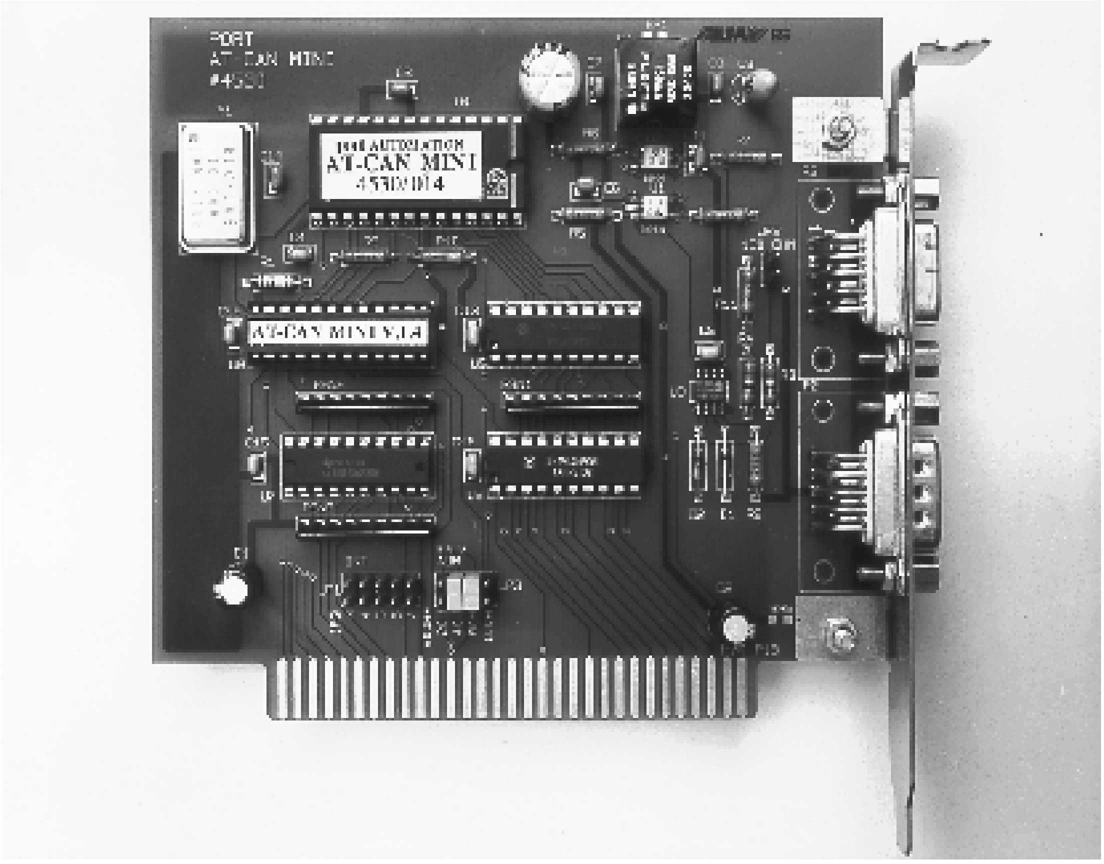
User Manual
AT-CAN-MINI



All rights reserved

The contents of this documentation were generated and checked carefully by engineers of *port* GmbH (in following *port*). *port* however accepts no responsibility for damage caused by the use of this documentation. In particular performance characteristics and technical data given in this document may not be constituted to be guaranteed product features in any legal sense.

port has the right to modify the products described or their documentation at any time without prior warning, as long as these changes are made for reasons of reliability or technical improvement.

All rights of this documentation lie with *port*. The transfer of rights to third parties or duplication of this document in any form, whole or in part, is subject to written approval by *port*. Copies of this document may however be made exclusively for the use of the user and his engineers. The user is thereby responsible that third parties do not obtain access to these copies.

The soft- and hardware designations used are mostly registered and are subject to copyright.

1. Technical Data

AT-CAN-MINI provides a CAN network interface for PC-AT compatible computers. It is designed as a short PC slot card with dimensions 100mm x 150mm. PC interface is a ISA direct connector. The CAN network connection is made by an SUB-D-9male and SUB-D-9female connector physically it is designed according to ISO/DIS 11898 and the recommendations of CiA (CAN in Automation e. V.). After plug in the card into the PC slot the CAN connectors are available at the PC's backboard.

The CAN interface is optically isolated against the PC voltage level.

AT-CAN-MINI occupies 32 addresses in the PC's I/O address space. The starting address is selectable by jumpers.

The CAN controller can generate interrupts at the PC bus. The interrupt number is also selectable by jumpers.

As CAN controller the BASIC CAN SJA1000 by Philips is used.

Port GmbH provides C-library functions for simple functions to program the controller. The library is foreseen and tested with the following compilers:
BORLAND C++ 2.0 and
BORLAND TURBO C 2.01

PC-Interface	ISA-direct board connector
CAN-Interface	2 x SUB-D-9 according ISO/DIS 11898 and CiA recommendation
CAN-Controller	SJA1000
CAN-driver	SJA1000 or SI9200
address area	32 byte between 200h - 3E0h
Interrupts	IRQ3 - IRQ7 selected by jumper
Dimension	100mm x 150mm

Table 1, technical data overview

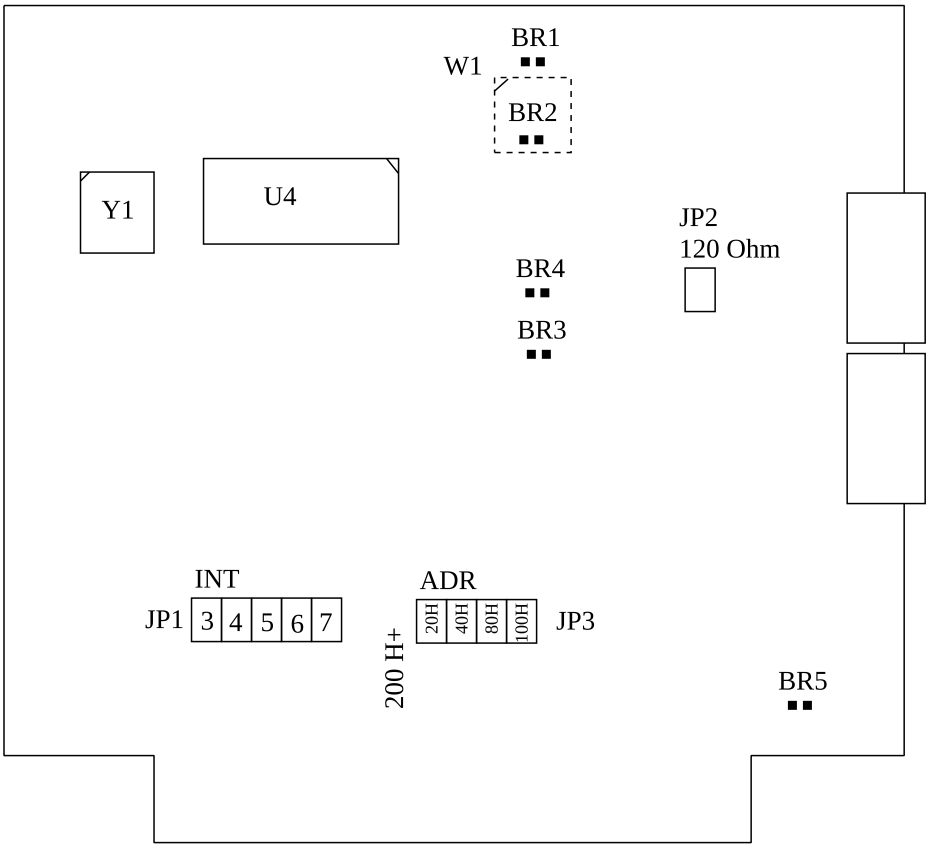


Figure 1, jumpers and soldering bridges

2. Hardware

Following is an overview schematic of the board.

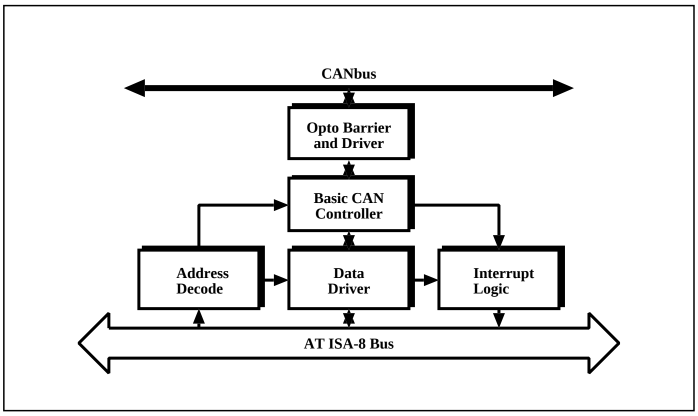


Figure 2, AT-CAN-MINI schematic

2.1. Addressing

AT-CAN-MINI uses an 32 byte address window in the I/O address space of the PC. In this area the controllers register are mapped.

Begin address of this I/O area is located between 200h and 3E0h. With jumper JP3 it can be freely assigned in steps of 32 byte, 20h.

1-2	3-4	5-6	7-8	Adresse
I	I	I	I	200h
	I	I	I	220h
I		I	I	240h
		I	I	260h
I	I		I	280h
	I		I	2A0h
I			I	2C0h
			I	2E0h
I	I	I		300h
	I	I		320h
I		I		340h
		I		360h
I	I			380h
	I			3A0h
I				3C0h
				3E0h

Table 2, I/O address jumpering with JP3

Legend:
I ... jumper applied

2.2. CAN-Controller

As CAN controller the Philips Basic CAN SJA1000 clocked with 16 MHz is used.

The clock is generated at board.

Together with the CAN driver circuit SJA1000 bit rates up to 1 MBaud are possible.

The controller is used in the **Intel**-Mode. Signals are driven from a PAL (22V10).

By using a CAN controller which handles all layer 2 protocol the user only has to provide data in the transmit buffer. The controller then transmits the message automatically. Also receiving is done automatically. Received messages must only be read from the registers.

2.3. Address-/data multiplexor

The used CAN controller SJA1000 has an shared data/address bus. To use the 32 registers of the SJA1000 address lines A0...A4 of the PC are used. These lines must be multiplexed with data lines D0..D7.

Address/data multiplexor is built by the circuit U6 (unidirection address) and U5 (bidirectional data).

Control signals "DEN*" and "AEN*" as also the correct timing are created by the PAL U8.

2.4. Connection to the Bus

The physical connection to the CAN bus is realized by two parallel connected SUB-D-9 connectors according to DIN 41652.

Pin	Signal	Description
1	-	reserved
2	CAN_L	CAN_L (dominant low)
3	GND	ground
4	-	reserved
5	-	reserved
6	(GND)	ground (optional)
7	CAN_H	CAN_H (dominant high)
8	-	reserved (error line)
9	(V+)	external power supply 7V - 13V (optional)

Table 3, pin assignment for the SUB-D-9 CAN

The two connectors make it possible to grind the bus through the AT-CAN-MINI.

2.5. Optical isolation

Optical isolation is provided by the coupler U2 for transmitting and U1 for receiving.

Galvanical isolation between the 5 V Power Supply and the CAN driver circuit with opto-coupler U2 is done with the DC/DC converter W1.

If this is not necessary following changes has to be made:
- components: U1, U2 and W1 do not mount;
- soldering bridges: BR1, BR2, BR3 and BR4 close;

If the board is already assigned with U1, U2 and W1, then the soldering bridges BR2, BR3 and BR4 are not accessible in order to protect the components.

If a galvanic connection is desired and the components already mounted, the soldering bridge BR1 must be closed.

The slot plate with mounted DSUB connectors is normally connected with ground of the PC by bridge BR5. Within industrial PC’s with no connection between housing and processor ground the bridge must be disrupted.

2.6. CAN driver

As a CAN driver the PCA82C250T or SI9200 can electively be used. Both are pin compatible. Driver and opto coupler build a CAN interface according to ISO/DIS 11898.

If the SI9200 is used, the resistor R2, reducing the steepness of signal edges with PCA82C250T, is not necessary.
Steepness is proportional to the used resistor.

Following relations are valid:

value R2	result
0 (PIN 8 at GND)	highest steepness
open	standby modus
10kOhm .. 100kOhm	slope control

Table 4, influence of steepness

Formula: calculation of edge steepness

$$t_{slope} = -3 * 10^{-5} + 36 \quad [V/\mu s]$$

Standard resistor with 10 KOhm yields in a steepness of 30V/μs.

2.7. Interrupt

To achieve the capability to generate interrupts from the CAN channel **one** of the interrupt lines must selected with the "INT" jumper JP1.

It can be selected from the lines IRQ3 to IRQ7. IRQ5 is preferred.

1-2	3-4	5-6	7-8	9-10	IRQ
I					IRQ3
	I				IRQ4
		I			IRQ5
			I		IRQ6
				I	IRQ7

Table 5, interrupt jumpering with JP1

I..jumper applied

Attention !

Only **one** jumper must be applied.

3. Software

3.1. Introduction

The CAN controller with the AT-CAN-MINI is driven in I/O mode, That means that communicating with the board is done by 32 addresses in the PC's I/O address range.

Port GmbH provides with the C-library *can_util.lib* some functions to easy access the CAN controller. Following functions are delivered:

Function	Short description
CAN_PortInit()	initialization of the CAN controller
CAN_PortMask()	programming acceptance and mask registers
CAN_PortStatus()	read controller status
CAN_PortTxRdy()	check transmit readiness
CAN_PortRxRdy()	check receive readiness
CAN_PortTx()	send message
CAN_PortRx()	read message
CAN_PortIRQ_E()	enable interrupts
CAN_PortIRQ_D()	disable interrupts
CAN_PortIRQRead()	read interrupt register

Module number

To be able to use the library also with boards with more then one CAN controller, a module number is provided with all calls to functions. Because AT-CAN-MINI has only one controller

Module number is always 1.

Following conventions are used:

- TRUE = 1
- FALSE = 0

The provided examples only should show how to use the library functions. Therefore no sanity checks are within it.

3.1.1. Data types and structures

Some of the functions get or deliver data types **BYTE**. It is declared as

```
typedef unsigned char BYTE;
```

and allocates one byte in memory.

telegramm is a structure, containing all necessary informations for a CAN message:

message — is the CAN identifier from or to other CAN nodes
11 Bit in a bit field

rtr — identifier for an Remote Transmit Request (RTR)
1 Bit in a bit field

count — number of bytes to send or received
4 Bit in a bit field

daten — With CAN up to 8 bytes can be transfered. The message data are stored in an 8 byte array of unsigned char.

3.2. Content of disk

With the provided disk are the following files:

File	Description
can_util.h	Header file for CAN functions
can_util.lib	library file
canrec.c	example 1
cansend.c	example 2
readme	notes for using examples

Table 6, content of supplied disk

4. Appendix

4.1. Example canrec.c

The example waits for an incoming CAN message and displays the contents of the message. It runs until the user hits one key. More notes to the example are in the file *readme*.

```
#include <stdio.h>
#include <dos.h>
#include <bios.h>
#include "can_util.h"

#define OFF  0
#define ON   1
#define MOD1 1                          /* Modul Nr. 1 */
#define MOD2 2                          /* Modul Nr. 2 */

BYTE bus0 =     0x89;                   /* Bustiming Register 0 */
BYTE bus1 =     0xEB;                   /* Bustiming Register 1 */
BYTE mask =     0xFF;                   /* Maskenregister */
BYTE acc =       0xFF;                  /* Akzeptanzregister */
int port_adr = 0x280;                   /* Adresse I/O-Port */
int modul =     MOD1;                   /* Modulnummer */

main()
{
can_telegramm rx;                       /* Empfangsstruktur */
int z;                                  /* Durchlaufzähler */
int i;                                  /* Zählvariable */
int schalter = OFF;

/*--- CAN-Controller initialisieren ----------------------------------*/

    CAN_PortInit (port_adr, MOD1, bus0, bus1);
    CAN_PortMask (port_adr, MOD1, mask, acc);

/*--- periodisch empfangen ----------------------------------------*/

    while (schalter == OFF){
        while (!CAN_PortRx(port_adr, modul, &rx) && schalter == OFF){
            delay(1);
            z++;
            if (z > 5000){
                printf("warte0);
                z = 0;
            }
            if (bioskey(1))
                schalter = ON;
        }
        if (schalter == OFF){
            printf("Message: %d0,rx.message);
            printf("Count: %d0,rx.count);
            for (i = 0; i < rx.count; i++){
                printf("Daten hex: %x0,rx.daten[i]);
                printf("Daten char: %c0,rx.daten[i]);
                printf("Daten dezimal: %d0,rx.daten[i]);
            }
        }
    }
}
```

4.2. Example cansend.c

The programm periodically sends a message. Message data are entered over the command line. It runs until the user hits one key. More notes to the example are in the file *readme*.

```
#include <stdio.h>
#include <dos.h>
#include <bios.h>
#include <stdlib.h>
#include <string.h>
#include "can_util.h"

#define OFF     0
#define ON      1
#define MOD1    1               /* Modul Nr. 1 */
#define MOD2    2               /* Modul Nr. 2 */

BYTE bus0 = 0x89;               /* Bustiming Register 0 */
BYTE bus1 = 0xEB;               /* Bustiming Register 1 */
BYTE mask = 0xFF;               /* Maskregister */
BYTE acc = 0xFF;                /* Acceptanzregister */

int port_adr = 0x280;           /* Adresse I/O-Port */
int modul = MOD1;               /* Modulnummer */

main (int argc, char *argv[])
{
can_telegramm tx;
int z;
int schalter = OFF;

/*--- Sendestruktur beschreiben -------------------------------------*/

    tx.message = atoi(argv[1]);
    tx.count = atoi(argv[2]);
    strcpy(tx.daten, argv[3]);

/*--- CAN-Controller initialisieren -----------------------------------*/

    CAN_PortInit(port_adr, modul, bus0, bus1);
    CAN_PortMask(port_adr, modul, mask, acc);

/*--- periodisch senden ------------------------------------------------*/

    while (schalter == OFF){
        while (!CAN_PortTx(port_adr, modul, &tx) && (schalter == OFF)){
            delay(1);
            z++;
            if ( z > 5000){
                printf("warte0);
                z = 0;
            }
            if (bioskey(1))
                schalter = ON;
        }
    }
}
```

Table of Contents