT4 31 /RTS1 29 P35 27 P11 25 DTR1 23 P33 21 P32 19 P31 17 P30 15 /INT7 13 P17 11 P16 9 TCLK2 7 NMI 5 P22 3 P24 1 DCD1
J1 HDRD40 VCC 1 MPO 3 RXD 5 TXD 7 VOFF 9 BHE 11 D15 13 /RST 15 RST 17 /CS6 19 D14 21 D13 23 M/IO 25 D12 27 /WR 29 /RD 31 D11 33 D10 35 D9 37 D8 39
2 GND 4 TOUT2 6 GND 8 D0 10 D1 12 D2 14 D3 16 D4 18 D5 20 D6 22 D7 24 GND 26 A7 28 A6 30 A5 32 A4 34 A3 36 A2 38 A1 40 BLE
H1 HDRS4 CLK2 WDTO GND
J5 HDRS3 A/W=/WB A18 /WR /WR
J4 HDRS3 A17P=VCC VCC VCC A17
J3 HDRS3 CE2=VRAM VRAM VRAM A17
U7D 74HC14 74HC14S /INT4 INT4
U7C 74HC14 74HC14S /INT5 INT5
U7B 74HC14 74HC14S /INT6 INT6
U7A 74HC14 74HC14S /INT7 INT7
U7F 74HC14 74HC14S W/R R/W
U7E 74HC14 74HC14S /RST RST1
U9 74HC259 D0 BLE A1 A2 /CS5 /RST D S0 S1 S2 G CLR Q0–Q7 T0 0XB000 T1 0XB001 T2 0XB002 T3 0XB003 T4 0XB004 T5 0XB005 T6 0XB006 T7 0XB007
U13 74HC138 74HC138S A4 A5 A6 A7 P22 M/IO A B C G1 G2A G2B Y0–Y7 /SCC SCC 0XA090 /RTC RTC 0XA0A0
T1 HDRD20 HDRD20 GND 1 AD0 3 AD2 5 AD4 7 AD6 9 AD8 11 AD10 13 VCC 15 V1 17 V3 19 2 GND 4 AD1 6 AD3 8 AD5 10 AD7 12 AD9 14 GND 16 REF+ 18 V2 20 V4
U8 SCC2691 /RD RXD TXD MPO MPI A2 A1 BLE X4 X3 RST VCC /WR D0–D7 /SCC /INT5 /EN /INT GND
J7 HDRS6 HDRS6 A25 A24 A23 A22 A21 A20
B1 BTH1 VBAT
RN1 1K RN10S1 VCC VC /INT7 /INT6 /INT5 /INT4 P23 /RST P15 RI1
RN2 1K RN10S1 GND RDY NMI TCLK2 P35 P34 P33 P32 P31 P30
U4 72423 72423S VOFF /RTC VCC BLE A1 A2 A3 /RD GND STD /CS NC ALE A0 A1 A2 A3 /RD G VCC X2 X1 NC CS1 D0 NC NC D1 D2 D3 /WR VRAM /RST D0 D1 D2 D3 /WR
U10 TLC2543 TLC2543S AD0–AD8 GND VCC EOC CLK DIN DOUT CS REF+ REF- AD10 AD9 T0 T1 P15 T2 REF+ GND AD10 AD9
U5 24C04S GND A0 A1 A2 VSS VCC WP SCL SDA VCC GND T7 P15
C6 C1 C2 C10 C11 C12 C9 C3 CAPNP
U11 LTC1446 LT1446S T0 T1 T4 CK DI LD DO VB 5V G VA V2 VCC GND V1
U12 LTC1446 LT1446S T0 T1 T3 CK DI LD DO VB 5V G VA V4 VCC GND V3
U6 MAX691 MAX691S VBAT VRAM VCC GND VB VO VCC GND BON /LL OSI OSS RST /RST WDO CEI CEO WDI PFO PFI RST /RST WDO /CS0 /RAM WDI /PFO VCC
J9 HDRD2 T6
L1 LED VC T5

TERN/STE		
Title i386-Engine		
Size B	Document Number IE-MANU.SCH	REV
Date: June 18, 1998	Sheet 1 of 1	