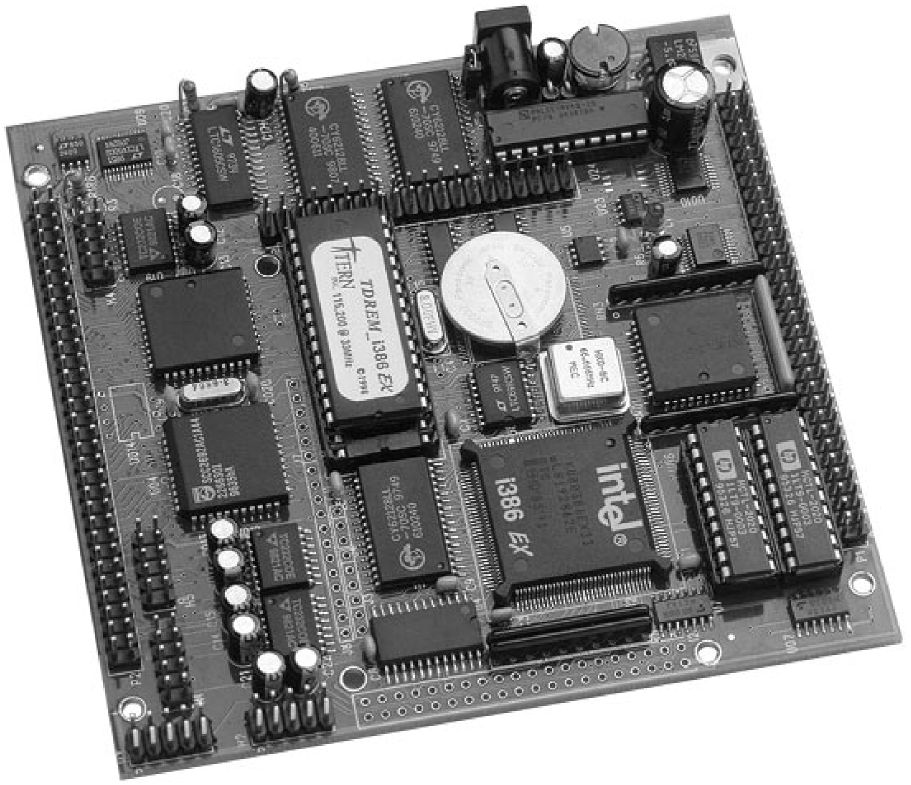
i386-Drive™

C/C++ programmable, 32-bit microprocessor module with
70+ I/Os, UARTs, ADC, DAC, and quadrature decoders
based on the Intel386EX



Technical Manual

TERN INC.

1724 Picasso Avenue, Suite A, Davis, CA 95616-0547, USA
Tel: 530-758-0180 Fax: 530-758-0181
Internet Email: tern@netcom.com *http://www.tern.com*

Version 2.02

August 13, 1999

1724 Picasso Avenue, Suite A, Davis, CA 95616-0547, USA
Tel: 530-758-0180   Fax: 530-758-0181
Internet Email: tern@netcom.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.

Table of Contents

Chapter 1: Introduction

1.1 Functional Description

The ***i386-Drive*** (***ID***) is a compact, low-cost, high performance controller based on the 33 MHz, 32-bit Intel386EX. It combines the powerful i386EX CPU and numerous peripherals on a single PCB measuring 4.7 by 4.5 inches.

The ***ID*** supports up to 512 KB 8-bit SRAM, 512 KB 8-bit Flash, 1 MB 16-bit SRAM, and 1 MB 16-bit Flash. A 512-byte serial EEPROM, which does not require a battery backup, can be used as an additional memory device to store important data. An optional real-time clock (RTC) provides information on the year, month, date, hour, minute, second, 1/64 second. A lithium coin battery can be installed to back up both the SRAM and RTC.

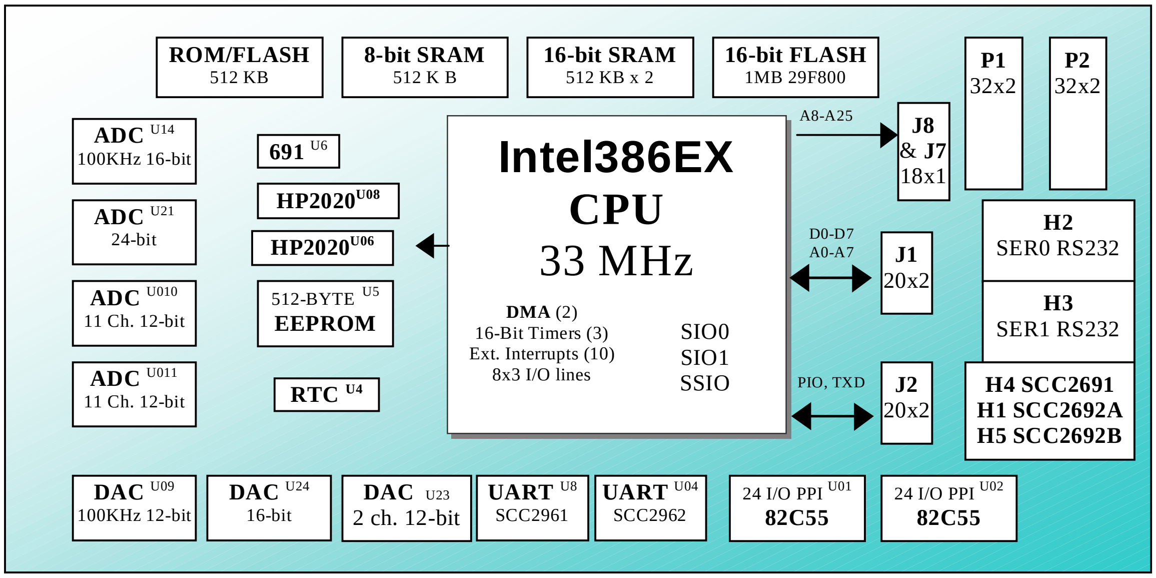


Figure 1.1 Functional block diagram of the i386-Drive

Two asynchronous serial ports from the i386EX support reliable DMA-driven serial communication at up to 115,200 baud with RS-232 drivers. The i386EX also offers one synchronous serial port. An optional UART SCC2691 and a dual UART SCC2692 can be added for an additional three asynchronous serial ports with RS-232 or RS-485 drivers.

Three PC-compatible 16-bit programmable timers/counters can generate interrupts or count external events, at a rate of up to 8 MHz, or can generate pulse outputs. Three 8-bit, multifunctional, user-programmable I/O ports are included in the i386EX. Four external interrupts are buffered by Schmitt-trigger inverters and provide active low inputs. A supervisor chip (LTC691) with a watchdog timer is on-board.

Two PPI chips (82C55) provide 48 user-programmable I/O lines totally free for application use. The optional SCC2692 UART provides 15 additional I/O lines.

The ***ID*** supports many optional ADC and DACs. Up to 22 channels of 12-bit ADC (LTC2543, 0-5V, 10 KHz), one 16-bit ADC (LTC1605, ±10V, 100 KHz), and one 24-bit ADC (LTC2400, 0-5V, 5 Hz) can be installed. Two 12-bit DACs (LTC1446, 0-4.095V, 10 KHz), one 100 KHz 12-bit DAC (LTC1450, 0-4.095V), and one 16-bit DAC (LTC1655, 0-4.095V, 10 KHz) are available.

Two quadrature decoders (HP2020) can interface to optical encoders for motion control. Schmitt-trigger inverters are provided.

On-board expansion headers provide data lines, address lines, control signals, and pre-decoded chip select lines.

By default, a 5V switching regulator (up to 35V DC input) is installed to reduce power consumption and heat. The switching regulator introduces more noise than a linear regulator: a linear regulator can be installed upon request.

In “power-off” mode, the ***ID*** consumes very low (µA) power. Users can turn off the switching regulator via software, and use the RTC or an external signal to turn it on.

A MemCard-A can be installed on the ***ID*** to provide an additional 33 12-bit ADC, 6 24-bit ADC, 420 MB PCMCIA memory, and an Ethernet interface.

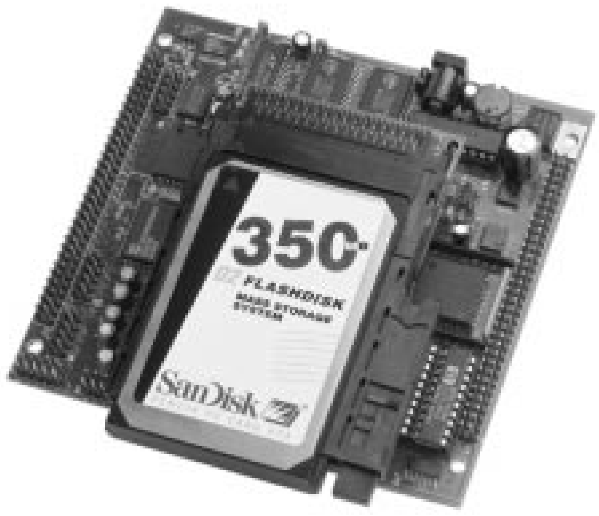


Figure 1.2 An i386-Drive with a MemCard-A installed

1.2 Features

Standard Features:

- Dimensions: 4.7 x 4.5 x 0.6 inches
- Easy to program in C/C++
- Power consumption: 300/160/80/30 mA at 8.5/12/24/35V
- Power input: +8.5 to +35 V
- Temperature range: -40°C to +80°C
- 32-bit CPU (Intel i386EX, 33 MHz), C/C++ programmable
- 24 multiplexed I/Os
- interrupts, DMA
- 512-byte serial EEPROM
- 48 bi-directional I/O lines from 2 PPIs
- Up to 3 MB SRAM/Flash supported

- 2 asynchronous serial ports with RS-232 drivers, 1 synchronous serial port
- Supervisor chip (691) for power failure, reset and watchdog

Optional Features (* surface-mounted components)**:**

- 32KB, 128KB, or 512KB 8-bit SRAM*
- 256KB or 1 MB 16-bit SRAM*
- 512KB or 1MB 16-bit Flash*
- up to 22 channels of 12-bit ADC, sample rate up to 10 KHz*
- 16-bit ADC (LTC1605, ±10V, 100 KHz)*
- 24-bit ADC (LTC2400, 0-5V, 5 Hz)*
- 2 channels of 12-bit DAC, 0-4.095V output*
- 100 KHz 12-bit DAC (LTC1450, 0-4.095V)*
- 16-bit DAC (LTC1655, 0-4.095V, 10 KHz)*
- SCC2691 UART (on-board) supports 8-bit or 9-bit networking, with RS-232* or RS-485 drivers
- SCC2692 dual UART, with RS-232* or RS-485 drivers
- up to 2 quadrature decoders (HP2020)
- Real-time clock RTC72423*, lithium coin battery*

1.3 Physical Description

The physical layout of the i386-Drive is shown in Figure 1.3.

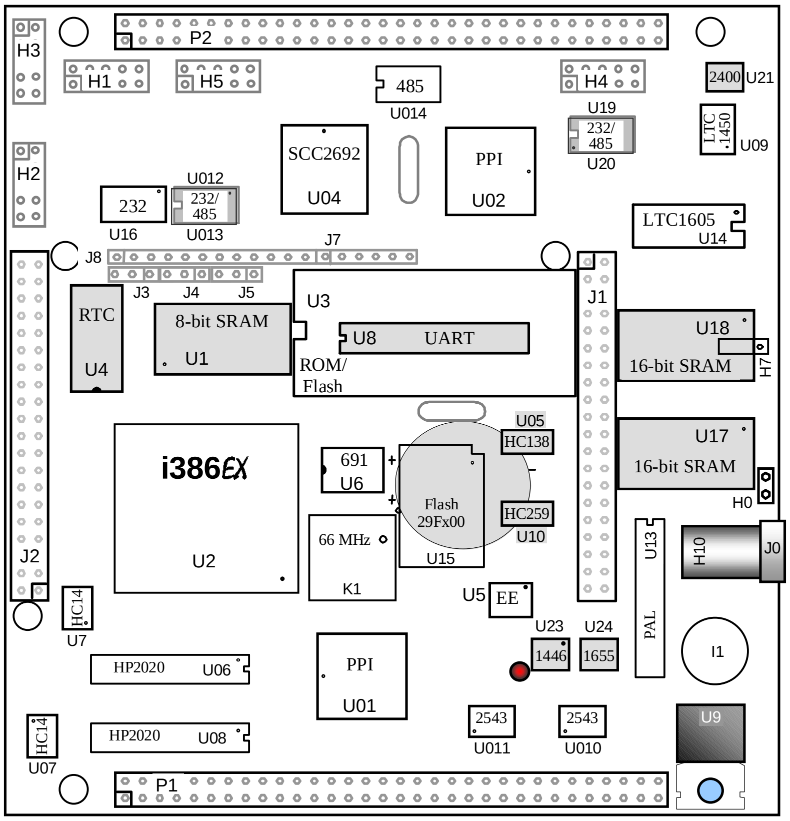
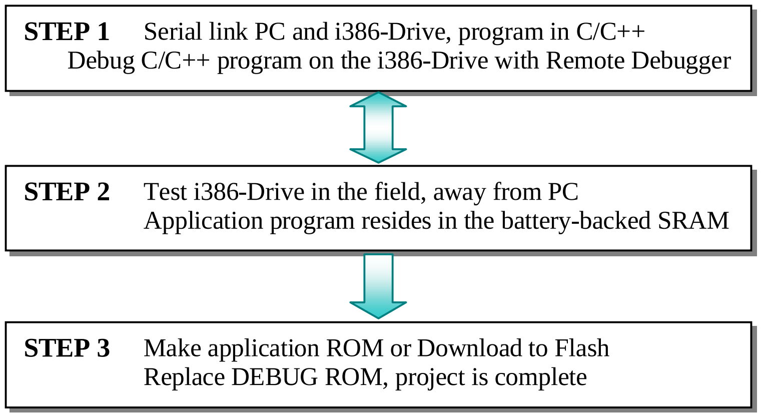


Figure 1.3 Physical layout of the i386-Drive

1.4 i386-Drive Programming Overview

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


STEP 1 Serial link PC and i386-Drive, program in C/C++
Debug C/C++ program on the i386-Drive with Remote Debugger
STEP 2 Test i386-Drive in the field, away from PC
Application program resides in the battery-backed SRAM
STEP 3 Make application ROM or Download to Flash
Replace DEBUG ROM, project is complete

You can program the i386-Drive 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-Drive 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 on-board Flash.

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

1.4.1 Step 1

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.

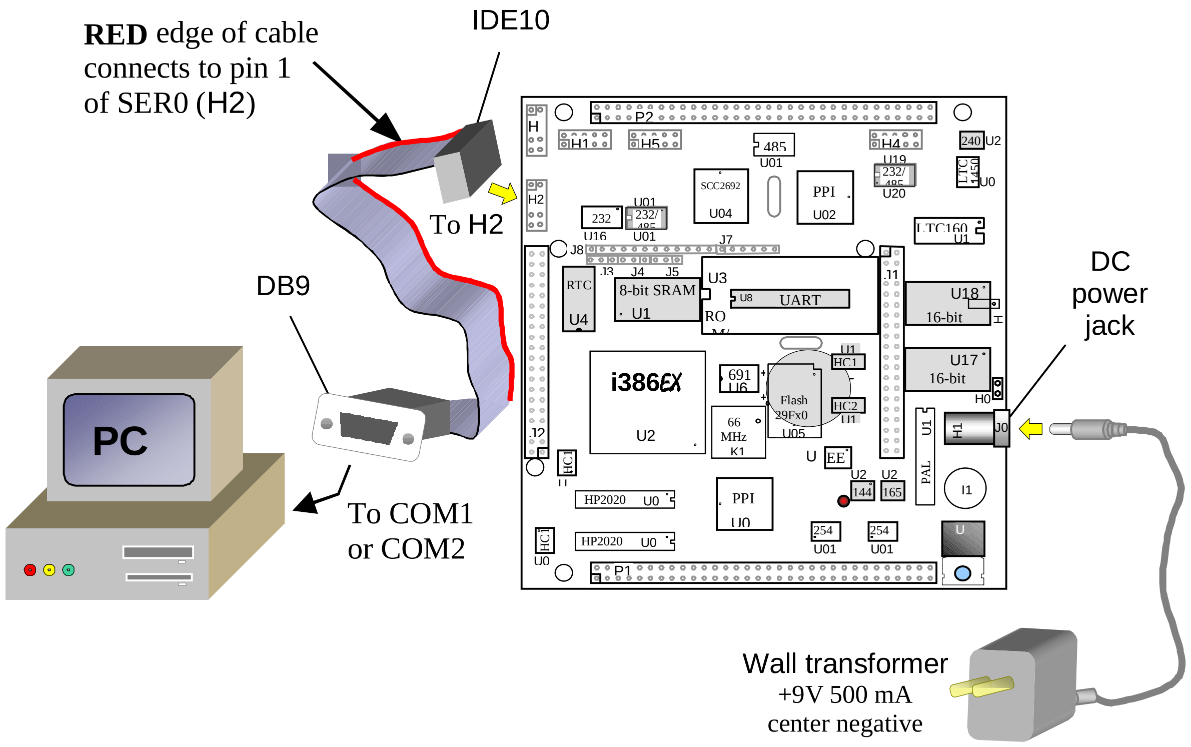

RED edge of cable connects to pin 1 of SER0 (H2)
IDE10
To H2
DB9
PC
To COM1 or COM2
DC power jack
Wall transformer +9V 500 mA center negative
i386EX
U2
8-bit SRAM
UART
Flash 29Fx0
PPI
RTC
P1
P2
H1
H2
H4
H5

Figure 1.4 Step 1 connections for the i386-Drive

1.4.2 Step 2

STEP 2: Standalone Field Test.

- Set the jumper on J2 pins 38-40 on the i386-Drive (Figure 1.5).
- At power-on or reset, if J2 pin 38 (RI1) 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-Drive will operate in Step Two mode. If the jumper is off J2 pins 38-40 at power-on or reset, the i386-Drive will operate in Step One mode. The status of J2 pin 38 (signal RI1) of the Intel386EX is only checked at power-on or at reset.

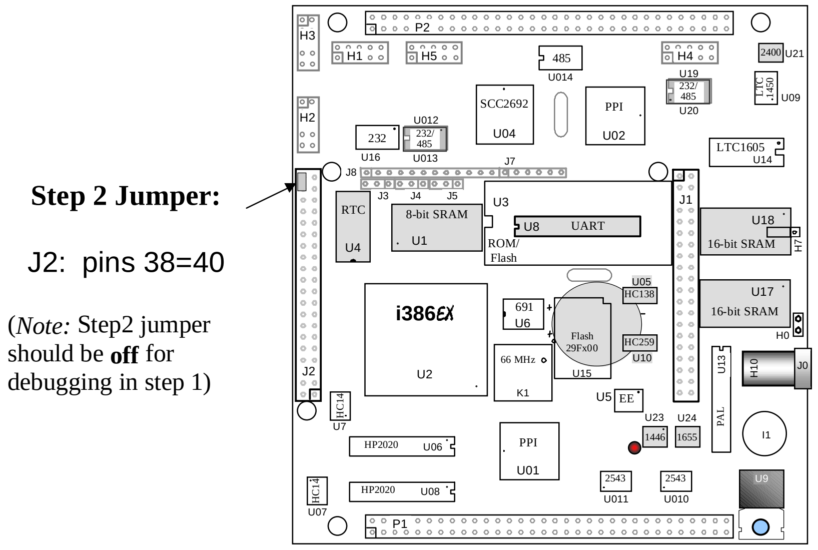


Figure 1.5 Location of Step 2 jumper on the i386-Drive

1.4.3 Step 3

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 2 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-Drive.

1.5 Minimum Requirements for i386-Drive System Development

1.5.1 Minimum Hardware Requirements

- PC or PC-compatible computer with serial COMx port that supports 115,200 baud
- i386-Drive controller with DEBUG ROM *TDREM_i386EX*
- PC-V25 serial cable (RS-232; DB9 connector for PC COM port and IDE 2x5 connector for controller)
- center negative wall transformer (+9V 500 mA)

1.5.2 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-Drive in Step Three and Step Two, but you cannot run Step Three. In order to generate an application ROM/Flash file, make production version ROMs, and complete a 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 and Evaluation Kit for TERN 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

Hardware installation for the i386-Drive consists primarily of connecting the microcontroller to your PC.

Overview

- Connect PC-V25 cable:
  For debugging (STEP 1), place the 5x2 pin header on SER0 (H2) with red edge of cable at pin 1 of H2
- Connect wall transformer:
  Connect 9V wall transformer to power and plug into power jack

2.2.1 Connecting the i386-Drive to the PC

The following diagram (Figure 2.1) illustrates the connection between the i386-Drive and the PC. The i386-Drive is linked to the PC via a serial cable (PC-V25).

The *TDREM_i386EX* DEBUG ROM communicates through SER0 by default. Install the 5x2 IDE connector on the SER0 header (H2). ***IMPORTANT:*** *Note that the **red** side of the cable must point to pin 1 of the H2 header.* The DB9 connector should be connected to one of your PC's COM Ports (COM1 or COM2).

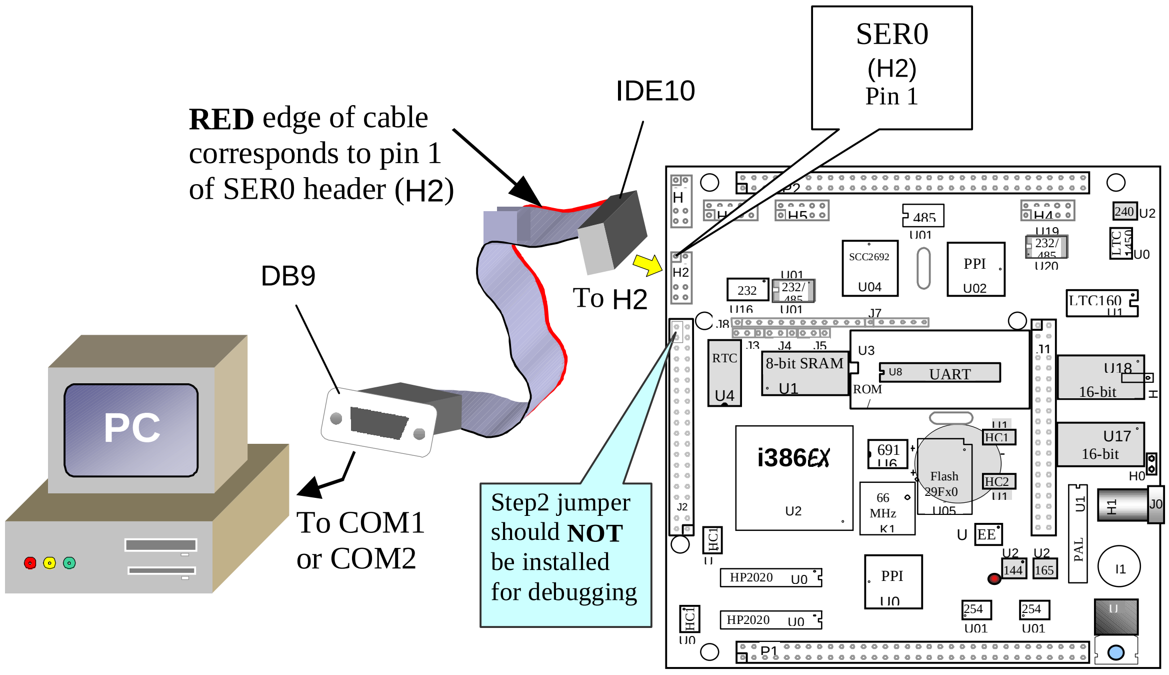

RED edge of cable corresponds to pin 1 of SER0 header (H2)
IDE10
SER0 (H2) Pin 1
DB9
To H2
PC
To COM1 or COM2
Step2 jumper should NOT be installed for debugging
i386EX
8-bit SRAM
UART
Flash 29Fx0
RTC
SCC2692
PPI
PAL

Figure 2.1 Connecting the i386-Drive to the PC

2.2.2 Powering-on the i386-Drive

Connect a wall transformer +9V DC output to the DC power jack.

The on-board LED should blink twice and remain on after the i386-Drive is powered-on or reset, as shown in Figure 2.2.

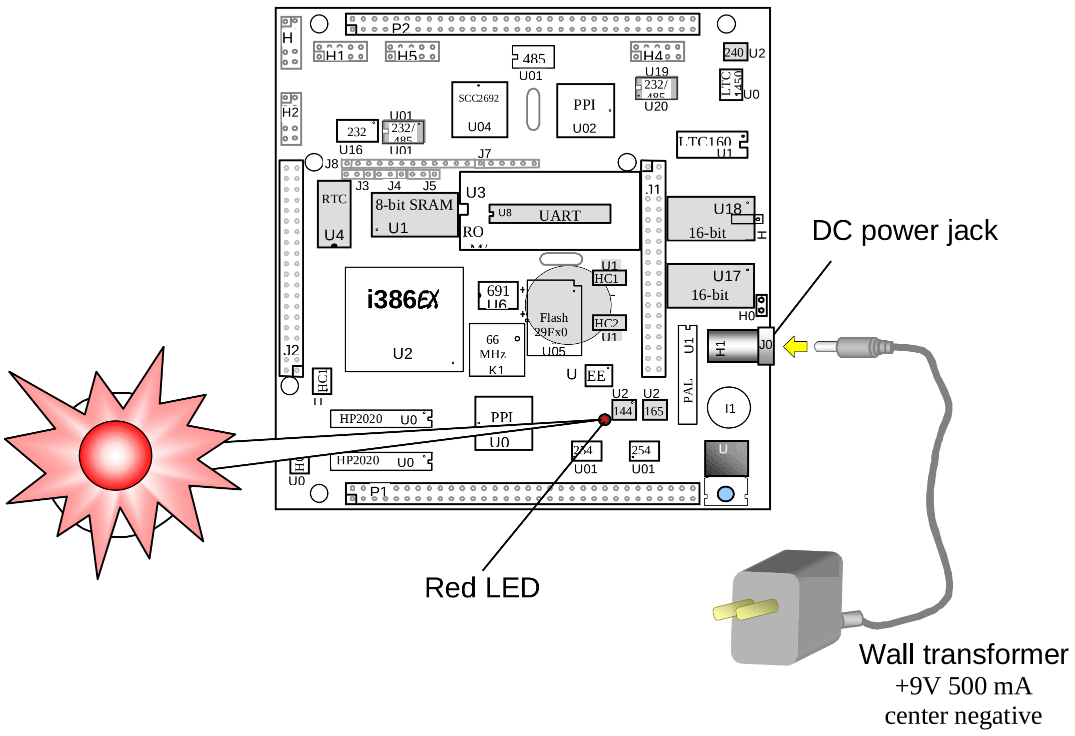


Figure 2.2 The LED blinks twice after the i386-Drive is powered-on or reset

Chapter 3: Hardware

3.1 Intel386EX Processor

The Intel386EX is based on the Intel386SX processor. 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-Drive 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-Drive Pin No.*	*i386-Drive 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	8-bit SRAM select
P21	CS1#	Output with pullup	J2 pin 37	U15 Flash select
P22	CS2#	Output with pullup	J2 pin 5	SCC & RTC I/O select
P23	CS3#	Output with pullup	J2 pin 10	16-bit SRAM select
P24	CS4#	Output with pullup	J2 pin 3	16-bit ADC/DAC select
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-Drive 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**
P21 = /CS1	J2.37	16-bit Flash U15
P22 = /CS2	J2.5	8-bit I/O for U4 RTC, U8 SCC, PPIs, SCC2692, /HP1, /HP2
P23 = /CS3	J2.10	16-bit SRAM U17+U18
P24 = /CS4	J2.3	16-bit I/O for high speed ADC/DAC
/CS5	(N/A)	U10 74HC259 chip for internal signals T0 to T7
RI1	J2.38 and P2.3	STEP 2 jumper
P15	U5.5	EEPROM SDA = U010 pin 16 ADC DOUT Shared with U010 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
DSR1	J2.4	U011 TLC2543 ADC DOUT or U08 HCTL2020 U/D to i386EX They cannot be used at the same time.
DCD1	J2.1	U06 HCTL2020 U/D to i386EX
P30	J2.17	Timer0 out as HCTL2020 clock
/INT4	J2.33	U14 16-bit ADC LTC1605 Busy
/INT5	J2.8	U8 SCC2691 UART interrupt
/INT6	J2.6	U04 SCC2692 DUAL UART interrupt

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

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. /INT6, J2 pin 6, is used by the on-board optional SCC2692 Dual 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.

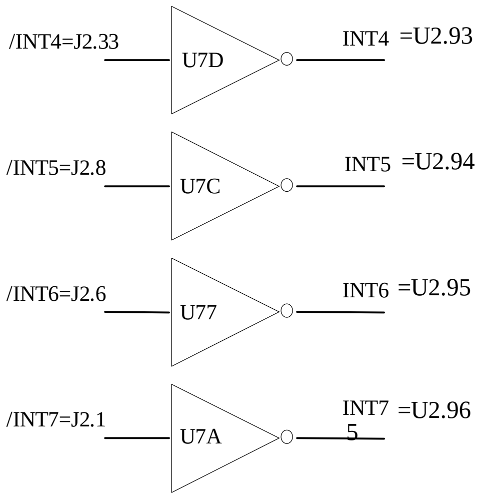


Figure 3.1 External interrupt inputs

The i386-Drive 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 (Use for U06/U08 HCTL2020(s), if installed)
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 **`c:\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-Drive 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 **`c:\tern\386\samples\ie`**.

3.6 Serial Ports

The i386-Drive can provide up to five asynchronous serial channels. Two are Intel386EX-internal: SER0, SER1. One external UART SCC2691 can be installed underneath the ROM. One optional dual UART SCC2692 can be installed. All of the UARTs can operate in full-duplex communication mode. SER0 and SER1 use DMA for receiving and for interrupt-driven transmit. The UART SCC2691 and Dual UART SCC2692 are interrupt-driven for both transmitting and receiving. For more information about the external SCC2691/2 UARTs, refer to Appendix C and the datasheets from the IC manufacturer (Philips Semiconductor, Sunnyvale, California, tel. 408-991-3737).

With the DEBUG ROM (*TDREM_i386EX*) installed, the internal serial port SER0 is used by the i386-Drive for DEBUG programming with the PC. It uses 115,000 Baud rate, as default, for programming. SER0 and SER1 can both be used 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-Drive can serve as a high-performance processor module for applications that require low power consumption. 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-Drive has a VOFF signal routed to J1 pin 9 and H0. 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.

See the sample program **poweroff.c** in the **`c:\tern\386\samples\ie`** directory.

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 i386EX 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 8-bit SRAM, U1. 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 8-bit ROM, U3. The U3 ROM socket supports both 8-bit ROM and 8-bit Flash chips. The default wait state for this component is two cycles, to allow use with ROM components with speeds of up to 120 ns. The preferred ROM speed is 70 ns, 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 the i386-Engine technical manual.

In certain applications, you might also choose to re-map the memory address space differently to other chip select lines. An optional 16-bit FLASH (29F400, U15) can be installed. The default setting uses P21=/CS1 as chip select. See the sample program **id_f.c** in the **`c:\tern\386\samples\id`** directory.

Two optional 8-bit SRAM chips can be installed in U17 and U18 to form up to 1MB 16-bit SRAM for the i386EX, using P23=/CS3 as chip select.

See the sample file ***id_ram.c*** in the **`c:\tern\386\samples\id`** directory.

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 C/C++ EV/DV Software 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 or MemCard-A
0xa000-0xa001	/CS2	J2 pin 5 = P22	Select U06 HP2020-1
0xa002-0xa003	/CS2	J2 pin 5 = P22	Select U08 HP2020-2
0xa004-0xa005	/CS2	J2 pin 5 = P22	RST1 for U06 HP2020-1
0xa006-0xa007	/CS2	J2 pin 5 = P22	RST2 for U06 HP2020-2
0xa080-0xa08f	/CS2	J2 pin 5 = P22	/LD for DAC U09 LTC1450
0xa090-0xa09f	/CS2	J2 pin 5 = P22	UART, SCC2691
0xa0a0-0xa0af	/CS2	J2 pin 5 = P22	RTC 72423
0xa0b0-0xa0bf	/CS2	J2 pin 5 = P22	/S1 for PPI1

I/O space	Select Signal	Location	Usage
0xa0c0-0xa0cf	/CS2	J2 pin 5 = P22	/S2 for PPI2
0xa0e0-0xa0ef	/CS2	J2 pin 5 = P22	/S4 for SCC2692
0xa0f0-0xa0ff	/CS2	J2 pin 5 = P22	/CLR for DAC U09 LTC1450
0xb000-0xb0ff	/CS5	None (**U9**-74HC259)	Internal Usage (T0-T7)
Not mapped	/CS0	N/A	SRAM
0x????	/CS1	J2 pin 37 = P21	16-bit Flash U15
0x????	/CS3	J2 pin 10 = P23	16-bit SRAM, U17 and U18
0xc000	/CS4	J2 pin 3 = P24	Read/write 16-bit ADC/DAC

A total of eight pre-decoded chip-select lines are available on the ID. These include the UCS (upper chip select), and signals CS0-6. The upper chip select is dedicated for boot-up ROM use.

The pre-decoded chip select lines listed in the table above can be used for application, if the on-board optional corresponding device is not installed.

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-Drive with external I/O boards, a simple decoding circuit for interfacing to an external 82C55 I/O chip is shown in Figure 3.2.

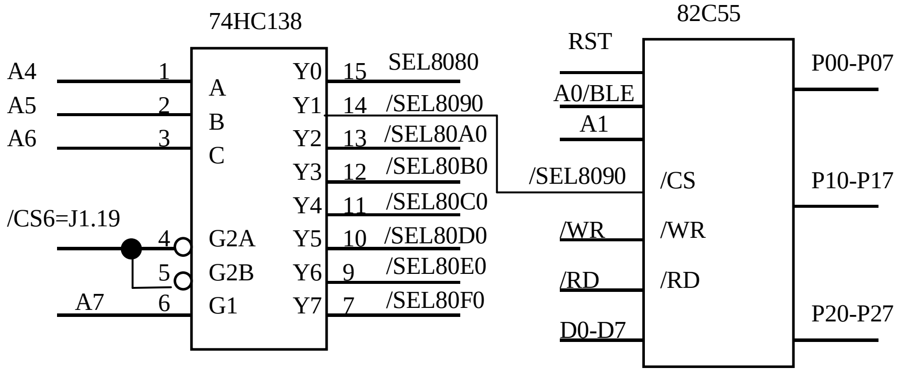


Figure 3.2 Interface i386-Drive to external I/O devices

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 Programmable Peripheral Interface (82C55A)

U02 and U01 PPIs (82C55, or uPD81055L) are low-power CMOS programmable parallel interface units for use in microcomputer systems. They each provide 24 I/O pins that may be individually programmed in two groups of 12 and used in three major modes of operation.

In MODE 0, the two groups of 12 pins can be programmed in sets of 4 and 8 pins to be inputs or outputs. In MODE 1, each of the two groups of 12 pins can be programmed to have 8 lines of input or output. Of the 4 remaining pins, 3 are used for handshaking and interrupt control signals. MODE 2 is a strobed bi-directional bus configuration.

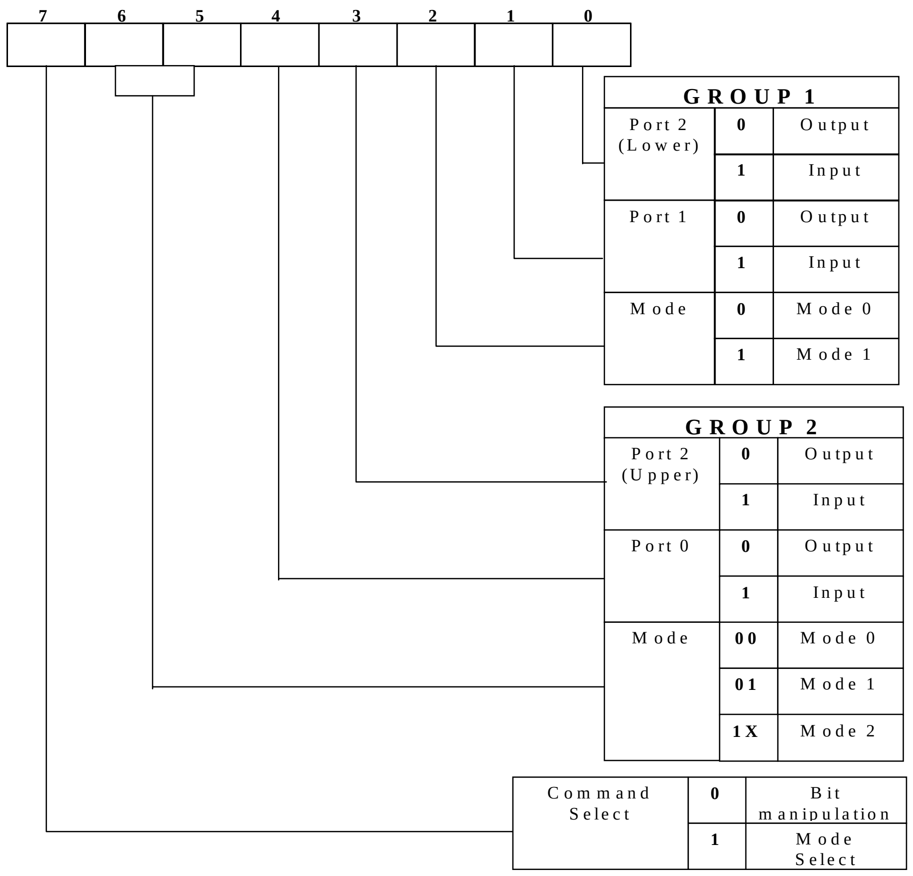


Figure 3.3 Mode Select Command Word

The i386-Drive maps U01, the PPI1 82C55/uPD71055, at base I/O address PPI1=0xa0b0.

The i386-Drive maps U02, the PPI2 82C55/uPD71055, at base I/O address PPI2=0xa0c0.

Use PPI1 as example, all ports/registers are offsets of this I/O base address.

The Command Register = PPI1+3; Port 0 = PPI1; Port 1 = PPI1+1; and Port 2 = PPI1+2.

The following code example will set all ports to output mode:

```
outportb(PPI1+3,0x80);/* Mode 0 all output selection. */
outportb(PPI1+0,0x55);/* Sets port 0 to alternating high/low I/O pins. */
outportb(PPI1+1,0x55);/* Sets port 1 to alternating high/low I/O pins. */
outportb(PPI1+2,0x55);/* Sets port 2 to alternating high/low I/O pins. */
```

To set all ports to input mode:

```
outportb(PPI1+3,0x9f);     /* Mode 0 all input selection. */
```

You can read the ports with:

```
inportb(PPI1+0); /* Port 0 */
inportb(PPI1+1); /* Port 1 */
inportb(PPI1+2); /* Port 2 */
```

This returns an 8-bit value for each port, with each bit corresponding to the appropriate line on the port.

There are a total of 48 TTL level I/O pins are free to use for your applications. These I/O lines are specified as 4 mA driving current capability.

See schematics for PPI connection headers of P1 and P2.

3.9.3 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 4, Software, for details).

3.9.4 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.

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

The RxD (J1 pin 5), TxD (J1 pin 7), and MPO (J1 pin 3) are TTL-level signals. You may choose to have RS-232 (U19) or RS-485 (U20) drivers installed on the ID board. The RS-232/485 signal is routed to H4.

3.9.5 UART SCC2692

The UART SCC2692 (Signetics, U04) is a 44-pin PLCC chip mapped into the I/O address space at **0xa0e0**. The SCC2692 includes two independent full-duplex asynchronous receiver/transmitters, 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.

A 3.6864 MHz external crystal can be installed on the ID, as the default crystal for the dual UART.

For more detailed information, refer to the SCC2692 data sheets (Signetics, tel. 408-991-3737).

Either RS-232 (default) or RS-485 drivers are supported for the Dual UART. The RS-232/485 signals for channel A are routed to the H1 header. The RS-232/485 signals for channel B are routed to the H5 header.

3.10 Other Devices

A number of other devices are also available on the i386-Drive. 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-Drive 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

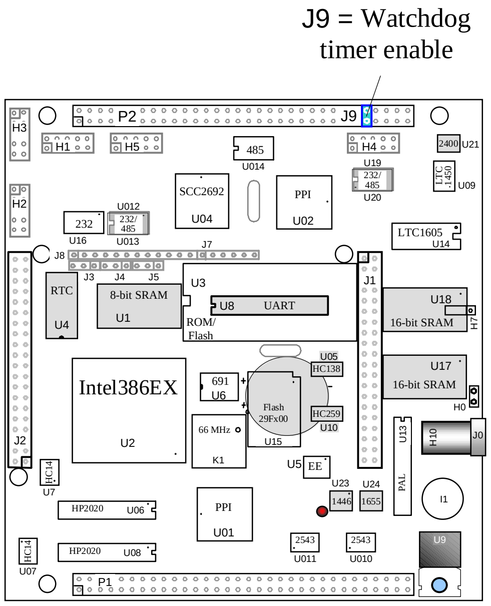


Figure 3.4 Location of watchdog timer enable jumper

The watchdog timer is activated by setting a jumper on J9 of the i386-Drive. 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-Drive 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 the on-board supervisor chip 691 senses power failure, 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 2 Kbytes (24C16) can be installed in U5 (512-byte 24C04 is default). The i386-Drive 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. The EEPROM can be read and written to 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 D of this manual.

3.10.3 12-bit ADC, TLC2543

Up to two 12-bit ADC surface-mount chips (TLC2543, TI) can be installed (U101, U011). The TLC2543 is a 12-bit, switched-capacitor, successive-approximation, 11-channel, serial interface, analog-to-digital converter. Four TTL I/O lines are required to handle the ADC: /CS (chip select=T2 or T5); SCK (clock to the chip=T0); DIN (serial command data to the chip=T1); and D12 (12-bit serial data output from the chip=P15 for U010, and DSR1 for U011). If the chip select line is low, the TLC2543 will have output on D12. If the chip select line is high, the TLC2543 is disabled and D12 is in high-impedance state. The serial access allows a conversion rate of up to approximately 10 KHz for a 33 MHz i386-Drive.

A reference voltage of VCC (+5V) can be provided to the 12-bit ADC **REF+** via P1 pin 59=pin 60. An external precision 2.5V-5.0V reference can be connected to the **REF+** pin via P1 pin 59.

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

Analog signal inputs are routed at the P1 37-58. A total of 22 channels of 12-bit ADC inputs are all routed to P1.

See the sample program **`id_ad12.c`** `in the` **`c:\tern\386\samples\id`** directory.

3.10.4 24-bit ADC, LTC2400

A single-channel 24-bit ADC surface-mount chip (LTC2400, Linear Technology) can be installed on the ID U21. The LTC2400 is a 24-bit analog-to-digital converter with an integrated oscillator. It uses delta-sigma technology, providing a typical conversion time of 160 ms. Based on the LTC2400 data sheets, it can provide 24-bit ADC data, with 4 ppm full-scale error with no missing codes.

A F0 signal is at P2 pin 57 to configure the LTC2400 for better than 110 dB noise rejection at 50 Hz (F0=GND, P2.57=P2.59) or at 60 Hz (F0=VCC, P2.57-P2.58). The 24-bit ADC REFA pin, U21 pin 2, is not connected. User may connect REFA=U21 pin 2 to the 5V at U21 pin1. The 24-bit ADC can also use an on-board external reference of 2.5V from DAC LT1450 U09 pin 19, or the 16-bit ADC LTC1605, U14.4. The 24-bit ADC communicates with i386EX via a 3-wire digital interface. Three TTL lines are required to drive a LTC2400: SCK=T6 (clock to the chip), /CS (chip select=T1), and D24=P35 (24-bit serial data output from the chip). If the chip select line (T1) is high, the TLC2400 is disabled, and D24=P35 line is in high-impedance state.

See sample program **`c:\tern\386\samples\id\id_ad24.c`**.

3.10.5 100 KHz 16-bit ADC, LTC1605

The LTC1605 (U14) is a 100 ksps, sampling 16-bit A/D converter that draws only 55 mW from a single 5V supply. This device includes sample-and-hold, precision reference, switched capacitor successive approximation A/D and trimmed internal clock.

The LTC1605 has an industry standard ±10V input range. Maximum DC specs include ±2.0 LSB INL and 16-bit no missing codes over temperature. An external reference can be used if greater accuracy is needed.

The ADC has a microprocessor compatible, 16-bit or two-byte parallel output port. The ID uses T6 to control the ADC's R/C pin and directly interface the full 16-bit data bus for maximum data transfer rate.

The LTC1605 requires 8 µs AD conversion time. The busy signal has an 8 µs low period indicating the conversion in process.

In order to get the 100 KHz sample rate, The ID can not use interrupt operation to acquire data. A polling method is demonstrated in the sample program **id_ad16.c** located in the **`c:\tern\386\samples\id`** directory.

3.10.6 Dual 12-bit DAC, LTC1446

The LTC1446 is a dual 12-bit digital-to-analog converter (DAC) in an SO-8 package. It is 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 40 Ω when driving a load to the rails. The buffer amplifiers can drive 1000 pf without going into oscillation.

The DAC is installed in U23 on the i386-Drive. The outputs are routed to header **P1** pins 61 and 62 for channels A and B. The DAC uses T0 as CLK, T1 as DI, and T4 as LD/CS. Please contact Linear Technology (tel. 408-432-1900) for LT1446 technical data sheets.

See the sample program **ie_da.c** in the **`c:\tern\386\samples\ie`** directory.

3.10.7 Parallel 12-bit DAC, LTC1450

The LTC1450 (U09) is a 12-bit parallel DAC with an internal reference. It has a voltage output of 0-4.095V at 12-bit resolution. The ID can write a full 12-bit data into the LTC1450 in a single I/O instruction. The typical voltage output slew rate is 1 V/µs, and the typical voltage output settling time is 14 µs. Please contact Linear Technology (tel. 408-432-1900) for LT1450 technical data sheets.

See the sample program **id_da12.c** in the **`c:\tern\386\samples\id`** directory.

3.10.8 16-bit DAC, LTC1655

The LTC1655 is a single 16-bit digital-to-analog converter (DAC) in an SO-8 package. It is complete with a rail-to-rail voltage output amplifier, an 2.048V internal reference and a 3-wire serial interface. The LTC1655 outputs a full-scale of 4.096V, making 1 LSB equal to 1/16 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 1000 pf without going into oscillation.

The 16-bit DAC is installed in U24 on the i386-Drive. The outputs are routed to header **P1** pins 63=V3. The DAC uses T0 as CLK, T1 as DI, and T3 as LD/CS. Please contact Linear Technology (tel. 408-432-1900) for LT1655 technical data sheets.

See the sample program **id_da16.c** in the **`c:\tern\386\samples\id`** directory.

3.10.9 HCTL2020

Two quadrature decoder/counter interface chips, (HCTL2020, Hewlett Packard, U08 and U06) can be installed on the ID. The quadrature decoder is used to interface incremental motion encoders with the microprocessor system or to improve system performance for digital closed-loop motion control systems. The HCTL2020 includes a quadrature decoder, a 16-bit counter, and an 8-bit bus interface. It features full 4x decoding, up to 14 MHz clock operation, high noise immunity due to the use of Schmitt-trigger inputs and digital noise filters, quadrature decoder output signals, up/down signal, count signals, and cascade output signal. Many types of optical incremental encoder modules, such as the HEDS-9000, HEDS-9100, and HEDS-9200 from HP, can be directly interfaced to the HCTL2020.

Channel A and B signals buffered with Schmitt trigger inputs (U07, 74HC14, CHA1/2, CHB1/2) are routed at pin 5, 6, 9, and 10 on headers P1. The HCTL2020 has built-in filters, which allow reliable operation in noisy environments.

Two software functions (found in `c:\tern\386\samples\id\id_hp.c`) are available to operate the quadrature decoders:

unsigned int ***pd_hp_rd***(char ch);
void ***pd_hp_reset***(char ch);

3.11 Headers and Connectors

3.11.1 Expansion Headers J1 and J2

Two 20x2, 0.1 spacing headers are installed on the i386-Drive for 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.

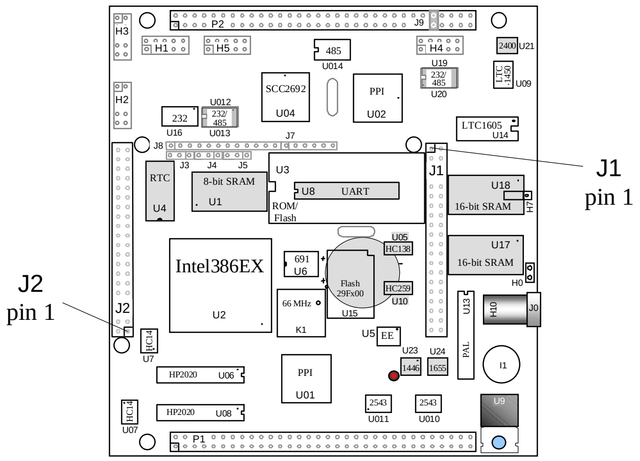


Figure 3.5 Pin 1 locations for J1 and J2

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 Headers P1 and P2

Two 32x2 pin headers, P1, and P2, provide signals for the ADCs, DACs, PPI, and quadrature decoders, as shown in Figure 3.6.

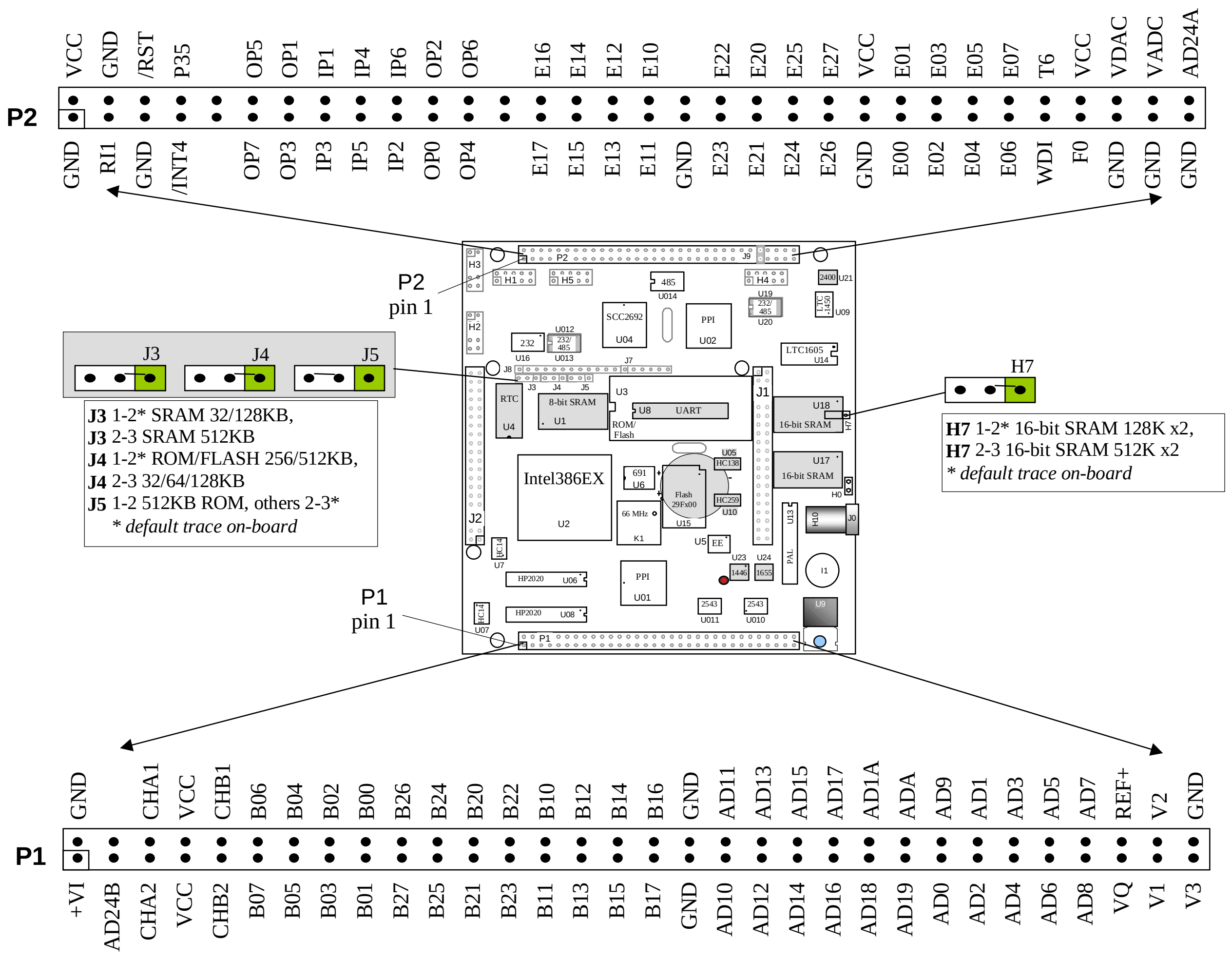

VCC GND /RST P35 OP5 OP1 IP1 IP4 IP6 OP2 OP6 E16 E14 E12 E10 E22 E20 E25 E27 VCC E01 E03 E05 E07 T6 VCC VDAC VADC AD24A
P2
GND RI1 GND /INT4 OP7 OP3 IP3 IP5 IP2 OP0 OP4 E17 E15 E13 E11 GND E23 E21 E24 E26 GND E00 E02 E04 E06 WDI F0 GND GND GND
P2 pin 1
J3 J4 J5
J3 1-2* SRAM 32/128KB,
J3 2-3 SRAM 512KB
J4 1-2* ROM/FLASH 256/512KB,
J4 2-3 32/64/128KB
J5 1-2 512KB ROM, others 2-3*
* default trace on-board
H7
H7 1-2* 16-bit SRAM 128K x2,
H7 2-3 16-bit SRAM 512K x2
* default trace on-board
P1 pin 1
Intel386EX
GND CHA1 VCC CHB1 B06 B04 B02 B00 B26 B24 B20 B22 B10 B12 B14 B16 GND AD11 AD13 AD15 AD17 AD1A ADA AD9 AD1 AD3 AD5 AD7 REF+ V2 GND
P1
+VI AD24B CHA2 VCC CHB2 B07 B05 B03 B01 B27 B25 B21 B23 B11 B13 B15 B17 GND AD10 AD12 AD14 AD16 AD18 AD19 AD0 AD2 AD4 AD6 AD8 VQ V1 V3

Figure 3.6 Signals on headers P1 and P2; SRAM and ROM/Flash selection headers

3.11.3 Jumpers and Headers

The jumpers and connectors on the i386-Drive are listed below.

Name	Size	Function	Possible Configuration
J1	20x2	main expansion port	(same as the i386-Engine and i386-Engine-P)
J2	20x2	main expansion port	(same as the i386-Engine and i386-Engine-P)
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

Name	Size	Function	Possible Configuration
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
H0	2x1	Switching power regulator	Enabled if jumper is on; Disabled is jumper is off VOFF = GND
H1	5x2	SCC2692 channel A: TXDA, RXDA, GND, 485A-, 485A+	
H2	5x2	SER0 (DEBUG)	
H3	5x2	SER1	
H4	5x2	SCC2691: TXD, RXD, GND	
H5	5x2	SCC2692 channel B: TXDB, RXDB, GND	
H7	3x1	16-bit SRAM selection:	pin 1-2, 128K x2, default pin 2-3, 512K x2
P1	32x2	HCTL2020, PPI, ADC (TLC2543), DAC (LTC1446 and LTC1655)	
P2	32x2	PPI, ADC (LTC2400 and LTC1605), DAC (LTC1450)	

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-Drive 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,
ID.H	PPI, ADC, DAC, HCTL2020
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-Drive Initialization

ie_init

This function should be called at the beginning of every program running on i386-Drive 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-Drive 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-Drive, 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-Drive. 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 two 12-bit ADC units (TLC2543, U010 and U011) each provide 11 channels of analog inputs based on the reference voltage supplied to **REF+**. For details regarding the hardware configuration, see the Hardware chapter.

The U010 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 files demonstrating the use of these ADC, please see **ie_ad12.c** in **`c:\tern\386\samples\ie`** (for U010), and **id_ad12.c** in **`c:\tern\386\samples\id`** (for U011).

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

One dual 12-bit DAC (LTC1446) mad be installed on the i386-Drive in positions **U23**. It 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 U23 DAC can be found in **ie_da12.c** in the directory **`tern\386\samples\ie`**.

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

Ie_da() is used for the DAC chip installed in position U23.

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

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 rollover in digits in the year 2000. One solution might be to store an offset value in non-volatile storage such as the EEPROM.

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

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

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

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

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 that are specific to SER1 can be easily changed into function calls for SER0. While selecting a serial port for use, please realize that some pins might be shared with other peripheral functions. This means that in certain limited cases, it might not be possible to use a certain serial port with other on-board controller functions. For details, you should see both chapter 11 of the Intel 386EX Embedded Microprocessor User's Manual and the schematic of the i386-Drive 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.

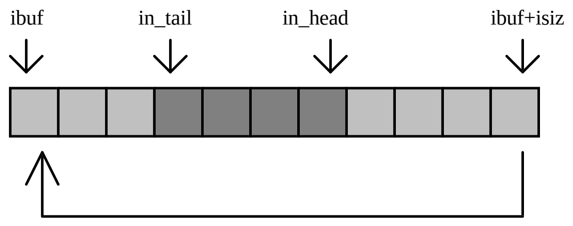


Figure 4.1 Circular ring input buffer

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.

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

Function Argument	Baud Rate
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_s*n***

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

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

4.5 Functions in 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 U010 ADC on the i386-Drive, 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

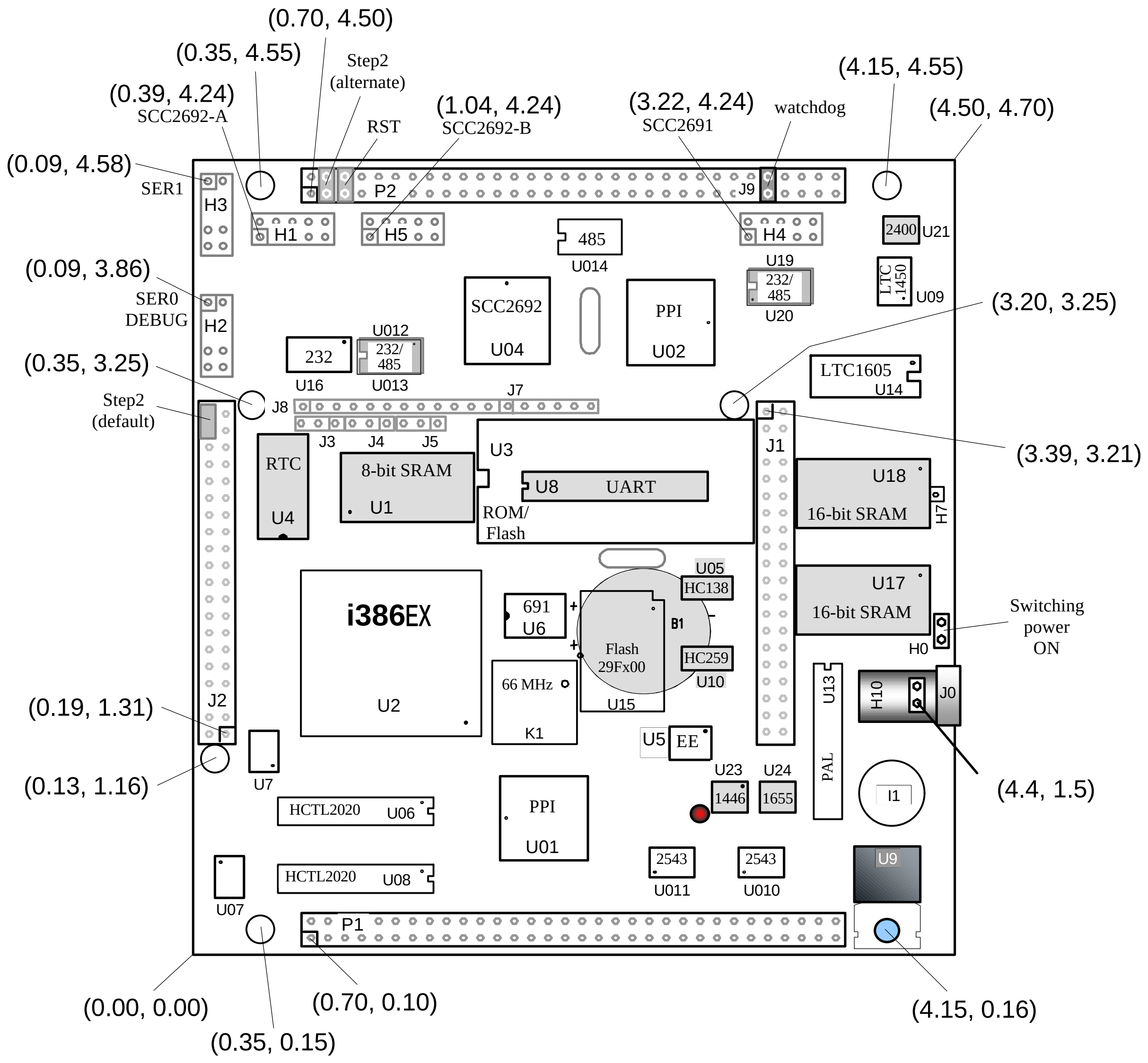
This function is used to write the passed in **dat** to the specified **addr**. The return value is 0 in success.

ee_rd
Arguments: int addr
Return value: int data

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

Appendix A: i386-Drive Layout

The **i386-Drive** measures 4.7 x 4.5 inches. Its layout is shown below.
All dimensions are in inches.



Appendix B: 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] = 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] = 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			

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

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 C: 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-minute 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 D: 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 ACTR™		
0x11	CS low byte, used by ACTR™		
0x12	IP high byte, used by ACTR™		
0x13	IP low byte, used by ACTR™		
0x18	MM page register 0		
0x19	MM page register 1		
0x1a	MM page register 2		
0x1b	MM page register 3		

Appendix E: 16-bit Flash/RAM Programming

1. Overview

The TERN i386-Engine-P (IE-P) and i386-Drive (ID) support 16-bit Flash and 16-bit RAM. The TERN ACTF Flash Kit now supports on-board programming/execution of the 16-bit Flash.

2. Minimum Requirements

- TERN Development Kit (DV-Kit)
- ACTF Flash Kit
- i386-Engine-P or i386-Drive with 256K Flash and 256K RAM
- TD_IE_16 Debug ROM

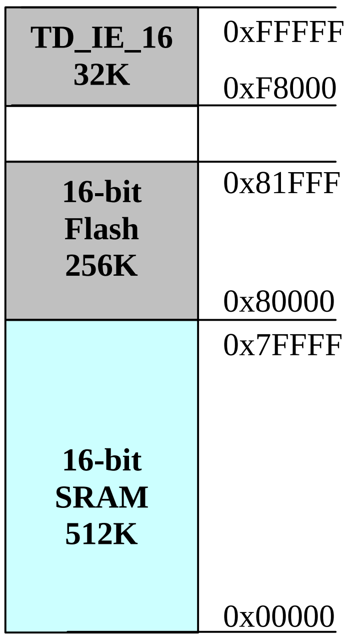


Figure 1 **TD_IE_16 memory mapping configuration**

3. Memory Mapping

Memory for the 16-bit Flash configuration is shown in 2. Figure 1. The TD_IE_16 Debug ROM is located at the top of the memory map and is the first block to execute after power-on/reset. At power-on/reset, TD_IE_16 selects the dual chip 16-bit SRAM as memory.

3.1 Generating a HEX File

You must modify the MAKEFILE to generate a HEX for the 16-bit Flash. Modify the BOARD flag to IEP16 or ID16 respectively. Use the flash512.rm configuration file when generating HEX files. *See the ACTF Flash Kit manual for the rest of the details about generating a HEX file.*

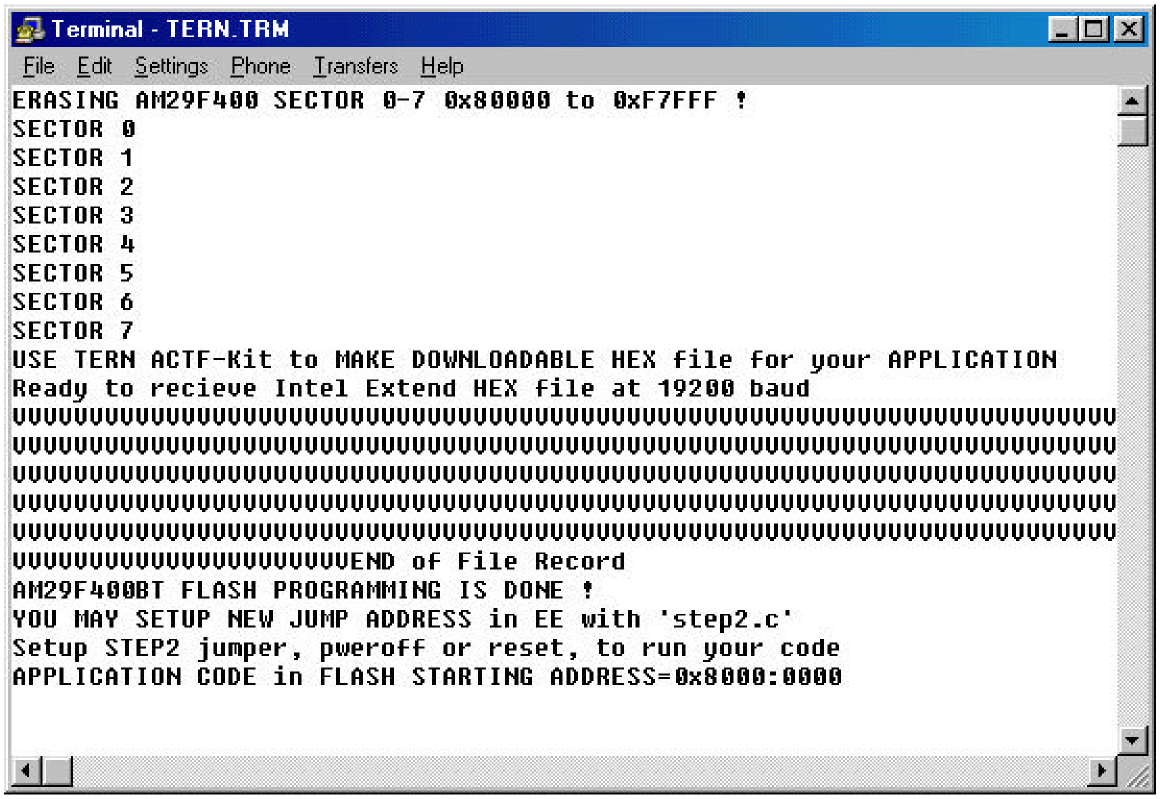
3.2 Downloading a HEX file into the 16-bit Flash

Be sure that the step 2 address is set up correctly. If you are not sure, run ***step2.c*** *in the debugger.*

The downloading process requires an intermediate loading program, ***l_f16.c***, to prepare the 16-bit flash and receive the final HEX file. ***l_f16.c*** is located in `C:\TERN\ACTF386`. Copy ***l_f16.c*** into the `C:\TERN\386` directory.

- Install TD_IE_16 Debug ROM in ROM socket..
- Use ***t.bat*** to download ***l_f16.c*** via Turbo Debugger. *Do not run the code in the debugger*.
- Exit the debugger and set the Step 2 jumper.
- Exit DOS and open a terminal window. Set baud rate for 19200.
- Reset the board by shorting J1 pin15 = /RST and J1 pin 13 GND. *The 16-bit SRAM is not battery-backed. Do not power off the board to reset the board.*
- The ***l_f16*** program will request your hex file. Use *Send Text File* to transfer your HEX file to the board.
- The program will modify your step 2 address to 0x80000.
- Power off and on the board to reset. With the Step 2 jumper on, your code should be executing from the 16-bit flash.

Figure 2 shows a sample session with ***l_f16***.



```
ERASING AM29F400 SECTOR 0-7 0x80000 to 0xF7FFF !
SECTOR 0
SECTOR 1
SECTOR 2
SECTOR 3
SECTOR 4
SECTOR 5
SECTOR 6
SECTOR 7
USE TERN ACTF-Kit to MAKE DOWNLOADABLE HEX file for your APPLICATION
Ready to recieve Intel Extend HEX file at 19200 baud
VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
VVVVVVVVVVVVVVVVVVVVVVVVEND of File Record
AM29F400BT FLASH PROGRAMMING IS DONE !
YOU MAY SETUP NEW JUMP ADDRESS in EE with 'step2.c'
Setup STEP2 jumper, pweroff or reset, to run your code
APPLICATION CODE in FLASH STARTING ADDRESS=0x8000:0000
```

Figure 2 Sample session

Appendix F:
Special TDREM_IE DCD1 and Multi-function Pins

A special debug ROM TDREM_IE DCD1 is designed to use J2 pin 1=DCD1 as STEP2 jumper.
There are several pins are sharing functions on i386-Drive design. Please check and modify your hardware and software to suit your application.

DCD1=J2.1=U06.5 (HP2020 U/D)=STEP2
DSR1=J2.4=U08.5 (HP2020 U/D) = U011.16 (ADC data out)
T5 = LED = U011.15 (ADC chip enable)

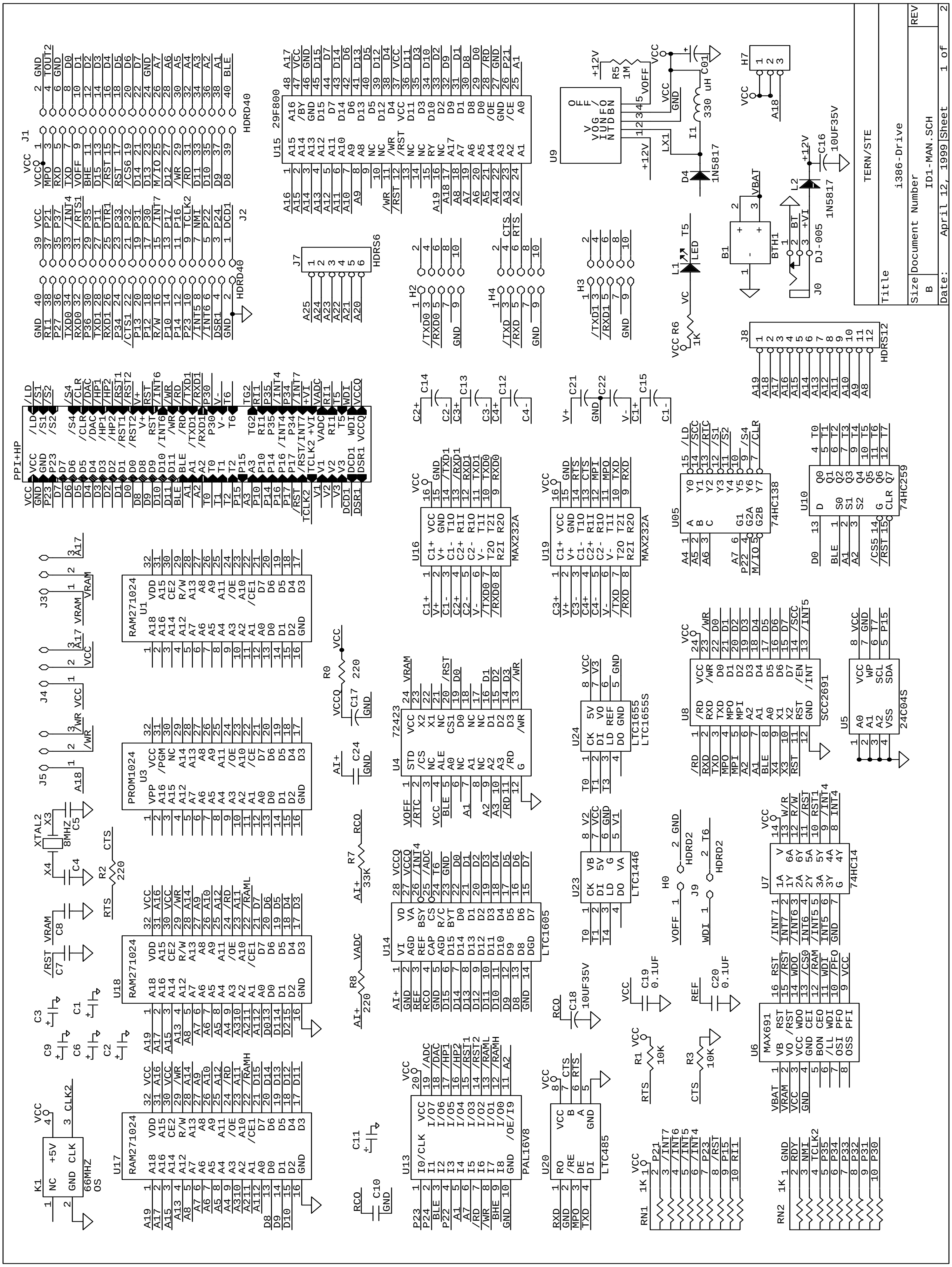
The DCD1 must be pull up to VCC via a 10K resistor, so it is high at power-on.
The U06.5 HP2020 pin 5 must be cut, in order to allow DCD1 STEP2 work.

DSR1 is shared with U08 HP2020 pin 5, U011 ADC pin 16, and STXCLK.
If you want to use DSR1 as STXCLK, you must cut HP2020 U08 pin 5, and
Use software to disable U011 ADC with T5 high, or LED off.

Some of your problem may be related to the above conflict. Please remember, the DCD1 ROM is only special developed, and is not used or tested by many users.

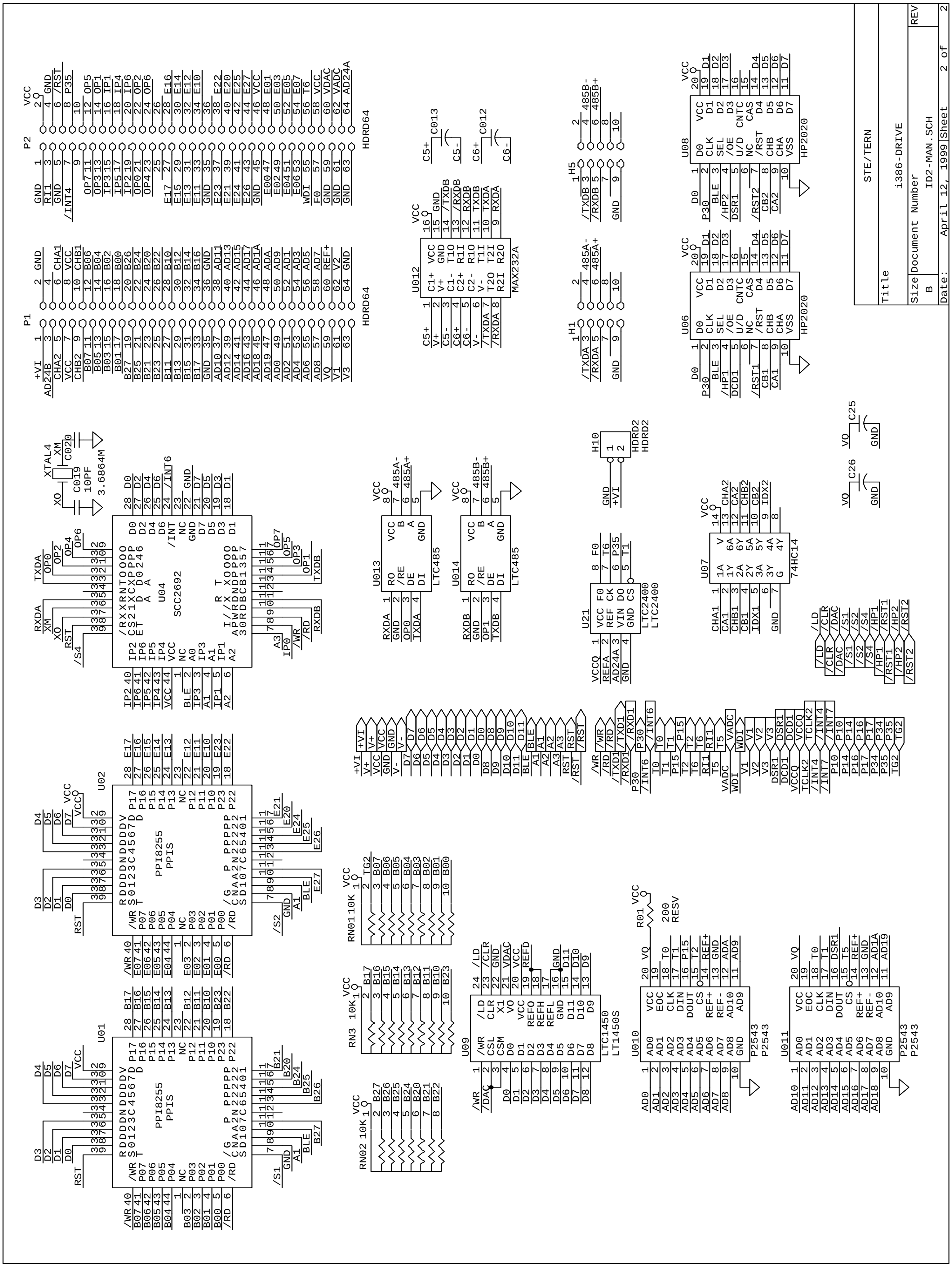
In order to use SSIO, use DCD1 DEBUG ROM, and run STEP2,
1) You may have to cut off HP2020 pin 5.
2) Try to run c:\tern\386\samples\ie\step2.c once to setup a correct jump address.
3) Download led.c program, power off.
4) Setup STEP2 jumper at J2 pin 1=2 for DCD1=GND, then power on.
5) Led should running in STEP2. If you can not make STEP2 work and tested in Standalone, there is no chance or reason that you can make ACTF work.
6) Make sure you can make the "led.c" works in STEP2.
7) You must turn the LED off, or T5 high, in order to disable the U011 ADC (if installed).
8) If the ADC is installed, and if you are turn ON the LED, which means T5=low, the ADC will be enabled and hold ADC data out pin = DSR1 low, then The SSIO transmit will not work.

Please refer to i386-Drive schematics for details.



TERN/STE

Title	i386-Drive	
Size	Document Number	REV
B	ID1-MAN.SCH	
Date:	April 12, 1999	Sheet 1 of 2



STE/TERN

Title	i386-DRIVE	
Size	Document Number	REV
B	ID2-MAN.SCH	
Date:	April 12, 1999	Sheet 2 of 2