

# PCM-3680 PC/104 Dual Port CAN Interface Module



## Jumper Setting

The PCM-3680 is a special purpose communication card that brings the Control Area Network to your PC. With the built-in CAN controller, the PCM-3680 provides bus arbitration and error detection with automatic transmission repeat function. This drastically avoids data loss and ensures system reliability. The on-board CAN controllers are located at different positions in the memory. You can run both CAN controllers at the same time, independently. The PCM-3680 operates at baud rates up to 1 Mbps and can be installed directly into the expansion slot of your PC.

### Control Area Network

The CAN (Control Area Network) is a serial bus system especially suited for networking "intelligent" I/O devices as well as sensors and actuators within a machine or plant. Characterized by its multi-master protocol, real-time capability, error correction, high noise immunity, and the existence of many different silicon components, the CAN serial bus system, originally developed by Bosch for use in automobiles, is increasingly being used in industrial automation.

### Direct Memory Mapping

The PCM-3680 is assigned with memory address, which allows direct access to the CAN controller. This is the simplest and fastest way of programming any board in a PC because the board is regarded as standard RAM.

### Optical Isolation Protection

On-board optical isolators protect your PC and equipment against damage from ground loops, increasing system reliability in harsh environments.

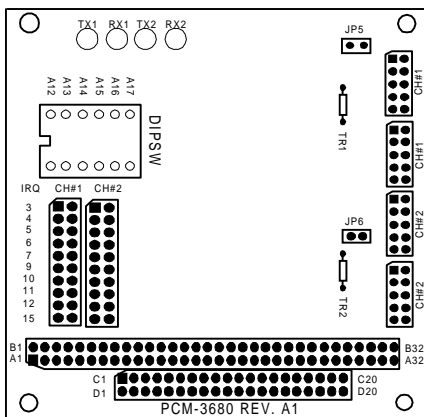
## Specifications

- **Ports:** 2
- **CAN controller:** SJA-1000
- **CAN transceiver:** 82C250
- **Signal support:** CAN-L, CAN-H
- **Memory address:** From C800H to EF00H
- **IRQ:** 3, 4, 5, 6, 7, 9, 10, 11, 12, 15
- **Isolation voltage:** 1000 V<sub>DC</sub>
- **Power consumption:** +5 V @ 400 mA typical, 950 mA max.
- **Connectors:** Dual DB-9 male connectors
- **Operating temperature:** 32 to 122° F (0 to 50° C)
- **PC/104 form factor:** 3.6" x 3.8" (90 mm x 96 mm)
- **Shipping weight:** 0.9 lb (0.4 kg)

## Features

- Operates 2 separate CAN networks at the same time
- High speed transmission up to 1 Mbps
- 16 MHz CAN controller frequency
- Takes a 4 KB address space, 40 base address adjustable in steps from C800H up to EF00H
- Optical isolation protection of 1000 V<sub>DC</sub> ensures system reliability
- Wide IRQ selection for each port includes: IRQ 3, 4, 5, 6, 7, 9, 10, 11, 12, 15
- LED indicates Transmit/Receive status on each port
- Direct memory mapping enables speedy access to the CAN controllers
- C library and examples included

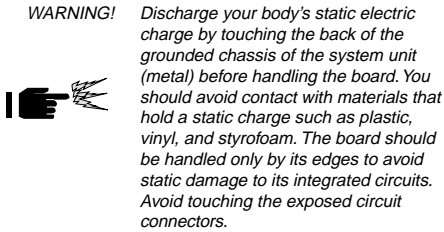
## Jumper & Switch Locations



## Initial Inspection

In addition to this user's manual, your shipping box should contain the following items:

- PCM-3680 Dual-port CAN Interface Card
- C Driver and DataMonitor Utility Diskette



## Installation

### Card Configuration

The PCM-3680 has two ports, each with one jumper. The jumpers set the IRQ for the ports, which can be configured separately. A DIP switch sets the memory base address for each port. The following chart shows the function of the jumper and the switch (see the previous page for jumper and switch locations).

### Switch and jumper functions

#### IRQ Setup

|     |        |
|-----|--------|
| JP1 | Port 1 |
| JP2 | Port 2 |

#### Memory base address

|     |                |
|-----|----------------|
| SW1 | Port 1, Port 2 |
|-----|----------------|

#### Default Settings

- Port 1 is set for COM1 (IRQ=12, Memory address = DA00:0000).
- Port 2 is set for COM2 (IRQ=15, Memory address = DA00:0200).

If you need to change these settings, see the following sections. Otherwise, you can simply install the card. Note that you will need to disable your CPU card's on-board COM ports, if any, or set them to alternate addresses/IRQs.

### Jumpers and Switches

Jumpers JP1 and JP2 set the interrupts for Port 1 and Port 2, respectively. You can choose any IRQ from 3 to 15, except 8, 13 and 14. When you choose IRQs, make sure they are not used for other cards in the system. The following figures show the card's default settings.

#### JP1: Port 1 IRQ Default

| IRQ | Ch. 1                            |
|-----|----------------------------------|
| 3   | <input type="radio"/>            |
| 4   | <input type="radio"/>            |
| 5   | <input type="radio"/>            |
| 6   | <input type="radio"/>            |
| 7   | <input type="radio"/>            |
| 9   | <input type="radio"/>            |
| 10  | <input type="radio"/>            |
| 11  | <input type="radio"/>            |
| 12  | <input checked="" type="radio"/> |
| 15  | <input type="radio"/>            |

#### JP2: Port 2 IRQ Default

| IRQ | Ch. 2                            |
|-----|----------------------------------|
| 3   | <input type="radio"/>            |
| 4   | <input type="radio"/>            |
| 5   | <input type="radio"/>            |
| 6   | <input type="radio"/>            |
| 7   | <input type="radio"/>            |
| 9   | <input type="radio"/>            |
| 10  | <input type="radio"/>            |
| 11  | <input type="radio"/>            |
| 12  | <input type="radio"/>            |
| 15  | <input checked="" type="radio"/> |

### Memory Base Address (SW1)

The memory base address for the PCM-3680, which requires 4 KB of address space, is made up of the memory segment and its associated offset. The address for the memory segment is set through SW1, a six-position DIP switch. You can choose any base address from C800 to EF00. The following table shows the DIP switch settings and the corresponding base addresses.

#### Memory address configuration (SW1)

| Address/DIP switch | A12 | A13 | A14 | A15 | A16 | A17 |
|--------------------|-----|-----|-----|-----|-----|-----|
| C800H              | on  | on  | on  | off | on  | on  |
| C900H              | off | on  | on  | off | on  | on  |
| CA00H              | on  | off | on  | off | on  | on  |
| CB00H              | off | off | on  | off | on  | on  |
| CC00H              | on  | on  | off | off | on  | on  |
| CD00H              | off | on  | off | off | on  | on  |
| CE00H              | on  | off | off | off | on  | on  |
| CF00H              | off | off | off | off | on  | on  |
| D000H              | on  | on  | on  | on  | off | on  |
| D100H              | off | on  | on  | on  | off | on  |
| D200H              | on  | off | on  | on  | off | on  |
| D300H              | off | off | on  | on  | off | on  |
| D400H              | on  | on  | off | on  | off | on  |
| D500H              | off | on  | off | on  | off | on  |
| D600H              | on  | off | off | on  | off | on  |
| D700H              | off | off | off | on  | off | on  |
| D800H              | on  | on  | on  | off | off | on  |
| D900H              | off | on  | on  | off | off | on  |
| DA00H              | on  | off | on  | off | off | on  |
| DB00H              | off | off | on  | off | off | on  |
| DC00H              | on  | on  | off | off | off | on  |
| DD00H              | off | on  | off | off | off | on  |
| DE00H              | on  | off | off | off | off | on  |
| DF00H              | off | off | off | off | off | on  |
| E000H              | on  | on  | on  | on  | on  | off |
| E100H              | off | on  | on  | on  | on  | off |
| E200H              | on  | off | on  | on  | on  | off |
| E300H              | off | off | on  | on  | on  | off |
| E400H              | on  | on  | off | on  | on  | off |
| E500H              | off | on  | off | on  | on  | off |
| E600H              | on  | off | off | on  | on  | off |
| E700H              | off | off | off | on  | on  | off |
| E800H              | on  | on  | on  | off | on  | off |
| E900H              | off | on  | on  | off | on  | off |
| EA00H              | on  | off | on  | off | on  | off |
| EB00H              | off | off | on  | off | on  | off |
| EC00H              | on  | on  | off | off | on  | off |
| ED00H              | off | on  | off | off | on  | off |
| EE00H              | on  | off | off | off | on  | off |
| EF00H              | off | off | off | off | on  | off |

## Memory Area

Once the memory segment for the base address is selected, the offset will be automatically assigned for Port 1, Port 2, and hardware reset. The following table shows the base addresses of the CAN controllers.

| Base address (hex)      | CAN controller          |
|-------------------------|-------------------------|
| base:0000h - base:00FFh | Basic- Port 1           |
| base:0100h - base:01FFh | HW reset Basic - Port 1 |
| base:0200h - base:02FFh | Basic- Port 2           |
| base:0300h - base:03FFh | HW reset Basic - Port 2 |
| base:0400h - base:0FFFh | Not used                |

## Software Programming

### Quick Reference Table

The following table lists the available functions and their corresponding syntax and descriptions.

#### Library functions

| Function | Syntax (in C)   | Description                |
|----------|-----------------|----------------------------|
| 1        | canInitHW()     | Sets IRQs                  |
| 2        | canExitHW()     | Releases settings          |
| 3        | canReset()      | Resets CAN port            |
| 4        | canConfig()     | Controls CAN port settings |
| 5        | canNormalRun()  | Sets mode                  |
| 6        | canSendMsg()    | Sends message              |
| 7        | canReceiveMsg() | Reads data                 |

### Complete Function Description

#### Function 1

Sets an IRQ number for Port1 and Port 2.

- **Command**     canInitHW (UI segment, BYTE IRQ1, BYTE IRQ2)
- **Argument**     UI segment, BYTE IRQ1, BYTE IRQ2  
segment=c000-df00 step 0x100  
IRQ1=Port 1 IRQ number 0 (polling),  
3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15  
IRQ2=Port 2 IRQ number 0 (polling),  
4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15  
0: polling
- **Response**     1=successful  
0=fail
- **Example**  
#include "can841.h"  
main()  
{  
    UI gSegment=0xDA00;  
    BYTE CAN1\_IRQ, CAN2\_IRQ;  
    CAN1\_IRQ=12;  
    CAN2\_IRQ=15;  
    if (canInitHW (gSegment, CAN1\_IRQ, CAN2\_IRQ)==0)  
        printf ("HARDWARE INITIALIZATION ERROR!\n");  
}

#### Function 2

Releases all settings of the CAN card.

- **Command**     canExitHW()
- **Argument**     None
- **Response**     1=successful  
0=fail
- **Example**  
#include "can841.h"  
main()  
{  
    if (canExitHW()==0)  
        printf ("CAN RELEASE FAIL!\n");  
}

#### Function 3

Resets CAN port and flushes the TX/RX buffers.

- **Command**     int canReset (BYTE port);
- **Argument**     BYTE port; port= port number (0 or 1)
- **Response**     1=successful  
0=fail
- **Example**  
#include "can841.h"  
main()  
{  
    if (canReset (0)==0)  
        printf ("RESET PORT 1 FAIL!\n");  
}

#### Function 4

Controls the setting of the CAN port's acceptance code, acceptance mask, and bus timing register.

- **Command**     canConfig (BYTE port, CAN\_STRUCT can);
- **Argument**     BYTE port, CAN\_STRUCT can;  
port= port number (0 or 1)  
can= CAN struct pointer
- **Response**     1=successful  
0=fail
- **Example**  
#include "can841.h"  
main()  
{  
    CAN\_STRUCT can1, can2;  
    can1.acc\_code=0;  
    can1.acc\_mask=0xff;  
    can1.bt0=0;  
    can1.bt1=0x1c;  
    if (canConfig(0,can1)==0)  
        printf ("CAN PORT 1 CONFIGURE ERROR!\n");  
}

### Function 5

Sets a CAN port to normal mode for normal operation.

**Command** canNormalRun (BYTE port);  
**Argument** BYTE port;port= port number (0 or 1)  
**Response** 1=successful  
0=fail

#### Example

```
#include "can841.h"
main()
{
    if (canNormalRun(0)==0)
        printf ("CAN Port 1 can't change to Normal Mode!\n");
}
```

### Function 6

Tells the CAN port to send a message.

**Command** canSendMsg (BYTE port, MSG\_STRUCT send\_msg);  
**Argument** BYTE port, MSG\_STRUCT send\_msg; port= port number (0 or 1) send\_msg= send buffer pointer  
**Response** 1=successful  
0=fail

#### Example

```
#include "can841.h"
main()
{
    MSG_STRUCT smsg1;
    UI i;
    smsg1.id=0x015;
    smsg1.rtr=0;
    smsg1.dlen=8;
    for(i=0; i<smsg1.dlen; i++)
        smsg1.data[i]=i;
    if (canSendMsg(0,smsg1)==1)
        printf ("TRANSMISSION SUCCESSFUL!\n");
}
```

### Function 7

Read data from CAN port input buffer.

**Command** int canReceiveMsg (BYTE port, MSG\_STRUCT \*msg\_ptr);  
**Argument** BYTE port, MSG\_STRUCT \*msg\_ptr; port= port number (0 or 1) \*msg\_ptr= input buffer pointer  
**Response** 1=message received  
0=no message received

#### Example

```
#include "can841.h"
main()
{
    MSG_STRUCT rmsg2;
    if (canReceiveMSG, &rmsg2)==1)
    {
        printf ("Port2 receive: ID=%3X RTR=%ld Length=%ld", rmsg2.id, rmsg2.rtr, rmsg2.dlen);
        for (i=0; i<rmsg2.dlen; i++)
            printf (" %2X", rmsg2.data[i]);
    }
}
```

## Example Program

The following example program, can841.lib, implements the sending and receiving of messages over the CAN controller. The program is written in C.

```
#include "can841.h" /*Library function declaration*/
/*-----*/

/* CAN controller interrupt connection */
#define CAN1_IRQ 12 /* 0 means polling */
#define CAN2_IRQ 15 /* 0 means polling */
#define PORT1 0
#define PORT2 1
#define FAIL 0
#define SUCCESS 1

void main(void)
{
    /* Declare the CAN card segment address. */
    UI gSegment=0xDA00;
    CAN_STRUCT can1, can2;
    MSG_STRUCT smsg1, smsg2;
    MSG_STRUCT rmsg1, rmsg2;
    UI i;

    if(canInitHW(gSegment,CAN1_IRQ,CAN2_IRQ)==FAIL)
    {
        clrscr();
        printf("\n\n Hardware Initializa
tion Error");
        return;
    }

    /* Reset CAN controller */
    canReset(PORT1);
    canReset(PORT2);

    can1.acc_code=0; /* */
    can1.acc_mask=0xff; /* */
    can1.bt0=03; /*baud rate 1Mbps*/
    can1.bt1=0x1c;
    if(canConfig(PORT1,can1)==FAIL)
    {
        clrscr();
        printf("\n\n CAN Port %d Configuration
Error",1);
        return;
    }
    memcpy(&can2, &can1, sizeof(CAN_STRUCT));
    if (canConfig(PORT2,can2)==FAIL)
    {
        clrscr();
        printf("\n\n CAN Port %d Configuration
Error", 2);
        return;
    }
    canNormalRun(PORT1); /*Put CAN1 into normal mode.*/
    canNormalRun(PORT2); /*Put CAN2 into normal mode.*/

    clrscr();

    smsg1.id = 0x015; /* Set ID =8 */
    smsg1.rtr=1; /* Data lengths =8 */
    smsg1.dlen=8;
    for(i=0; i<smsg1.dlen; i++)
        smsg1.data[i] = i;
    while(1)
    {
        canSendMsg(PORT1, smsg1); /*Send to CAN1*/
        if (canReceiveMsg(PORT2, &rmsg2)==1)
        {
            printf("PORT2 receive:ID=%3X
RTR=%ld Length=%ld",
rmsg2.id,rmsg2.rtr, rmsg2.dlen);

```

```

        for(i=0; i< rmsg2.dlen; i++)
            cprintf(" %2X",rmsg2.data[i]);
        printf("\n");
    }
    if (kbhit())
    {
        getch();
        break;
    }
}

/* Reset CAN controller. */
canReset(PORT1);
canReset(PORT2);
canExitHW();
clrscr();
}

```

## DataMonitor Utility

### Software Overview

The PCM-3680 comes with a utility disk with the following software capabilities:

- CAN controller configuration
- CAN transmission monitoring
- Terminal emulation

### Main Menu

Run DataMonitor at the DOS prompt. DataMonitor's main menu screen will appear as shown below:



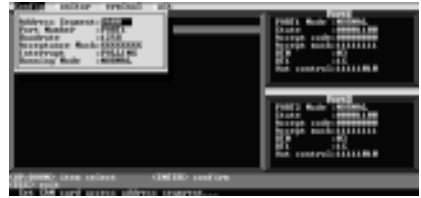
The main screen consists of:

- Menu bar:** Lists the available functions. From the main menu you can select Configuration, Monitoring, and Terminal.
- Monitor screen:** Shows monitored data, including message index, CAN device ID, data length, and data.
- Status fields:** Display the status of the two ports and the status register of the CAN controllers.
- On-line help/message bar:** Shows various key commands and states the function of the currently highlighted item.

### Configuration

Before you transmit a CAN object, you must configure the CAN controller by selecting the <Config> menu with the cursor keys and pressing <Enter>. The Configuration function determines the ports to be used and their communication parameters.

The port configuration window is shown below.



The parameters below need to be configured for each CAN controller:

**Address segment:** The base address (address segment) of the PCM-3680 is normally adjusted during the installation process. The selection of the address segment needs to be the same as that of the hardware configuration.

**Port:** Select the port you want to configure.

**Baud rate:** The baud rate must be coordinated with the CAN network. Choose the appropriate one from the list of baud rates.

**Acceptance code:** Specifies the value of the 8 most significant bits of the identifier (ID10 ... ID 3)

**Acceptance mask:** Specifies the bit positions which are "relevant" for acceptance filtering.

**Note:** The acceptance code and acceptance mask are configured through eight digits (1 digit per bit) using 0 or 1.

| Value | Definition   |
|-------|--|
| 0     | This bit position will accept only a "relevant" message. |
| 1     | This bit position will not screen messages.              |

**Example:** Acc Code = 11111111  
Acc Mask =11111111

The shown acceptance filter will accept every received message.

**Interrupt:** Sets the interrupt for each port. Be sure that this setting matches the IRQ already selected for the PCM-3680, which accepts values between IRQ3 to IRQ15, except 8 and 13.

**Running mode:** During the normal configuration and communication process, select Normal Mode. When the system fails, you can hit <Enter> to reset the CAN controller. Hit <Enter> again to return to Normal Mode to further execute your configuration.

### Monitor

Select the port to be monitored from the <Monitor> pull-down menu. Press F3 to start and stop the monitoring process.

Monitor Screen

The monitored data for a selected port appears in the monitor screen (see area **B** in the diagram under Main Menu section.)

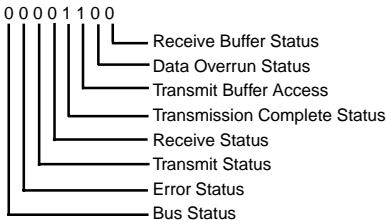
If the CAN controller is configured correctly and the transmission has been successfully completed, every CAN object will be shown in order of appearance.

Status Fields

Status fields at the right of the screen display the status of the two ports:



The status fields show information including the Mode (Normal or Reset), Acceptance Code, Acceptance Mask, BTR0, BTR1, Output Control Register, and Status Register. The normal value of the Status Register is:



The registers can only be read if the CAN controller is in Normal mode. If the CAN controller operates correctly and the transmission has been completed successfully, the status register will show as the normal value: 00001100. If the Error Status and/or the Bus Status is 1, you have selected the wrong baud rate or the CAN cable is damaged. Also check the correct bus terminator.

Terminal

This function provides a direct way to:

- 1. Send data over the CAN network.
- 2. Test CAN transmission.

Select <Terminal> from the menu bar for the following screen:



Transmitting Data

To transmit data, the PCM-3680 must be connected to a CAN network with at least one node and the configuration for the card must be complete.

First, select <Terminal> to edit the data. Enter the port, the object ID and the data bytes as hexadecimal value. Press <Enter> to begin data transmission. If the CAN controller is configured correctly and the transmission has been successfully completed, every CAN object will be shown in order of appearance at the left side of the screen.

Testing Data Transmission

To test CAN transmission without actually sending, connect Port 1 to Port 2 on the PCM-3680. Select <Terminal> and enter port 1 as transmitting port. Port 2 will therefore be designated as receiving port.

**Note:** To send Data Frame (Transmit), enter "0" for RTR. If you want to send Remote Frame (Request), enter "1" for RTR.

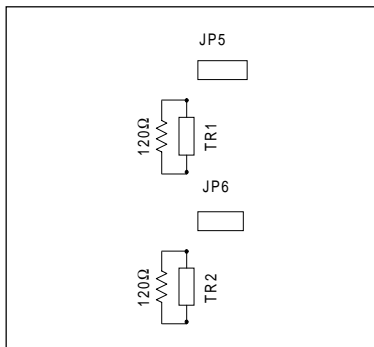
## Pin Assignments

The following figure shows the pin assignments for the card's DB-9 connectors.

### Termination Resistor Setup

Termination resistors are factory-installed to allow for impedance matching. These resistors can be enabled by using jumpers number 5 and 6 (shown below). Jumper 5 enables the terminal resistor for port 1, while jumper 6 enables the terminal resistor for port 2. The value of the resistor equals the characteristic impedance of the signal wires (approximately 120  $\Omega$ )

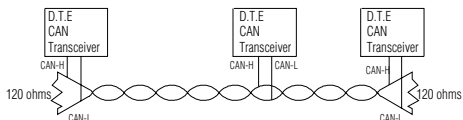
The following figure shows resistor placements. Port TR1 enables jumper JP5, and TR2 enables jumper JP6



**CAN termination resistor installation**

### CAN signal wiring

The CAN standard supports half-duplex communication. This means that just two wires are used to transmit and receive data.



**Wiring topography**

Wiring connections are as follows:

| PCM-3860 | DTE (male DB-9) | Terminal DTE |
|----------|-----------------|--------------|
| Pin      | Signal          | Signal       |
| 7        | CAN-H           | CAN-H        |
| 3        | GND             | GND          |
| 2        | CAN-L           | CAN-L        |

This appendix gives a brief description of the CAN controller registers. For more detailed information, please refer to the *Stand-alone CAN-controller Data Book* from Philips Semiconductors Microcontroller Products. (You may also find the information on the enclosed disk under the "Manual" directory, in the Word 6.0 file: REGISTER.DOC.)

## CAN Controller Address Allocation

### Philips PCX82C200 CAN Controller

#### ADDRESS

## Register address map

|    |                           |                 |
|----|---------------------------|-----------------|
| 0  | CONTROL                   | control segment |
| 1  | COMMAND                   |                 |
| 2  | STATUS                    |                 |
| 3  | INTERRUPT                 |                 |
| 4  | ACCEPTANCE CODE           |                 |
| 5  | ACCEPTANCE MASK           |                 |
| 6  | BUS TIMING 0              |                 |
| 7  | BUS TIMING 1              |                 |
| 8  | OUTPUT CONTROL            |                 |
| 9  | TEST                      |                 |
| 10 | IDENTIFIER                | descriptor      |
| 11 | RTR BIT, DATA LENGTH CODE |                 |
| 12 | BYTE 1                    |                 |
| 13 | BYTE 2                    |                 |
| 14 | BYTE 3                    |                 |
| 15 | BYTE 4                    |                 |
| 16 | BYTE 5                    |                 |
| 17 | BYTE 6                    |                 |
| 18 | BYTE 7                    |                 |
| 19 | BYTE 8                    |                 |
| 20 | IDENTIFIER                | descriptor      |
| 21 | RTR BIT, DATA LENGTH CODE |                 |
| 22 | BYTE 1                    |                 |
| 23 | BYTE 2                    |                 |
| 24 | BYTE 3                    |                 |
| 25 | BYTE 4                    |                 |
| 26 | BYTE 5                    |                 |
| 27 | BYTE 6                    |                 |
| 28 | BYTE 7                    |                 |
| 29 | BYTE 8                    |                 |

| #                         | TITLE                    | ADDRESS | 7          | 6            | 5                     | 4                        | 3                            | 2                         | 1                        | 0                     |
|---------------------------|--------------------------|---------|------------|--------------|-----------------------|--------------------------|------------------------------|---------------------------|--------------------------|-----------------------|
| <b>Control Segment</b>    |                          |         |            |              |                       |                          |                              |                           |                          |                       |
| 1                         | Control Register         | 0       | Test Mode  | Sync         | Reserved              | Overrun Interrupt Enable | Error Interrupt Enable       | Transmit Interrupt Enable | Receive Interrupt Enable | Reset Request         |
| 2                         | Command Register         | 1       | Reserved   | Reserved     | Reserved              | Go to Sleep              | Clear Overrun Status         | Release Receive Buffer    | Abort Transmission       | Transmission Request  |
| 3                         | Status Register          | 2       | Bus Status | Error Status | Transmit Status       | Receive Status           | Transmission Complete Status | Transmit Buffer Access    | Data Overrun             | Receive Buffer Status |
| 4                         | Interrupt Register       | 3       | Reserved   | Reserved     | Reserved              | Wake-Up Interrupt        | Overrun Interrupt            | Error Interrupt           | Transmit Interrupt       | Receive Interrupt     |
| 5                         | Acceptance Code Register | 4       | AC.7       | AC.6         | AC.5                  | AC.4                     | AC.3                         | AC.2                      | AC.1                     | AC.0                  |
| 6                         | Acceptance Mask Register | 5       | AM.7       | AM.6         | AM.5                  | AM.4                     | AM.3                         | AM.2                      | AM.1                     | AM.0                  |
| 7                         | Bus Timing Register 0    | 6       | SJW.1      | SJW.0        | BRP.5                 | BRP.4                    | BRP.3                        | BRP.2                     | BRP.1                    | BRP.0                 |
| 8                         | Bus Timing Register 1    | 7       | SAM        | TSEG2.2      | TSEG2.1               | TSEG2.0                  | TSEG1.3                      | TSEG1.2                   | TSEG1.1                  | TSEG1.0               |
| 9                         | Output Control Register  | 8       | OCTP1      | OCTN1        | OCPOL1                | OCTP0                    | OCTN0                        | OCPOL0                    | OCMODE1                  | OCMODE0               |
| 10                        | Test Register (note 1)   | 9       | Reserved   | Reserved     | Map Internal Register | Connect RX Buffer 0 CPU  | Connect TX Buffer CPU        | Access Internal Bus       | Normal RAM Connct        | Float Output Driver   |
| <b>Transmit Buffer</b>    |                          |         |            |              |                       |                          |                              |                           |                          |                       |
| 11                        | Identifier               | 10      | ID.10      | ID.9         | ID.8                  | ID.7                     | ID.6                         | ID.5                      | ID.4                     | ID.3                  |
|                           | RTR, Data Length Code    | 11      | ID.2       | ID.1         | ID.0                  | RTR                      | DLC.3                        | DLC.2                     | DLC.1                    | DLC.0                 |
|                           | Bytes 1-8                | 12-19   | Data       | Data         | Data                  | Data                     | Data                         | Data                      | Data                     | Data                  |
| <b>Receive Buffer 0/1</b> |                          |         |            |              |                       |                          |                              |                           |                          |                       |
| 12                        | Identifier               | 20      | ID.10      | ID.9         | ID.8                  | ID.7                     | ID.6                         | ID.5                      | ID.4                     | ID.3                  |
|                           | RTR, Data Length Code    | 21      | ID.2       | ID.1         | ID.0                  | RTR                      | DLC.3                        | DLC.2                     | DLC.1                    | DLC.0                 |
|                           | Bytes 1-8                | 22-29   | Data       | Data         | Data                  | Data                     | Data                         | Data                      | Data                     | Data                  |



### **Control Register (CR)**

The Control Register is used to change the behavior of the PCX82C200. Control bits may be set or reset by the attached microcontroller, which uses the Control Register as a read/write memory.

### **Command Register (CMR)**

A command bit initiates an action within the transfer layer of the PCX82C200. If a read access is performed to this address, the byte 11111111 (binary) is returned.

### **Status Register (SR)**

The Status Register reflects the status of the PCX82C200 bus controller. The Status Register appears to the microcontroller as a read-only memory.

### **Interrupt Register (IR)**

The Interrupt Register allows identification of an interrupt source. When one or more of this register's bits are set, the INT pin is activated. All bits are reset by the PCX82C200 after this register is read by the microcontroller. This register appears to the microcontroller as a read-only memory.

### **Acceptance Code Register (ACR)**

The Acceptance Code Register is part of the acceptance filter of the PCX82C200. This register can be accessed (read/write) if the Reset Request bit is set HIGH (present). When a message which passes the acceptance test is received and if there is an empty Receive Buffer, then the respective Descriptor and Data Field are sequentially stored in this empty buffer. In the case that there is no empty Receive Buffer, the Data Overrun bit is set HIGH (overrun).

### **Acceptance Mask Register (AMR)**

The Acceptance Mask Register is part of the acceptance filter of the PCX82C200. This register can be accessed (read/write) if the Reset Request bit is set HIGH (present). The Acceptance Mask Register classifies the corresponding bits of the acceptance code as "relevant" or "don't care" for acceptance filtering.

### **Bus Timing Register 0 (BTR0)**

The Bus Timing Register 0 defines the values of the Baud Rate Prescaler (BRP) and the Synchronization Jump Width (SJW). This register can be accessed (read/write) if the Reset Request bit is set HIGH (present).

### **Bus Timing Register 1 (BTR1)**

The Bus Timing Register 1 defines the length of the bit period, the location of the sample point, and the number of samples to be taken at each sample point. This register can be accessed (read/write) if the Reset Request bit is set HIGH (present).

### **Output Control Register (OCR)**

The Output Control Register allows, under software control, the setup of different driver configurations. This register may be accessed (read/write) if the Reset Request bit is set HIGH (present).

### **Test Register (TR)**

The Test Register is used only for production testing.

### **Transmit Buffer**

The Transmit Buffer stores a message from the microcontroller to be transmitted by the PCX82C200. It is subdivided into the Descriptor and Data Field. The Transmit Buffer can be written to and read from by the microcontroller.

### **Receive Buffer**

The layout of the Receive Buffer and the individual bytes correspond to the definitions given for the Transmit Buffer layout, except that the addresses start at 20 instead of 10.

