ECAN1000HR
Isolated CAN interface board

User's Manual

RTD Embedded Technologies, Inc.

(Real Time Devices)

"Accessing the Analog World"®

ISO9001 and AS9100 Certified

BDM-610020026
Rev. B

ECAN1000HR ISOLATED 1 MB/S CAN INTERFACE BOARD USER'S MANUAL

RTD Embedded Technologies, INC.
103 Innovation Blvd.
State College, PA 16803-0906

Phone: +1-814-234-8087
FAX: +1-814-234-5218

E-mail
sales@rtd.com
techsupport@rtd.com

Web site
http://www.rtd.com

Revision History

28/11/1991	Release 1.0
15/07/2001	Name of company changed, reformatted
Rev. A	New Manual Naming Method
Rev. B 05/04/2007	Clarified references to QNX support, improved formatting and grammar

Published by:

RTD Embedded Technologies, Inc.
103 Innovation Blvd.
State College, PA 16803-0906

TABLE OF CONTENTS

ILLUSTRATIONS

TABLES

Chapter 1 INTRODUCTION

This user's manual describes the operation of the ECAN1000HR CAN bus Interface board.

Features

Some of the key features of the ECAN1000HR include:

- SJA1000 CAN-network controller, Electrically compatible with the PCA82C200 stand-alone CAN controller chip
- 1 Mb/s maximum data rate (fully programmable)
- Full CAN-functionality 2.0 B
- Extended receive buffer (64 byte FIFO)
- 16 MHz clock frequency
- Galvanically isolated physical interfaces
- I/O mapped host interface using three addresses
- -40 to +85C operational temperature
- +5V only operation
- PC/104 compliant

The following paragraphs briefly describe the major features of the ECAN1000HR. A more detailed discussion in included in Chapter 4 (Hardware description) and in Chapter 5 (Board operation and programming). The board setup is described in Chapter 2 (Board Settings). A full description of the Philips SJA1000 CAN-controller is included in Chapter 5 (Board operation and programming).

CAN bus controller

The ECAN1000HR CAN bus interface is implemented using the Philips SJA1000 controller. This controller supports CAN Specification 2.0. This versatile chip supports standard and extended Data and Remote frames; a Programmable Global Message Identifier Mask; 15 message objects of 8-byte Data Length and a Programmable Bit Rate. This fully integrated chip supports all the functionality of the CAN bus protocol. The internal 64 byte receive FIFO is ideal for block mode data transfer from the CAN controller chip.

Physical Interface

Industrial environments require galvanic isolation and bus filtering to provide reliable data communication and safety. The *galvanically isolated physical interface* is uses high speed optocouplers and a DC/DC converter. To protect the input from radiated bus noise a specially balanced bus filter is used. The bus connectors conform to the ISO11898 /2 specification. (For more information on CAN bus please visit the CAN in Automation Website at**: http://www.can-cia.de**.)

Mechanical description

The ECAN1000HR is designed on a PC/104 form factor. An easy mechanical interface to both PC/104 and EUROCARD systems can be achieved. Stack your ECAN1000HR directly on a PC/104 compatible computer using the onboard mounting holes.

Connector description

There is a 10-pin interface connector on the ECAN1000HR to directly interface to the galvanically isolated CAN-networks. This header is compliant with the ISO11892/2 specified pinout.

What comes with your board?

Your ECAN1000HR package contains the following items:

- ECAN1000HR CAN bus interface module
- User's manual

Note: Software and drivers can be downloaded from our website

If any item is missing or damaged, please call RTD Embedded Technologies, Inc. customer service department at the following number: (814) 234-8087.

Board accessories

In addition to the items included in your ECAN1000HR delivery, several accessories are available. Contact your local distributor for more information and for advice on selecting the most appropriate accessories to support your system.

Using this manual

This manual is intended to help you install your new ECAN1000HR card and get it working quickly, whilst also providing enough detail about the board and its functions so that you can obtain maximum use of its features even in the most demanding applications. This manual does not cover CAN bus network programming and system design.

When you need help

This manual and all the example programs will provide you with enough information to fully utilize all the features on this board. If you have any problems with installation or use of the board, contact our Technical Support Department (814) 234-8087. Alternatively, send an Email to: **sales@rtd.com.** When sending an Email request, please include the following information: Your company's name and address, your name, your telephone number, and a brief description of the problem.

Chapter 2 BOARD SETTINGS

The ECAN1000HR CAN bus interface board has jumper settings which can be changed to suit your application and host computer memory configuration. The factory settings are listed and shown in the diagram at the beginning of this chapter.

Factory-Configured Jumper Settings

Table 1 below illustrates the factory jumper setting for the ECAN1000HR. Figure 1 shows the board layout of the ECAN1000HR and the locations of the jumpers. The following paragraphs explain how to change the factory jumper settings to suit your specific application.

Table 1: Factory configured jumper settings (Please, see figure 1 for detailed locations)

JUMPER NAME	DESCRIPTION	NUMBER OF JUMPERS	FACTORY SETTING
JP1 BASE	Base Addresses	8	300H
JP5 IRQ SELECT	Host interrupts	10	5
J34	CAN bus termination	1	Closed

Table 1: Factory configured jumper settings

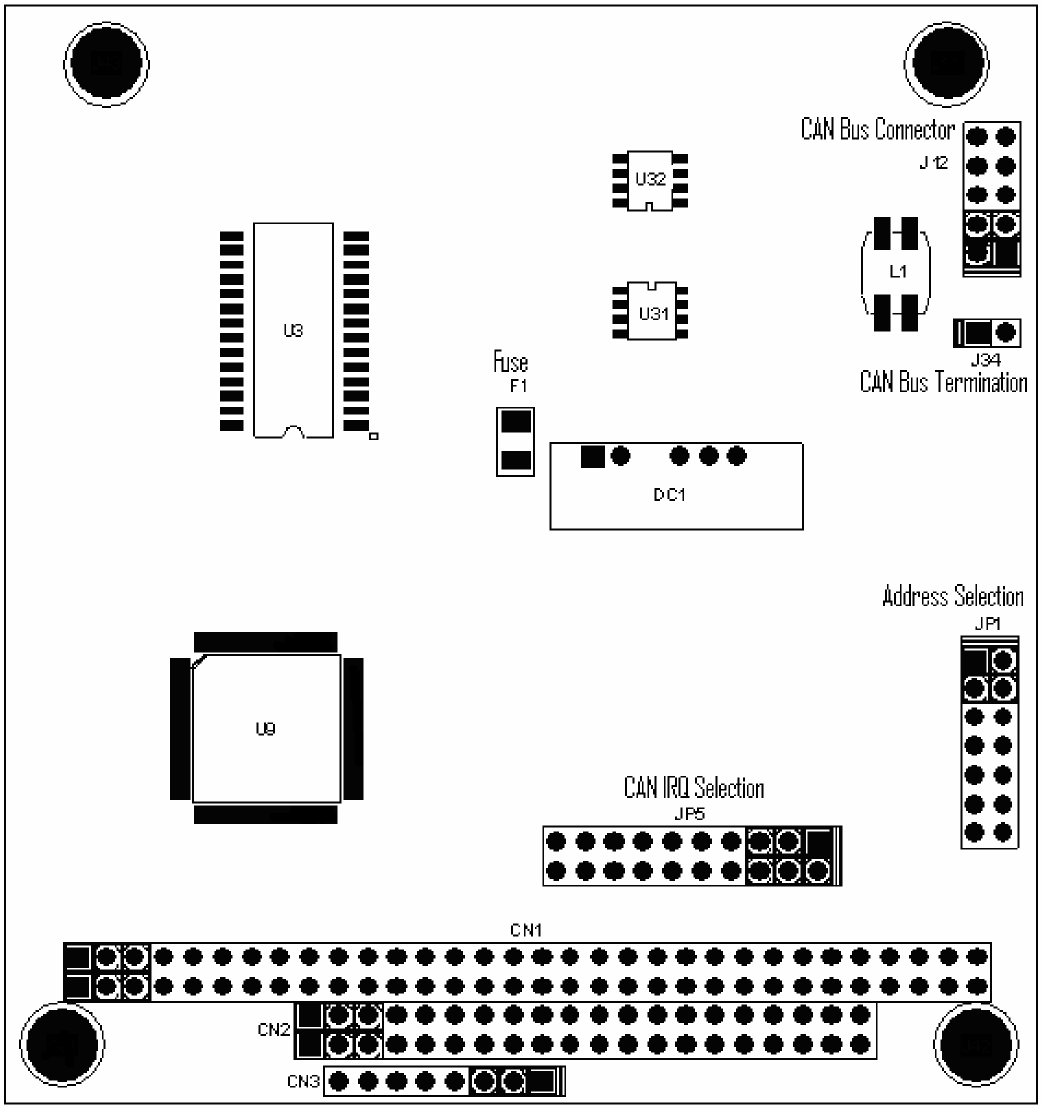

Figure 1: Component Locations

Base Address Jumpers (Factory setting: 300h)

The ECAN1000HR is I/O mapped into the memory space of your host XT/AT. This board occupies two I/O addresses starting from the base address.

The most common cause of failure when you are first setting up your module is address contention: Some of your computers I/O space is already occupied by other devices and memory resident programs. When the ECAN1000HR attempts to use its own reserved memory addresses (which are being already used by another peripheral device) erratic performance may occur and the data read from the board may be corrupted.

To avoid this problem make sure you set up the base address by using the ten jumpers marked “BASE”. This allows you to choose from a number of different addresses in your host computer I/O map. Should the factory installed setting of 300h be incompatible to your system configuration, you may change this setting to another using the options illustrated in Table 2. The table shows the jumper settings and their corresponding values in hexadecimal form. Ensure that you verify the correct location of the base address jumpers. When the jumper is removed it corresponds to a logical "0", connecting the jumper to a "1". When you set the base address of the board, record the setting inside the back cover of this manual

BASE ADDRESS JUMPER SETTINGS ECAN1000HR

Base address Hex	Jumper Settings A8 A7 A6 A5
200	0 0 0 0
220	0 0 0 1
240	0 0 1 0
260	0 0 1 1
280	0 1 0 0
2A0	0 1 0 1
2C0	0 1 1 0
2E0	0 1 1 1
300	**1 0 0 0**
320	1 0 0 1
340	1 0 1 0
360	1 0 1 1
380	1 1 0 0
3A0	1 1 0 1
3C0	1 1 1 0
3E0	1 1 1 1

0 = JUMPER OFF **1 = JUMPER CLOSED**

Table 2: JP1 Base Address Jumper Settings

Note: The above table illustrates the settings for the high address bits A8-A5, A9 is always decoded “1".

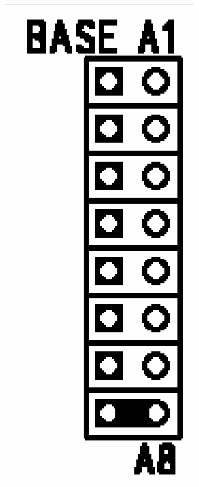

Figure 2: Base address jumpers illustrating address 300 h

Interrupt channel (Factory setting: IRQ5)

The header connector, shown in Figure 3 below, lets you connect the on-board SJA1000 CAN controllers interrupt outputs to one of the interrupt channels available on the host XT/AT bus. If your board has no AT-extension interrupts, IRQ 10-15 are not available.

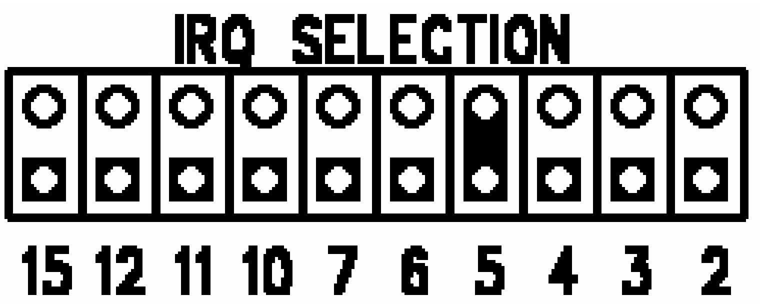

Figure 3: Interrupt set to IRQ 5.

Note: The ECAN1000HR does not support interrupt sharing! This feature is sometimes regarded as a part of the PC/104 special features. After extensive software and hardware tests we have found that error free interrupt performance can not be guaranteed when sharing interrupts.

Chapter 3 BOARD INSTALLATION

The ECAN1000HR CAN bus interface board is very easy to connect to your industrial distributed control system. Direct interface to PC/104 systems as well as EUROCARD boards is possible. This chapter gives step-by-step instructions on how to install the ECAN1000HR into your system.

After completing the installation it is recommended that you use the diagnostic and test software to fully verify that your board is working.

Board Installation

Keep your board in the antistatic bag until you are ready to install it to your system! When removing it from the bag, hold the board at the edges and do not touch the components or connectors. Please handle the board in an antistatic environment and use a **grounded** workbench for testing and handling of your hardware. Before installing the board in your computer, check the jumper settings. Chapter 2 reviews the factory settings and how to alter them. If any alterations are needed, please refer to the appropriate instructions in this chapter. Do however note that incompatible settings can result in unpredictable board operation and erratic response.

General installation guidelines:

- ***Turn OFF the power to your computer and all devices connected to the ECAN1000HR.***
- Touch the grounded metal housing of your computer to discharge any antistatic build-up and then remove the board from its antistatic bag.
- Hold the board by the edges and install it in an enclosure or place it on the table on an antistatic surface.
- Connect the board to the CAN field bus using the field bus interface header connector J12. Make sure that the orientation of the cable is correct.

Installation integrated with a PC/104 module stack

- Secure the four PC/104 installation holes with standoffs.
- Connect the board to the CAN bus using the CAN header connector J12.

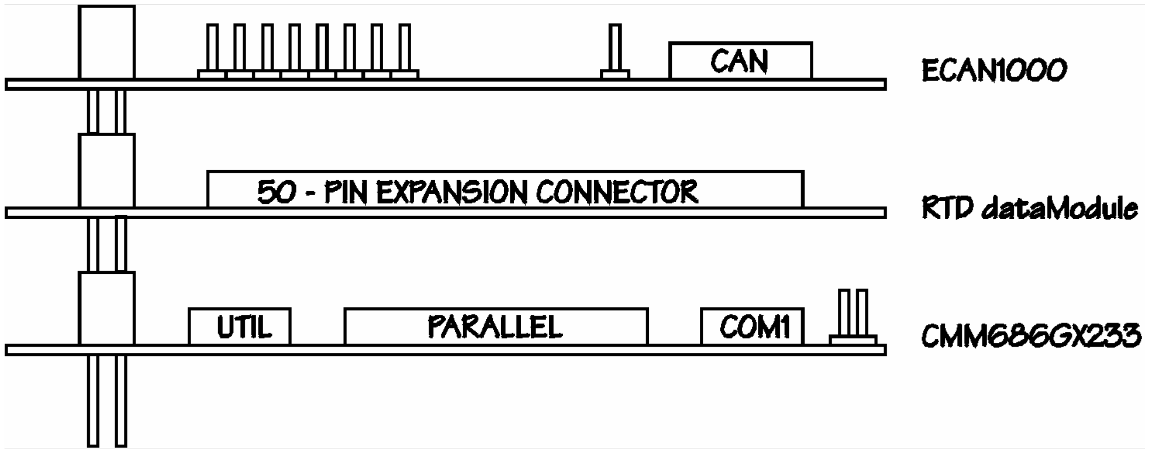

Figure 4: ECAN1000HR integrated in a PC/104 RTD cpuModule stack

3U rack or enclosure installation with a EUROCARD CPU containing an ECAN1000HR

The PC/104 system can be easily inserted into a 19" rack installation using the CPU as a "form factor adaptor". Assemble your PC/104 data modules on an RTD single board EUROCARD computer and install the system in a 19" enclosure. Multiple ECAN1000HR boards can be easily connected to this system. See figure 5 below.

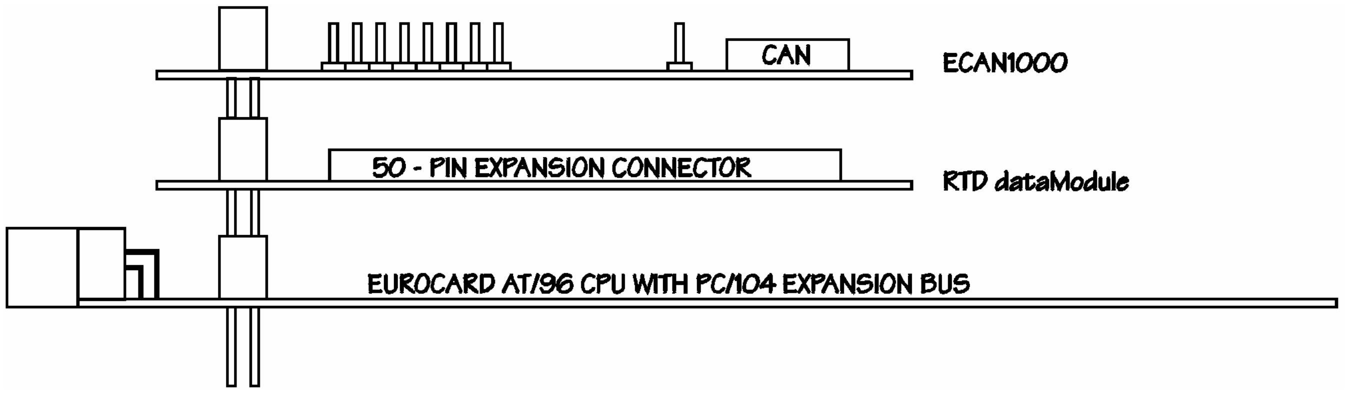
CAN
ECAN1000
50 - PIN EXPANSION CONNECTOR
RTD dataModule
EUROCARD AT/96 CPU WITH PC/104 EXPANSION BUS

Figure 5: 19" Eurocard rack installation with an integrated PC/104 dataModule and EUROCARD cpuModule computer system

J12 Galvanically isolated CAN bus connector

Table 3 below shows the CAN physical interface connector pinout. This connector is to the right hand side of your board marked J12. The pinout conforms to the ISO 11898/2 standard specification.

PIN number	Function
1	N.C.
2	GND_isolated
3	BUS_L
4	BUS_H
5	GND_isolated
6	N.C.
7	N.C.
8	+5V-isolated
9	GND-isolated
10	N.C.

Table 3: J12 CAN Bud Connector

9	7	5	3	1
I_GND	N.C.	I_GND	BUS_L	N.C.
N.C.	I_+5V	N.C.	BUS_H	I_GND
10	8	6	4	2

Table 4: Physical interface connector J12 pinout of the ECAN1000HR

J34 Galvanically isolated CAN bus termination jumper

The jumper marked as J34 is the CAN bus termination jumper. Only two termination jumpers should be closed at the endpoints of the CAN bus. Failure to do so may degrade the performance of the bus and it will affect the bus timing characteristics of the CAN bus. The maximum guaranteed drive capability of the CAN bus transceiver is 32 nodes.

Chapter 4 HARDWARE DESCRIPTION

This chapter describes in detail the major features of the ECAN1000HR:

- The Philips SJA1000 CAN bus controller
- Galvanic isolation of the CAN bus

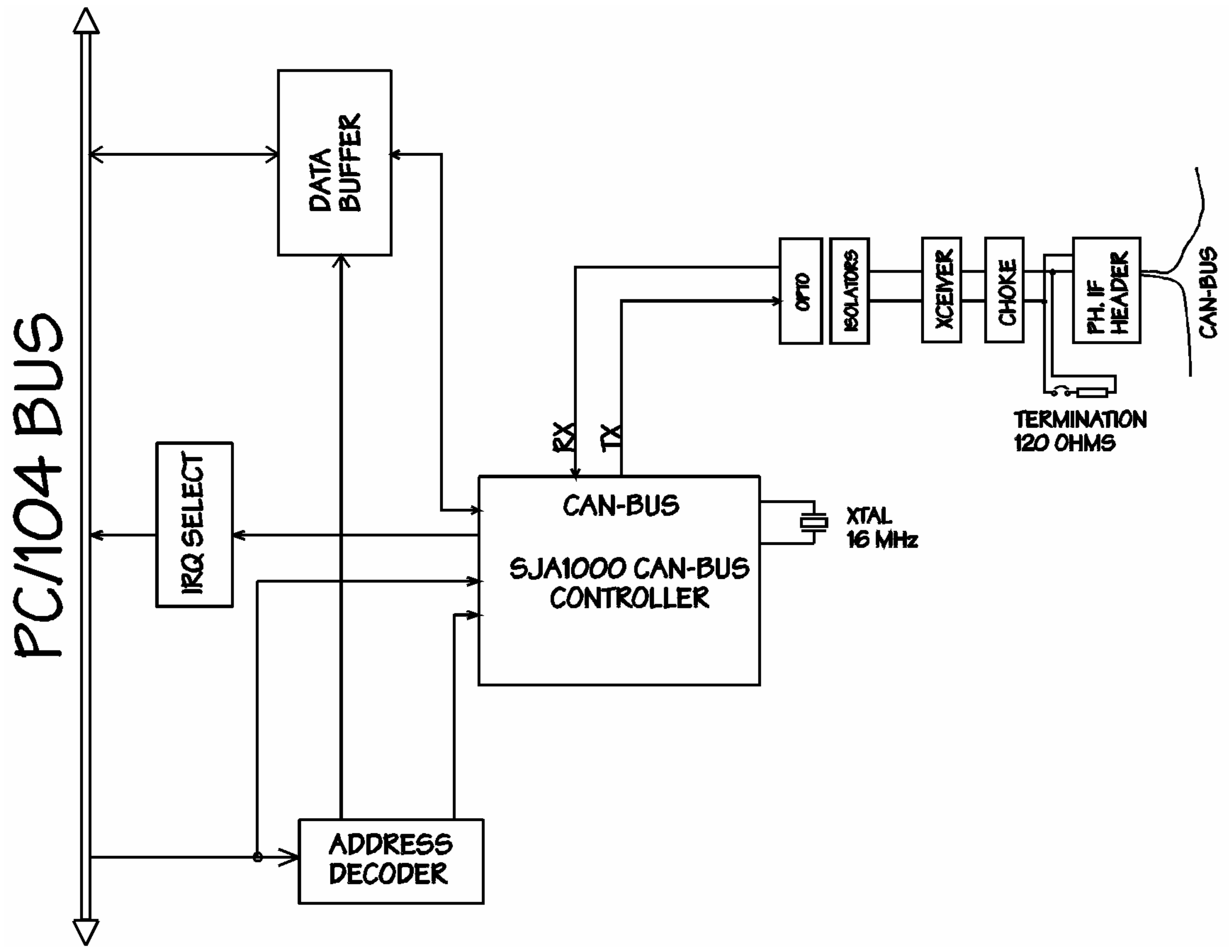

Figure 6: ECAN1000HR Block diagram

The CAN bus controller

The SJA1000 CAN bus controller uses a 16 MHz base oscillator. This must be taken into account when performing settings in the CAN bus timing registers that set the baud rate and sampling times of the CAN network.

The SJA1000 CAN controller consists of seven functional blocks. The host interface logic; the Transmit Buffer; the Receive Buffer; the Acceptance Filter; the Bit Stream Processor; the Bit Timing Logic and the Error management logic. A detailed description of these blocks is listed in the detailed component specific datasheet reprinted from Philips Semiconductors.

The SJA1000 internal message FIFO RAM provides storage for 64 message bytes. Each message can vary from one to up to 8 bytes in length. Each message object has a unique identifier and can be configured to either transmit or to receive.

Each message identifier contains control and status bits. A message object with a direction set for receive will send a remote frame by requesting a message transmission. A message set as transmit will be configured to automatically send a data frame whenever a remote frame with a matching identifier is received over the CAN bus. All message objects have separate transmit and receive interrupts and status bits, allowing the CPU full flexibility in detecting when a remote frame has been sent or received.

The SJA1000 also features masking for acceptance filtering. This feature allows the user to globally mask, or "don't care", any identifier bits of the incoming message. This mask is programmable to allow the user to design an application specifically message identification strategy. There are separate global masks for standard and extended frames. The incoming message first passes through the global mask and is then matched to the identifiers in the message objects

Galvanic isolation of the CAN bus

The galvanic isolation of the ECAN1000HR is implemented using the following:

- Optocouplers for reliable data transmission
- DC/DC converter to supply power to the CAN bus and the physical interface circuitry.

The high-speed optocouplers are directly connected to the SJA1000. The optocouplers drive the CAN bus transceiver. A special balanced CAN bus choke is used not only to improve immunity to bus noise, but also to protect the bus transceiver. This choke also reduces the radiated emissions in the range of 30-200MHz.

A 1W DC/DC converter may be used to power other remote devices on the CAN bus. The output power of this converter is isolated up to 1.5 kV peak. A 125mA fuse (green) are used to protect the DC/DC converter.

The jumper marked as J34 is the CAN bus termination jumper. Only two termination jumpers should be closed at the endpoints of the CAN bus. Failure to do so may degrade the performance of the bus or even cause permanent damage to the driver chips. The maximum drive of the transceiver is 32 nodes.

Chapter 5 BOARD OPERATION AND PROGRAMMING

This chapter shows you how to program and use your ECAN1000HR. It provides a complete description of the internal memory map of the chip and a detailed discussion of the internal registers to aid you in programming your CAN controller chip. The full functionality of the ECAN1000HR is described in the attached datasheet reprint from Philips on the SJA1000 CAN controller chip.

Defining the Memory Map

The memory map of the ECAN1000HR occupies 2 bytes of host PC I/O memory space. This 3-byte window is freely selectable by the user as described in *Chapter 2, Table 2.* After setting the base address you have access to the internal resources of the SJA1000 CAN controller chip. These resources are described in the next sections reprinted from the SJA1000 chip specific user's manual.

ADDRESS	Description
BASE+00h	ADDRESS
BASE+01h	DATA of ADDRESS
BASE+02h	HARDWARE RESET OF SJA1000

Table 5: ECAN Memory Map

The SJA1000 chip access is multiplexed in such a way that the host must first write to BASE+0 the internal address of the CAN chip and after that perform a write to address BASE+1 with the actual data to be written into the desired memory location. An example is listed below using "C" syntax. (We assume base address is 300H.)

Write 78H to the CAN controller Control byte located in the on-chip address 0.

```
outp(0x300,0x00);
outp(0x301,0x78);
```

Address BASE+02h is a hardware-reset function of the SJA1000. Performing a read or write to this address will cause a hardware reset to the CAN controller. You may need to reset the chip in case of an unrecoverable error in the CAN controller chip.

On the following pages is attached the chip specific user's manual for the SJA1000 CAN controller chip.

INTERRUPTS

What is an interrupt?

An interrupt is an event that causes the processor in your computer to temporarily halt its current process and execute another routine. Upon completion of the new routine, control is returned to the original routine at the point where its execution was interrupted.

Interrupts are a very flexible way of dealing with asynchronous events. Keyboard activity is a good example; your computer cannot predict when you might press a key and it would be a waste of processor time to do nothing whilst waiting for a keystroke to occur. Thus the interrupt scheme is used and the processor proceeds with other tasks. When a keystroke finally occurs, the keyboard then 'interrupts' the processor so that it can get the keyboard data .It then places it into the memory, and then returns to what it was doing before the interrupt occurred. Other common devices that use interrupts are A/D boards, network boards, serial ports etc.

Your ECAN1000HR can interrupt the main processor when a message is received or transmitted if interrupts are enabled on the ECAN1000HR board. By using interrupts you can write powerful code to interface to your CAN network.

Interrupt request lines

To allow different peripheral devices to generate interrupts on the same computer, the PC AT bus has interrupt request channels (IRQs). A rising edge transition on one of these lines will be latched into the interrupt controller. The interrupt controller checks to see if the interrupts are to be acknowledged from that IRQ and, if another interrupt is being processed, it decides if the new request should supersede the one in progress or if it has to wait until the one in progress has been completed. The priority level of the interrupt is determined by the number of the IRQ as follows; IRQ0 has the highest priority whilst IRQ15 has the lowest. Many of the IRQs are used by the standard system resources, IRQ0 is dedicated to the internal timer, IRQ1 is dedicated to the keyboard input, IRQ3 for the serial port COM2, and IRQ4 for the serial port COM1. Often interrupts 2,5 and 7 are free for the user.

8259 Programmable Interrupt Controller

The chip responsible for handling interrupt requests in a PC is the 8259 Interrupt Controller. To use interrupts you will need to know how to read and set the 8259's internal interrupt mask register (IMR) and how to send the end-of-interrupt (EOI) command to acknowledge the 8259 interrupt controller.

Interrupt Mask Register (IMR)

Each bit in the interrupt mask register (IMR) contains the mask status of the interrupt line. If a bit is set (equal to 1), then the corresponding IRQ is masked, and it will not generate an interrupt. If a bit is cleared (equal to 0), then the corresponding IRQ is not masked, and it can then generate an interrupt. The interrupt mask register is programmed through **port 21h**.

End-of-Interrupt (EOI) Command

After an interrupt service routine is complete, the 8259 Interrupt Controller must be acknowledged by writing the value 20h to port 20h.

What exactly happens when an interrupt occurs?

Understanding the sequence of events when an interrupt is triggered is necessary to correctly write interrupt handlers. When an interrupt request line is driven high by a peripheral device (such as the ECAN1000HR), the interrupt controller checks to see if interrupts are enabled for that IRQ. It then checks to see if other interrupts are active or requested and determines which interrupt has priority. The interrupt controller then interrupts the processor. The current code segment (CS), instruction pointer (IP), and flags are pushed onto the system stack, and a new set if CS and IP are loaded from the lowest 1024 bytes of memory.

This table is referred to as the interrupt vector table and each entry to this table is called an interrupt vector. Once the new CS and IP are loaded from the interrupt vector table, the processor starts to execute code from the new Code Segment (CS) and from the new Instruction Pointer (IP). When the interrupt routine is completed the old CS and IP is popped from the system stack and the program execution continues from the point where the interruption occurred.

Using Interrupts in your program

Adding interrupt support to your program is not as difficult as it may seem especially when programming under DOS. The following discussion will cover programming under DOS. Note that even the smallest mistake in your interrupt program may cause the computer to hang up and will only restart after a reboot. This can be frustrating and time-consuming.

Writing an Interrupt Service Routine (ISR)

The first step in adding interrupts to your software is to write an interrupt service routine (ISR). This is the routine that will be executed automatically each time an interrupt request occurs for the specified IRQ. An ISR is different from other sub-routines or procedures. First, on entrance the processor registers must be pushed onto the stack_before anything else! Second, just before exiting the routine, you must clear the interrupt on the ECAN1000HR by writing to the SJA1000 CAN controller, and write the EOI command to the interrupt controller. Finally, when exiting the interrupt routine the processor registers must be popped from the system stack and you must execute the IRET assembly instruction. This instruction pops the CS, IP and processor flags from the system stack. These were pushed onto the stack when entering the ISR.

Most compilers allow you to identify a function as an interrupt type and will automatically add these instructions to your ISR with one exception: most compilers do not automatically add the EOI command to the function, you must do it yourself. Other than this and a few exceptions discussed below, you can write your ISR as any code routine. It can call other functions and procedures in your program and it can access global data. If you are writing your first ISR, we recommend you stick to the basics; just something that enables you to verify you have entered the ISR and executed it successfully. For example: set a flag in your ISR and in your main program check for the flag.

Note: If you choose to write your ISR in in-line Assembly, you must push and pop registers correctly and exit the routine with the IRET instruction instead of the RET instruction.

There are a few precautions you must consider when writing ISRs. The most important is, **do not use any DOS functions or functions that call DOS functions from an interrupt routine.** DOS is not re-entrant; that is, a DOS function cannot call itself. In typical programming, this will not happen because of the way DOS is written. But what about using interrupts? Consider then the following situation in your program: If DOS function X is being executed when an interrupt occurs and the interrupt routine makes a call to the same DOS function X, then function X is essentially being called while active. Such cases will cause the computer to crash. DOS does not support such operations. The general rule is that do not call any functions that use the screen, read keyboard input or any file I/O routines. These should not be used in ISRs.

The same problem of reentrancy also exists for many floating-point emulators. This effectively means that you should also avoid floating point mathematical operations in your ISR.

Note that the problem of reentrancy exists, no matter what programming language you use. Even, if you are writing your ISR in Assembly language, DOS and many floating point emulators are not re-entrant. Of course there are ways to avoid this problem, such as those which activate when your ISR is called. Such solutions are, however, beyond the scope of this manual.

The second major concern when writing ISRs is to make them as short as possible in term of execution time. Spending long times in interrupt service routines may mean that other important interrupts are not serviced. Also, if you spend too long in your ISR, it may be called again before you have exited. This will lead to your computer hanging up and will require a reboot.

Your ISR should have the following structure:

- Push any processor registers used in your ISR. Most C compiler do this automatically
- Put the body of your routine here
- Read interrupt status register of the SJA1000 chip on your ECAN1000HR board
- Clear the interrupt bit by writing to the SJA1000 CAN controller
- Issue the EOI command to the 8259 by writing 20h to address 20h
- Pop all registers. Most C compilers do this automatically

The following C example shows what the shell of your ISR should be like:

```
/*-----------------------------------------------------------------------------
| Function:     new_IRQ_handler
| Inputs:               Nothing
| Returns:      Nothing         - Sets the interrupt flag for the EVENT.
|-----------------------------------------------------------------------------*/
void interrupt far new_IRQ_handler(void)
{

        IRQ_flag = 1;           // Indicate to main process interrupt has oc-
curred
        {
        // Your program code should be here
        }
                                // Read interrupt status registers
                                // Clear interrupt on ECAN1000HR
        outp(0x20, 0x20);       /* Acknowledge the interrupt controller. */
}
```

Saving the Startup Interrupt Mask Register (IMR) and interrupt vector

The next step after writing the ISR is to save the “start up”-state of the interrupt mask register and the original interrupt vector you are using. The IMR is located in address **21h**. The interrupt vector you will be using is located in the interrupt vector table which is an array of pointers (addresses) and it is locate din the first 1024 bytes of the memory (Segment 0 offset 0). You can read this value directly, but it is better practice to use DOS function 35h (get interrupt vector) to do this. Most C compilers have a special function available for doing this. The vectors for the hardware interrupts on the XT-bus are vectors 8-15, where IRQ0 uses vector 8 and IRQ7 uses vector 15. Thus if your ECAN1000HR is using IRQ5 it corresponds to vector number 13.

Before you install your ISR, temporarily mask out the IRQ you will be using. This prevents the IRQ from requesting an interrupt while you are installing and initializing your ISR. To mask the IRQ, read the current IMR at I/O port 21h, and set the bit that corresponds to the IRQ. The IMR is arranged so that bit 0 is for IRQ0 and bit 7 is for IRQ7. See the paragraph entitled *Interrupt Mask Register (IMR*) earlier in this discussion for help in determining your IRQ's bit. After setting the bit, write the new value to I/O port 21h.

With the startup IMR saved and the interrupts temporarily disabled, you can assign the interrupt vector to point to your ISR. Again you can overwrite the appropriate entry in the vector table with a direct memory write, but this is not recommended. Instead use the DOS function 25h (Set Interrupt Vector) or, if your compiler provides it, the library routine for setting up interrupt vectors. Remember that interrupt vector 8 corresponds to IRQ0, vector 9 for IRQ1 etc.

If you need to program the source of your interrupts, do that next. For example, if you are using transmitted or received messages as interrupt sources, program it to do that. Finally, clear the mask bit for your IRQ in the IMR. This will enable your IRQ.

Common Interrupt mistakes

Remember that hardware interrupts are from 8-15, XT IRQs are numbered 0-7. Forgetting to clear the IRQ mask bit in the IMR Forgetting to send the EOI command after ISR code disables further interrupts.

Example on Interrupt vector table setup in C-code:

```
void far _interrupt new_IRQ1_handler(void );          /* ISR function proto-
type */
#define IRQ1_VECTOR        3                          /* Name for IRQ */
void (interrupt far *old_IRQ1_dispatcher)
        (es,ds,di,si,bp,sp,bx,dx,cx,ax,ip,cs,flags);  /* Variable to store
old IRQ_Vector */
void far _interrupt new_IRQ1_handler(void );

/*-----------------------------------------------------------------
| Function:    init_irq_handlers
| Inputs:              Nothing
| Returns:     Nothing
| Purpose:     Set the pointers in the interrupt table to point to
|              our funtions ie. setup for ISR's.
|-----------------------------------------------------------------*/
void init_irq_handlers(void)
{
        _disable();
                old_IRQ1_handler = _dos_getvect(IRQ1_VECTOR + 8);
                _dos_setvect(IRQ1_VECTOR + 8, new_IRQ1_handler);
                Gi_old_mask = inp(0x21);
                outp(0x21,Gi_old_mask & ~(1 << IRQ1_VECTOR));
        _enable();
}

|/*-----------------------------------------------------------------
| Function:    restore do this before exiting program
| Inputs:              Nothing
| Returns:     Nothing
| Purpose:     Restore interrupt vector table.
|-----------------------------------------------------------------*/
void restore(void)
{
        /* Restore the old vectors */

        _disable();

                _dos_setvect(IRQ1_VECTOR + 8, old_IRQ1_handler);
                outp(0x21,Gi_old_mask);

        _enable();
}
```

Chapter 6 ECAN1000HR SPECIFICATIONS

Host Interface

I/O mapped device, occupies 3 bytes
Jumper-selectable base address
8-bit data bus, 16-bit AT bus connector
Jumper selectable XT and AT interrupts

CAN Interfaces

Galvanically isolated transceiver 1 Mb/s data rate
Timing parameters and speed of bus programmable
Balanced CAN bus Choke
Jumper selectable 120 Ohm termination resistors
Transceiver type: Philips 82C251
0.8W 5V isolated output power for other field devices

Connectors

Galvanically isolated CAN bus	90 degree 10-pin header
Host bus	XT/AT PC/104 bus

Electrical

Operating temperature range	-40 to +85 C
Supply voltage	+5V, +-8%
Power consumption	1,0W typical

CE

The ECAN1000HR is CE certified in the IDAN Enclosure System.
Please consult the factory for more information on the system.

Chapter 7 RETURN POLICY AND WARRANTY

Return Policy

If you wish to return a product to the factory for service, please follow this procedure:

Read the Limited Warranty to familiarize yourself with our warranty policy.

Contact the factory for a Return Merchandise Authorization (RMA) number.

Please have the following available:

- Complete board name
- Board serial number
- A detailed description of the board's behavior

List the name of a contact person, familiar with technical details of the problem or situation, **along with their phone and fax numbers, address, and e-mail address** (if available).

List your shipping address!!

Indicate the shipping method you would like used to return the product to you.
We will not ship by next-day service without your pre-approval.

Carefully package the product, using proper anti-static packaging.

Write the RMA number in large (1") letters on the outside of the package.

Return the package to:

RTD Embedded Technologies, Inc.

103 Innovation Blvd.
State College PA 16803-0906

USA

Limited Warranty

RTD Embedded Technologies, Inc. warrants the hardware and software products it manufactures and produces to be free from defects in materials and workmanship for one year following the date of shipment from RTD Embedded Technologies, INC. This warranty is limited to the original purchaser of product and is not transferable.

During the one year warranty period, RTD Embedded Technologies will repair or replace, at its option, any defective products or parts at no additional charge, provided that the product is returned, shipping prepaid, to RTD Embedded Technologies. All replaced parts and products become the property of RTD Embedded Technologies. Before returning any product for repair, customers are required to contact the factory for an RMA number.

THIS LIMITED WARRANTY DOES NOT EXTEND TO ANY PRODUCTS WHICH HAVE BEEN DAMAGED AS A RESULT OF ACCIDENT, MISUSE, ABUSE (such as: use of incorrect input voltages, improper or insufficient ventilation, failure to follow the operating instructions that are provided by RTD Embedded Technologies, "acts of God" or other contingencies beyond the control of RTD Embedded Technologies), OR AS A RESULT OF SERVICE OR MODIFICATION BY ANYONE OTHER THAN RTD Embedded Technologies. EXCEPT AS EXPRESSLY SET FORTH ABOVE, NO OTHER WARRANTIES ARE EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND RTD Embedded Technologies EXPRESSLY DISCLAIMS ALL WARRANTIES NOT STATED HEREIN. ALL IMPLIED WARRANTIES, INCLUDING IMPLIED WARRANTIES FOR MECHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED TO THE DURATION OF THIS WARRANTY. IN THE EVENT THE PRODUCT IS NOT FREE FROM DEFECTS AS WARRANTED ABOVE, THE PURCHASER'S SOLE REMEDY SHALL BE REPAIR OR REPLACEMENT AS PROVIDED ABOVE. UNDER NO CIRCUMSTANCES WILL RTD Embedded Technologies BE LIABLE TO THE PURCHASER OR ANY USER FOR ANY DAMAGES, INCLUDING ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES, EXPENSES, LOST PROFITS, LOST SAVINGS, OR OTHER DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PRODUCT.

SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES FOR CONSUMER PRODUCTS, AND SOME STATES DO NOT ALLOW LIMITATIONS ON HOW LONG AN IMPLIED WARRANTY LASTS, SO THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.