ARCHITECTURAL
OVERVIEW

82527 Serial Communications Controller Architectural Overview

Automotive

January 1996

Order Number: 272410-003

COPYRIGHT © INTEL CORPORATION, 1995

Information in this document is provided in connection with Intel products. Intel assumes no liability whatsoever, including infringement of any patent or copyright, for sale and use of Intel products except as provided in Intel's Terms and Conditions of Sale for such products.

Intel retains the right to make changes to these specifications at any time, without notice. Microcomputer Products may have minor variations to this specification known as errata.

*Other brands and names are the property of their respective owners.

†Since publication of documents referenced in this document, registration of the Pentium, OverDrive and iCOMP trademarks has been issued to Intel Corporation.

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

Intel Corporation
P.O. Box 7641
Mt. Prospect, IL 60056-7641

or call 1-800-879-4683

COPYRIGHT © INTEL CORPORATION, 1995

82527 Serial Communications Controller

CONTENTS

PAGE

CONTENTS

PAGE

1.0 GENERAL FEATURES

- Supports CAN Specification 2.0
  - — Standard Data and Remote Frames
  - — Extended Data and Remote Frames
- Programmable Global Mask
  - — Standard Message Identifier
  - — Extended Message Identifier
- 15 Message Objects of 8-byte Data Length
  - — 14 TX/RX Buffers
  - — 1 RX Buffer with Programmable Mask
- Flexible CPU Interface
  - — 8-bit Multiplexed
  - — 16-bit Multiplexed
  - — 8-bit Synchronous Non-Multiplexed
  - — 8-bit Asynchronous Non-Multiplexed
  - — Serial Interface
- Programmable Bit Rate
- Programmable Clock Output
- Flexible Interrupt Structure
- Flexible Status Interface
- Configurable Input Comparator
- Two 8-bit Bidirectional l/O Ports
- 44-lead PLCC Package/44-Lead QFP Package
- Pinout Compatibility with the 82526

The 82527 serial communications controller is a highly integrated device that performs serial communication according to the CAN protocol. The CAN protocol uses a multi-master (contention based) bus configuration for the transfer of "communication objects" between nodes of the network. This multi-master bus is also referred to as CSMA/CR or Carrier Sense, Multiple Access, with Collision Resolution. The 82527 performs all serial communication functions such as transmission and reception of messages, message filtering, transmit search, and interrupt search with minimal interaction from the host microcontroller, or CPU.

The 82527 is Intel's first device to support the standard and extended message frames in CAN Specification 2.0 part B. It has the capability to transmit, receive, and perform message filtering on extended message frames with a 29-bit message identifier. Due to the backwardly compatible nature of CAN Specification 2.0, the 82527 also fully supports the standard message frames in CAN Specification 2.0 part A.

A communication object consists of an identifier along with control data segments. The control segment contains all the information needed to transfer the message. The data segment contains from 0 to 8 bytes in a single message. All communication objects are stored in the RAM of the corresponding CAN chip for each node. A transmitting node broadcasts its message to all other nodes on the network. An acceptance filter at each node decides whether to receive that message. A message is accepted only if a communication object with the same message identifier has been set up in the CAN RAM for that node.

CAN not only manages the transmission and reception of messages but also the error handling, without any burden on the CPU.

CAN features several error detection mechanisms. These include Cyclical Redundancy Check (CRC) and bit coding rules ("bit stuffing/destuffing"). The polynomial of the CRC has been optimized for control applications with short messages. If a message was corrupted by noise during transmission, it is not accepted at the receiving nodes. Current transmission status is monitored in the control segment of the appropriate communication object within the transmitting node, automatically initiating a repeated transmission in the case of errors. CAN also has built-in mechanisms to locate error sources and to distinguish permanent hardware failures from occasional soft errors. Defective nodes are switched off the bus, implementing a fail-safe behavior (thus, hardware errors will not let defective nodes control the bus indefinitely).

The message storage is implemented in an intelligent memory, or RAM, which can be addressed by the CAN controller and the CPU. The CPU controls the CAN controller by selectively modifying the various registers and bit fields in the RAM. The content of the various bit fields are used to perform the functions of acceptance filtering, transmit search, interrupt search and transfer completion.

In order to initiate a transfer, the transmission request bit has to be written to the message object. The entire transmission procedure and eventual error handling is then done without any CPU involvement. If a communication object has been configured to receive messages, the CPU easily reads its data registers using CPU read instructions. The message object may be configured to interrupt the CPU after every successful message transmission or reception.

The 82527 features a powerful CPU interface that offers flexibility to directly interface to many different CPUs. It can be configured to interface with CPUs using an 8-bit multiplexed, 16-bit multiplexed, or 8-bit non-multiplexed address/data bus for Intel and non-Intel architectures. A flexible serial interface is also available when a parallel CPU interface is not required.

The 82527 provides storage for 15 message objects of 8-byte data length. Each message object can be configured as either transmit or receive except for the last message object. The last message object is a receive only buffer with a special acceptance mask designed to allow select groups of different message identifiers to be received.

The 82527 also implements a global acceptance masking feature for message filtering. This feature allows the user to globally mask any identifier bits of the incoming message. The programmable global mask can be used for both standard and extended messages.

The 82527 provides an improved set of network management and diagnostic functions including fault confinement and a built-in development tool. The built-in development tool alerts the CPU when a global status change occurs. Global status changes include message transmission and reception, error frames, or sleep mode wake-up. In addition, each message object offers full flexibility in detecting when a data or remote frame has been sent or received.

The 82527 offers hardware, or pinout, compatibility with the 82526. It is pin-to-pin compatible with the 82526 except for pins 9, 30, and 44. These pins are used as chip selects on the 82526 and are used as CPU interface mode selection pins on the 82527.

The 82527 is fabricated in Intel's reliable CHMOS III 5-V technology and is available in a 44-lead PLCC and 44-lead QFP for the automotive temperature range (−40°C to +125°C ambient).

1.1 Functional Overview

The 82527 CAN controller consists of six functional blocks. The CPU Interface logic manages the interface between the CPU (host microcontroller) and the 82527 using an address/data bus. The CAN controller interfaces to the CAN bus and implements the protocol rules of the CAN protocol for the transmission and reception of messages. The RAM is the interface layer between the CPU and the CAN bus. The two port blocks provide 8-bit low speed I/O capability. The clockout block allows the 82527 to drive other chips, such as the host-CPU.

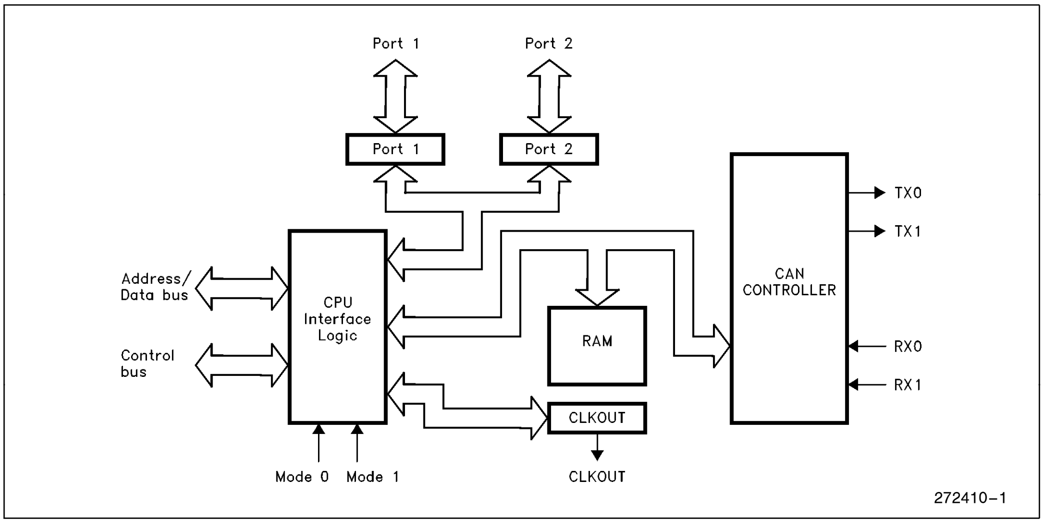
The 82527 RAM provides storage for 15 message objects of 8-byte data length. Each message object has a unique identifier and can be configured to either transmit or receive except for the last message object. The last message object is a receive only buffer with a special mask design to allow select groups of different message identifiers to be received.

Each message object contains control and status bits. A message object with the direction set as receive will send a remote frame by requesting a message transmission. A message object with the direction 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 or data frame has been sent or received.

The 82527 also implements a global masking feature 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-specific message identification strategy. There are separate global masks for standard and extended frames.

The incoming message first passes through the global mask and is matched to the identifiers in message objects 1–14. If there is no identifier match then the message passes through the local mask in message object 15. The local mask allows a large number of infrequent messages to be received by the 82527. Message object 15 is also buffered to allow the CPU time to service a message received.

A block diagram of the 82527 is shown below.



1.2 CAN Controller

The CAN controller controls the data stream between the RAM (parallel data) and the CAN busline (serial data). The CAN controller also manages the transceiver logic (RX0, RX1, TX0, TX1), the error management logic and the message objects.

1.3 RAM

The RAM is an interleaved access memory. This means the access to the RAM is timeshared between the CPU Interface Logic and the CAN bus (through the CAN controller). The RAM is addressed from 00H to FFH.

1.4 CPU Interface Logic

The 82527 provides a flexible CPU interface capable of interfacing to many commonly used microcontrollers. Five modes are selected using two CPU interface mode pins. Mode 0 (Mode1 pin = 0, Mode0 pin = 0) selects an 8-bit Intel multiplexed address data bus. If the RD# and WR# pins are tied low at reset in Mode 0, the serial interface (SPI) mode is entered. Mode 1 (Mode1 pin = 0, Mode0 pin = 1) selects 16-bit Intel multiplexed address data bus. Mode 2 (Mode1 pin = 1, Mode0 pin = 0) selects an 8-bit non-Intel multiplexed address data bus. Lastly, Mode 3 (Mode1 pin = 1, Mode0 pin = 1) selects an 8-bit non-multiplexed address data bus for either synchronous or asynchronous communication.

1.5 Clockout

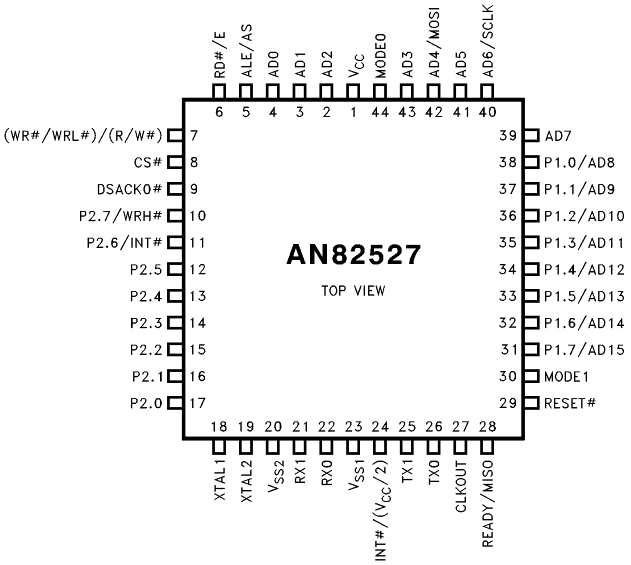
The on-chip clock generator consists of an oscillator, clock divider register and a driver circuit. The Clockout output range is XTAL (external crystal frequency) to XTAL/15. The Clockout output slew rate is programmable.

1.6 Two 8-Bit Ports

Two 8-bit low speed input/output (I/O) ports are available on-chip. Depending on the CPU interface selected, at least 7 and up to 16 of these I/O pins are available for system use.

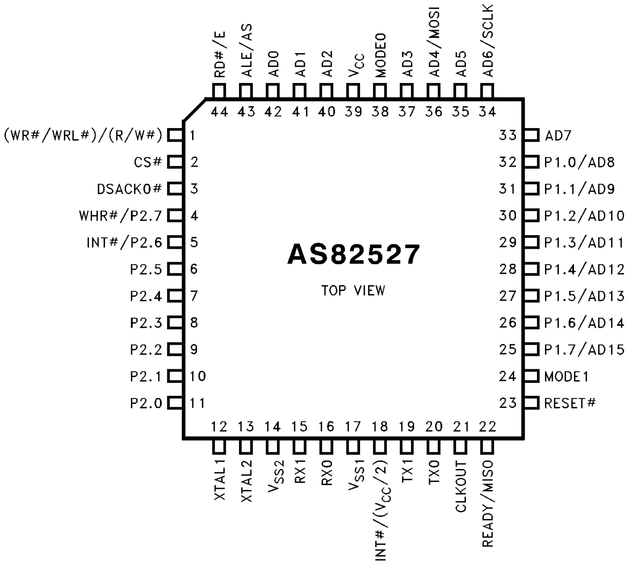
2.0 PACKAGE DIAGRAM/PIN OUT

44-Pin PLCC Package



272410–2

44-Pin QFP Package



272410–33

3.0 PIN DESCRIPTION

Pin	Description
V_{SSI}	Ground (0V) connection must be shorted externally to a V_{SS} board plane to provide digital ground.
V_{SS2}	Ground (0V) connection must be shorted externally to a V_{SS} board plane to provide ground for analog comparator.
V_{CC}	Power connection must be shorted externally to +5V DC to provide power to the entire chip.
XTAL1	Input for an external clock. XTAL1 (along with XTAL2) are the crystal connection to an internal oscillator.
XTAL2	Push-pull output from the internal oscillator. XTAL2 and XTAL1 are the crystal connections to an internal oscillator. If an external oscillator is used, XTAL2 must be floated or not be connected. XTAL2 must not be used as a clock output to drive other CPUs.
CLKOUT	Programmable clock output. This push-pull output may be used to drive the oscillator of the CPU.
RESET#	Warm Reset: (V_{CC} remains valid while RESET# is asserted), Reset# must be driven to a low level for 1 ns minimum. Cold Reset: (V_{CC} is driven to a valid level while Reset# is asserted), Reset# must be driven low for 1 ns minimum measured from a valid V_{CC} level. No falling edge on the Reset pin is required during a cold reset event.
CS#	A low level on this pin enables the CPU to access the 82527.
INT# or (V_{CC}/2)	The interrupt pin is an open collector output (requires external pullup resistor) to the CPU. V_{CC}/2 is the power supply for the ISO low speed physical layer. The function of this pin is determined by the MUX bit in the CPU Interface Register (Address 02H) when the DcR1 bit (Address 2FH) is set: when MUX = 1 and DcR1 = 1: then pin 24 = V_{CC}/2, pin 11 = INT# when MUX = 0: then pin 24 = INT#
RX0 RX1	Inputs from the CAN bus line(s) to the input comparator. A recessive level is read when RX0 > RX1. A dominant level is read when RX1 > RX0. When the CoBy bit (Bus Configuration register) is programmed as a "1", the input comparator is bypassed and RX0 is the CAN bus line input.
TX0 TX1	Serial push-pull data output to the CAN bus line. During a recessive bit, TX0 is high and TX1 is low. During a dominant bit, TX0 is low and TX1 is high. TX0/TX1 suggestion: Unlike the Intel 82526, the 82527 TX0 and TX1 output drivers can not be individually programmed to transmit either recessive or dominant bits; this is fixed as described in the TX0/TX1 definition. If 82527 and 82526 devices are not communicating on a CAN bus, the problem may be due to TX0/TX1 configuration differences. Reversing the TX0/TX1 connections for either device may allow these devices to communicate.
Ports1/2	Port1 and Port2 pins are weakly held high until the Port configuration registers have been written (locations 9FH and AFH respectively).

3.0 PIN DESCRIPTION (Continued)

<table>
<tr><th>Pin</th><th>Description</th></tr>
<tr><td>AD0–AD15</td><td>The functions of these pins are defined below.
<table>
<tr><th></th><th>8-Bit Intel Multiplexed</th><th>8-Bit Non-Intel Multiplexed</th><th>16-Bit Intel Multiplexed</th><th>8-Bit Non-Multiplexed</th><th>Serial Interface</th></tr>
<tr><td>AD0</td><td>AD0</td><td>AD0</td><td>AD0</td><td>A0</td><td>ICP</td></tr>
<tr><td>AD1</td><td>AD1</td><td>AD1</td><td>AD1</td><td>A1</td><td>CP</td></tr>
<tr><td>AD2</td><td>AD2</td><td>AD2</td><td>AD2</td><td>A2</td><td>CSAS</td></tr>
<tr><td>AD3</td><td>AD3</td><td>AD3</td><td>AD3</td><td>A3</td><td>STE</td></tr>
<tr><td>AD4</td><td>AD4</td><td>AD4</td><td>AD4</td><td>A4</td><td>MOSI</td></tr>
<tr><td>AD5</td><td>AD5</td><td>AD5</td><td>AD5</td><td>A5</td><td>Unused</td></tr>
<tr><td>AD6</td><td>AD6</td><td>AD6</td><td>AD6</td><td>A6</td><td>SCLK</td></tr>
<tr><td>AD7</td><td>AD7</td><td>AD7</td><td>AD7</td><td>A7</td><td>Unused</td></tr>
<tr><td>AD8</td><td>Port 1.0</td><td>Port 1.0</td><td>AD8</td><td>D0</td><td>Port 1.0</td></tr>
<tr><td>AD9</td><td>Port 1.1</td><td>Port 1.1</td><td>AD9</td><td>D1</td><td>Port 1.1</td></tr>
<tr><td>AD10</td><td>Port 1.2</td><td>Port 1.2</td><td>AD10</td><td>D2</td><td>Port 1.2</td></tr>
<tr><td>AD11</td><td>Port 1.3</td><td>Port 1.3</td><td>AD11</td><td>D3</td><td>Port 1.3</td></tr>
<tr><td>AD12</td><td>Port 1.4</td><td>Port 1.4</td><td>AD12</td><td>D4</td><td>Port 1.4</td></tr>
<tr><td>AD13</td><td>Port 1.5</td><td>Port 1.5</td><td>AD13</td><td>D5</td><td>Port 1.6</td></tr>
<tr><td>AD14</td><td>Port 1.6</td><td>Port 1.6</td><td>AD14</td><td>D6</td><td>Port 1.6</td></tr>
<tr><td>AD15</td><td>Port 1.7</td><td>Port 1.7</td><td>AD15</td><td>D7</td><td>Port 1.7</td></tr>
</table></td></tr>
<tr><td>P2.0
P2.1
P2.2
P2.3
P2.4
P2.5
P2.6/INT#</td><td>Port 2 function in all CPU interface modes.
P2.6 is INT# when MUX = 1 in the CPU Interface register (02H).</td></tr>
<tr><td>P2.7/WRH#</td><td>P2.7 is WRH# in 16-bit multiplexed mode (Mode1).</td></tr>
<tr><td>Mode0/
Mode1</td><td>These pins select one of the four parallel interfaces:
<table>
<tr><th><u>Mode1</u></th><th><u>Mode0</u></th><th></th><th></th></tr>
<tr><td>0</td><td>0*</td><td>8-bit multiplexed</td><td>Intel</td></tr>
<tr><td>0</td><td>1</td><td>16-bit multiplexed</td><td>Intel</td></tr>
<tr><td>1</td><td>0</td><td>8-bit multiplexed</td><td>non-Intel</td></tr>
<tr><td>1</td><td>1</td><td>8-bit non-multiplexed</td><td></td></tr>
</table>
* Note: If upon reset, Mode0 = Mode1 = 0, RD# = 0 and WR# = 0, then the serial interface mode is entered.
Mode0 and Mode1 pins are internally connected to weak pulldowns. These pins will be pulled low during reset if unconnected. Following reset, these pins float.</td></tr>
<tr><td>ALE/
AS</td><td>ALE used for Intel CPU Interface Modes 0 and 1.
AS used for non-Intel modes. Except Mode 3, this pin must be tied high.</td></tr>
</table>

3.0 PIN DESCRIPTION (Continued)

Pin	Description
RD#	RD# used for CPU Interface Modes 0 and 1.
E	E used for non-Intel modes. Except Mode 3 Asynchronous, this pin must be tied high.
WR#/WRL# R/W#	WR# used for Intel CPU Interface Modes 0 and 1. (WRL# function in Mode1, 16-bit Mode.) R/W# used for CPU Interface Mode3.
READY/ MISO	READY is an open-drain output to synchronize accesses from the CPU to the 82527 for CPU Interface Modes 0 and 1. MISO is the serial data output for the serial interface mode.
DSACK0#	DSACK0# is an open-drain output to synchronize accesses from the CPU to the 82527 for CPU Interface Mode3. DSACK0# is often used as a pulldown output with a 3.3 kΩ pullup resistor and a 100 pF load capacitance. An open-drain output is used because many peripherals may be connected to the DSACK0# line. The 82527 specifies a TCHKH timing (CS# high to DSACK0# high) equal to 55 ns, however a 3.3 kΩ resistor will not sufficiently charge the line when DSACK0# is floated by the 82527. To meet this timing, the 82527 has an active pullup that drives the DSACK0# output until it is high, and then the pullup is turned off. Therefore, the pullup is only active for a short time.

3.1 Hardware Reset

During power up, the RESET# pin must be driven to a valid low level (0.8V) for 1 ms measured from a valid V_{CC} level to ensure the oscillator is stable. The registers of the 82527 have the following values after warm reset:

Register	Reset Value
Control Register (00H)	01H
Status Register (01H)	Undefined
CPU Interface Register (02H)	61H
High Speed Register (04–05H)	Unchanged
Global Mask Short (06–07H)	Unchanged
Global Mask Long (08–0BH)	Unchanged
Mask Last Message (0C–0FH)	Unchanged
Clockout Register (1FH)	00H or 01H depending on CPU Interface Mode
Bus Configuration (2FH)	00H
Bit Timing Register 0 (3FH)	Unchanged
Bit Timing Register 1 (4FH)	Unchanged
Interrupt Register (5FH)	00H
P1 Configuration Register (9FH)	00H
P2 Configuration Register (AFH)	00H
P1 In (BFH)	FFH
P2 In (CFH)	FFH
PI Out (DFH)	00H
P2 Out (EFH)	00H
SPI Reset Address (FFH)	Undefined
Messages 1–15	Unchanged

The error management counters and the busoff state are reset by a hardware reset.

If a hardware reset occurs at power on, registers defined as unchanged should be interpreted as undefined.

Pins have the following states after reset:

Pin	Reset state
Mode 0/1	0 - while RESET# is active (high impedance - after reset)
Port 1/2	1 - weakly pulled high (following configuration, output or high impedance input)
Clockout	0
TX0	1 - recessive state
TX1	0 - recessive state
INT#	Float
DSACK0#	Float

3.2 Software Initialization

Software initialization is started by setting the Init bit in the Control Register, either by software, hardware reset, or by going busoff. While Init is set, all message transfers to and from the 82527 are stopped and the TX0 and TX1 outputs are recessive. The error counters are unchanged. Initialization is used to configure the 82527 RAM without risk of CAN bus receptions or transmissions.

Resetting Init completes initialization and the 82527 synchronizes itself to the CAN bus by waiting for 11 consecutive recessive bits (called bus idle) before it will take part in bus activities.

Note that busoff recovery cannot be hastened by setting or resetting the Init bit. If the 82527 goes busoff, the 82527 will set the Init bit itself and thereby stopping its bus activities. Once Init is cleared by the CPU, the 82527 will wait for 128 occurrences of bus idle before resuming normal operation. During the initialization sequence, each time eleven recessive bits are received, a Bit0 Error code is written to the status register enabling the CPU to readily check whether or not the CAN bus is stuck in a dominant state.

Software initialization does not change configuration register values.

4.0 FUNCTIONAL DESCRIPTION

This section discusses the functional operation of the 82527 by describing the registers used to configure the chip and message objects. The 82527 address map is shown in Section 4.1.

4.1 82527 Address Map

00H	Control Register
01H	Status Register
02H	CPU Interface Reg.
03H	Reserved
04–05H	High Speed Read
06–07H	Global Mask - Standard
08–0BH	Global Mask - Extended
0C–0FH	Message 15 Mask
10–1EH	**Message 1**
1FH	CLKOUT Register #
20–2EH	**Message 2**
2FH	Bus Config. Reg. #
30–3EH	**Message 3**
3FH	Bit Timing Reg. 0 #
40–4EH	**Message 4**
4FH	Bit Timing Reg. 1 #
50–5EH	**Message 5**
5FH	Interrupt Register
60–6EH	**Message 6**
6FH	Reserved
70H–7EH	**Message 7**
7FH	Reserved
80–8EH	**Message 8**
8FH	Reserved
90–9EH	**Message 9**
9FH	P1CONF #
A0–AEH	**Message 10**
AFH	P2CONF #
B0–BEH	**Message 11**
BFH	P1IN
C0–CEH	**Message 12**
CFH	P2IN
D0–DEH	**Message 13**
DFH	P1OUT
E0–EEH	**Message 14**
EFH	P2OUT
F0–FEH	**Message 15**
FFH	Serial Reset Address

NOTE:
denotes configuration registers. The CPU may write to these registers if CCE bit = 1 (Control register).

4.2 Control Register (00H)

7	6	5	4	3	2	1	0
0	CCE	0	0	EIE	SIE	IE	Init
rw	rw	r	r	rw	rw	rw	rw

r = readable
w = writable

The default value of the Control Register after a hardware reset is 01H.

CCE Change Configuration Enable

one The CPU has write access to configuration registers.

zero The CPU has no write access to configuration registers.

Configuration register addresses are 1FH, 2FH, 3FH, 4FH, 9FH, AFH. This bit is re-set by the CPU to provide protection against unintentional re-writing of critical registers by the CPU following the initialization sequence.

EIE Error Interrupt Enable

one Error interrupts enabled. A change in the error status of the 82527 will cause an interrupt to be generated.

zero Error interrupts disabled. No error interrupt will be generated.

Error interrupts are BOff and Warn in the status register. Error Interrupt Enable is set by the CPU to allow the 82527 to interrupt CPU when an abnormal number of CAN bus errors have been detected. It is recommended to enable this interrupt during normal operation.

SIE Status Change Interrupt Enable

one Status Change interrupt enabled. An interrupt will be generated when a CAN bus error is detected in the Status Register or a transfer (reception or transmission) is successfully completed, independent of the interrupt enable bits in any message object.

zero Status Change interrupt disabled. No status interrupt will be generated.

Status change interrupts are Wake, RXOK, TXOK, and LEC0-2 in the status register and this bit is set by the CPU. RXOK occurs upon every successful message transmission on the CAN bus, regardless of whether the message is stored by the 82527.

This interrupt is useful for hardware development to detect bus errors caused by physical layer issues such as noise. The LEC bits are very helpful to indicate whether bit or form errors are occurring. In normal operation it is not advised to enable this interrupt for LEC errors since the CAN protocol was designed to handle these error conditions in hardware by error frames and the automatic retransmission of messages. When cumulative LEC errors result, the warning and busoff flags will be set and the Error Interrupt should be enabled to detect these conditions.

In most applications, this bit should not be set. Since this interrupt will occur for every message, the CPU will be unnecessarily burdened. Instead, interrupts should be implemented on a message object basis so interrupts occur only for messages that are used by the CPU.

If the Status Change Interrupts and message object receive/transmit interrupts are enabled, there will be two interrupts for each message successfully received by a message object.

IE Interrupt Enable

one Global interrupts enabled. Applies to EIE, SIE, and message object TX/RX interrupts.

zero Global interrupts disabled. The 82527 will generate no interrupts although the interrupt register (5FH) will still be updated. If the interrupt contains some value other than zero when this bit is set to one, an interrupt will be generated. For example, no interrupt will be lost because of periodic setting or resetting of this bit.

The Interrupt Enable bit is set by the CPU.

Init Initialization

one Software initialization is enabled.

zero Software initialization is disabled.

Following a hardware reset, this bit will be set. The Init bit is written by the CPU and is set by the 82527 when it goes busoff. Initialization is a state which allows the user to configure the 82527 RAM without the chip participating in any CAN bus transmissions. While Init equals one, all message transfers to and from the CAN bus are stopped, and the status of the CAN bus outputs, TX0 and TX1 are recessive.

Initialization will most often be used the first time after power-up and when the 82527 has removed itself from the CAN bus after going busoff. Init should not be used in normal operation when the CPU is modifying transmit data; the CPU Update bit in each message object is used in this case.

Init set to one does not break a transmission or reception of a message in process, but will stop the 82527 from transmitting or receiving the next message.

Please see section 3.2 for additional information.

Reserved Bits 7, 5, and 4

one This value must not be programmed by the user.

zero A zero must always be written to this bit.

4.3 Status Register (01H)

7	6	5	4	3	2	1	0
BOff	Warn	Wake	RXOK	TXOK	LEC2	LEC1	LEC0
r	r	r	rw	rw	rw	rw	rw

The default value of the Status Register after a hardware reset is undefined.

BOff Bus Off Status

one There is an abnormal rate of occurrences of errors on the CAN bus. This condition occurs when an error counter in the 82527 has reached the limit of 256. This results in the 82527 going busoff. During busoff, no messages can be received or transmitted. The only way to exit this state is by resetting the Init bit in the Control register (location 00H). When this bit is reset, the busoff recovery sequence begins. The busoff recovery sequence resets the transmit and receive error counters. After the 82527 counts 128 packets of 11 consecutive recessive bits on the CAN bus, the busoff state is exited.

zero The 82527 is not busoff.

The Bus Off Status bit is written by the 82527.

Warn Warning Status

one There is an abnormal rate of occurrences of errors on the CAN bus. This condition occurs when an error counter in the 82527 has reached the limit of 96.

zero There is no abnormal occurrence of errors.

The Warning Status bit is written by the 82527. When this bit is set, an interrupt will occur if the EIE and IE bits of the Control Register (00H) are set. The Warning Status bit is written by the 82527.

Wake Wake up Status

one This bit is set when the 82527 had been previously set into Sleep mode by the CPU, and bus activity occurs. The Sleep bit or the Powerdown bit in the CPU Interface register is reset (location 02H) by the CPU.

zero The Wake Up interrupt is reset by reading the Status Register.

Setting the SLEEP bit (bit 3, register 02H) to a "1" will place the 82527 into SLEEP mode. While in SLEEP mode, the WAKE bit is "0". The WAKE bit will become "1" when bus activity is detected or when the CPU writes the SLEEP bit to "0". The WAKE bit will also be set to "1" after the 82527 comes out of Power Down mode. This bit is written by the 82527.

RXOK Receive Message Successfully

one Since this bit was last reset to zero by the CPU, a message has been successfully received.

zero Since this bit was last reset by the CPU, no message has been successfully received. This bit is never reset by the 82527.

A successfully received message may be any CAN bus transmission that is error-free, regardless of whether the 82527 has configured a message object to receive that particular message identifier. This bit may be cleared by the CPU. The 82527 will set this bit, but will not clear it.

TXOK Transmit Message Successfully

one Since this bit was last reset to zero by the CPU, a message has been successfully transmitted (error free and acknowledged by at least one other node).

zero Since this bit was last reset by the CPU, no message has been successfully transmitted. This bit is never reset by the 82527.

This bit may be cleared by the CPU. The 82527 will set this bit, but will not clear it.

LEC 0–2 Last Error Code
This field contains a code which indicates the type of the first error to occur in a frame on the CAN bus. If a message is without error the field will be cleared to 0. The code 7 is unused and may be written by the CPU to check for updates.

0 No error

1 Stuff Error

More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.

2 Form Error

The fixed format part of a received frame has the wrong format.

3 Acknowledgment Error (AckError)

The message transmitted by this device was not acknowledged by another node.

4 Bit 1 Error

During the transmission of a message (with the exception of the arbitration field), the 82527 wanted to send a recessive level (bit of logical value 1), but the monitored CAN bus value was dominant.

5 Bit 0 Error

During the transmission of a message (with the exception of the arbitration field), the 82527 wanted to send a dominant level (bit of logical value 0), but the monitored CAN bus value was recessive. During busoff recovery, this status is set each time a recessive bit is received (indicating the CAN bus is not stuck dominant).

6 CRC Error

The CRC checksum was incorrect in the message received. The CRC received for an incoming message does not match with the CRC value calculated by this device for the received data.

7 Unused

Status Interrupts

The status change interrupt has a value of 1. The Status Register must be read if a status change interrupt occurs. **NOTE:** Reading the status register will clear the status change interrupt (value = 1) from the Interrupt Register (5FH), if a status change interrupt is pending. A status change interrupt will occur on every successful reception or transmission, regardless of the state of the RXOK and TXOK bits. Therefore, if TXOK is set and a subsequent transmission occurs, an interrupt will occur (if enabled) even though TXOK was previously equal to one.

There are two ways to implement receive and transmit interrupts. The difference between these two methods is one relies on the hardwired priority of the message objects and the other is suitable for polling. The first and preferred method uses the TXIE and RXIE bits in the message control register for each corresponding message object. Whenever a message is transmitted or received by this message object, the corresponding interrupt is serviced in accordance with its priority (if the IE bit of register 00H is set). This method uses the hardwired priority scheme of the 82527 which requires minimal CPU intervention.

The second method sets the SIE bit of the Control Register to "1" which will force an interrupt whenever successful message transmissions or receptions occur. The RXOK and TXOK bits will be set when any of the message objects transmits or receives a message. A successfully received message may be any CAN bus transmission that is error-free, regardless of whether the 82527 has configured a message object to receive that particular message identifier. This method allows the user to more easily define the interrupt priority of each message object by polling the message objects following an SIE interrupt.

4.4 CPU Interface Register (02H)

7	6	5	4	3	2	1	0
RstST	DSC	DMC	PwD	Sleep	MUX	0	CEn
r	rw	rw	rw	rw	rw	rw	rw

The default value of the CPU Interface Register in hardware reset is E1H. The default value of the CPU Interface Register after hardware reset (RESET# is deactivated) is 61H.

RstSt Hardware Reset Status

one The hardware reset of the 82527 is active (RESET# is low). While reset is active, no access to the 82527 is possible.

zero Normal operation. The CPU must ensure this bit is zero before the first access to the 82527 after reset.

This bit is written by the 82527.

DSC Divide System Clock (SCLK). The SCLK may not exceed 10 MHz. See section 4.5.

one The system clock, SCLK, is equal to XTAL/2.

zero The system clock, SCLK, is equal to XTAL.

This bit is written by the CPU.

DMC Divide Memory Clock. The memory clock may not exceed 8 MHz. See section 4.5.

one The memory clock, MCLK, is equal to SCLK/2.

zero The memory clock, MCLK, is equal to SCLK.

This bit is written by the CPU.

PwD Power Down Mode enable

Sleep Sleep Mode enable

PwD	Sleep	
zero	zero	Both Power Down and Sleep Modes are not active.
one	zero	Power Down Mode is active.
zero	one	Sleep Mode is active.

These bits are written by the CPU.

MUX Multiplex for ISO Low Speed Physical Layer.

If $V_{CC}/2$ is used to implement the basic CAN physical layer, pin 24 provides the voltage output $V_{CC}/2$, and pin 11 is the interrupt output transmitted to the CPU. Otherwise, only the interrupt is available on pin 24. $V_{CC}/2$ is only available during normal operation and during Sleep Mode and not during Power Down Mode.

NOTE:

The DcR1 bit (Address 2FH) must be set to enable $V_{CC}/2$ on Pin 24.

one ISO low speed physical layer active: Pin 24 = $V_{CC}/2$, Pin 11 = INT#.

zero Normal operation: Pin 24 = INT#, Pin 11 = P2.6.

This bit is written by the CPU.

Reserved Bit 1

one This value must not be programmed by the user.

zero A zero must always be written to this bit.

CEn Clockout enable

one Clockout signal is enabled, (default after reset).

zero Clockout signal is disabled.

Low Current Modes

Power Down and Sleep Modes are activated by the PwD and Sleep bits in the CPU Interface register (02H) under the control of the programmer. This register is accessible during reset and normal operation. In both modes the oscillator and clockout output are not active and no access to the message objects is possible. Access to the CPU Interface register (02H) is allowed.

The 82527 exits from Power Down by either a hardware reset or by resetting the PwD bit to "0". The CPU must read the hardware reset bit, RstSt, (bit 7, register 02H) to ensure the 82527 has exited Power Down.

The 82527 enters Sleep Mode after the Sleep bit in the CPU Interface register (bit 3, register 02H) is set. The Sleep current is dependent on the MUX bit value of the CPU Interface register (bit 2, register 02H). When the $V_{CC}/2$ feature is enabled, ICC is specified to be 700 μA maximum and is 100 μA with the INT# feature enabled. Sleep mode is exited by resetting the Sleep bit or when there is activity on the CAN bus. The 82527 requires a minimum of 10 ms to come out of Sleep Mode after bus activity occurs.

Power Down and Sleep Mode should not be entered directly after RESET. The user program must perform a minimum RAM configuration at any time (preferably during the initialization) prior to entering these modes. Programming the following registers satisfies the minimum configuration requirement.

Control Register (00H) - set CCE bit to "1",
Bus Configuration Register (2FH)
Bit Timing Register 1 (4FH)
Control Register (00H) - Init bit reset to "0"

Writing these registers will activate 82527 circuitry required to ensure minimum power consumption.

4.5 Clocking Description

The clocking of the 82527 is dependent upon the oscillator (XTAL), the system clock (SCLK) and the memory clock (MCLK). The SCLK and MCLK frequencies are determined by the external oscillator (XTAL) and the DSC and DMC bits in the CPU Interface register (02H).

The 82527 is tested with XTAL set to 8 MHz and 16 MHz. Characterization data verifies the 82527 will operate with XTAL equal to 4 MHz.

The SCLK may be equal to XTAL or XTAL/2 depending upon the DSC bit value of the CPU Interface register. The SCLK controls the processing functions of the 82527 such as bit timing control and transceiver control logic. The MCLK may be equal to SCLK/2 or SCLK depending upon the value of the DMC bit. The MCLK controls the CPU interface timings and has a direct relationship to host CPU-to-82527 communications rate.

The SCLK is restricted to a 10 MHz maximum frequency, and the MCLK is restricted to a 8 MHz maximum frequency. The SCLK is used to calculate tq which is referenced in the Bit Timing Register 0 (3FH) description. The MCLK is used to define AC timings' specifications.

The maximum MCLK frequency for various oscillator frequencies is shown below:

fXTAL	SCLK (DSC bit)	MCLK (DMC bit)
4 MHz	4 MHz (0)	4 MHz (0)
8 MHz	8 MHz (0)	8 MHz (0)
10 MHz	10 MHz (0)	5 MHz (1)
12 MHz	6 MHz (1)	6 MHz (0)
16 MHz	8 MHz (1)	8 MHz (0)

Frequency of SCLK = fXTAL/(1 + DSC bit)

Frequency of MCLK = fSCLK/(1 + DMC bit)
= fXTAL/[(1 + DSC bit) × (1 + DMC bit)]

The SCLK (system clock) is used to control bit timings and the transceiver circuitry or in other words, the CAN bus. The MCLK (memory clock) is used to control the 82527 timings used for the 82527/CPU-host interface.

4.6 High Speed Read Register (04–05H)

04H

7	6	5	4	3	2	1	0
Low Byte							
r							

05H

7	6	5	4	3	2	1	0
High Byte							
r							

The High Speed Read register is a read only register and is the output buffer for the CPU Interface Logic. This register is part of the CPU Interface Logic and is not located in the RAM. During a read to the RAM (low speed registers) this register is loaded with the value of the low speed register being accessed.

The High Speed Read register is available to provide a method to read the 82527 when the CPU (host microcontroller) is unable to satisfy read cycle timings for low speed 82527 registers. In other words, if the read access time of the 82527 is too slow for the CPU and the CPU cannot extend the read bus cycle, the following method should be used.

The default value of the high speed read register after a hardware reset is unchanged. "Unchanged" default values should be interpreted as undefined if a hardware reset occurs during power on.

Double Read Operation

The CPU can execute double reads where the first read addresses the low speed register and the second read addresses the High Speed Read register. The first read is a dummy read for the CPU, however the low speed register value is stored in the High Speed Read register. The second read to the High Speed Read register will produce the data from the desired low speed register.

The advantage of double reads is both read operations have fast access times. The first read of the low speed register requires 40 ns (verify in current data sheet) to load the High Speed Read Register (the data on the address/data pins is not valid). The second read of the High Speed Read register requires 45 ns (verify in current data sheet) and the data on the address/data bus is valid.

Therefore, if the access time of a low speed register is too long for the CPU then a second read to the High Speed Register will produce the correct data. Please note low and High Speed Registers have different access timing specifications in the 82527 data sheet.

During a 16-bit read access the low and high byte will contain the 16-bit value from the read access. For an 8-bit read access the low byte will contain the value from the read access.

4.7 Global Mask—Standard Register (06–07H)

06H

7	6	5	4	3	2	1	0
ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
rw	rw	rw	rw	rw	rw	rw	rw

07H

7	6	5	4	3	2	1	0
ID20	ID19	ID18	Reserved				
rw	rw	rw					

Reserved bits read as 1

The default value of the Global Mask - standard after a hardware reset is unchanged.

The Global Mask - standard register applies only to messages using the standard CAN identifier, or for message objects with the XTD bit set to "0". This feature, also called message acceptance filtering, 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 develop an application specific masking strategy.

A "0" value means "don't care" or accept a "0" or "1" for that bit position. A "1" value means that the incoming bit value "must-match" identically to the corresponding bit in the message identifier.

When a remote frame is sent, an 82527 receiver node will use the Global Mask registers to determine whether the remote frame matches any of its message objects. If the 82527 is programmed to transmit a message in response to a remote frame message identifier, the 82527 will transmit a message with the message identifier of the 82527 message object. The result is the remote message and the responding 82527 transmit message may have different message identifiers because some 82527 global mask register bits are "0".

NOTE:

Please see section 4.9, Acceptance Filtering Implications.

4.8 Global Mask—Extended Register (08–0BH)

08H

7	6	5	4	3	2	1	0
ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
rw	rw	rw	rw	rw	rw	rw	rw

09H

7	6	5	4	3	2	1	0
ID20	ID19	ID18	ID17	ID16	ID15	ID14	ID13
rw	rw	rw	rw	rw	rw	rw	rw

0AH

7	6	5	4	3	2	1	0
ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5
rw	rw	rw	rw	rw	rw	rw	rw

0BH

7	6	5	4	3	2	1	0
ID4	ID3	ID2	ID1	ID0	Reserved		
rw	rw	rw	rw	rw	rw		

Reserved bits read as 000

The default value of the Global Mask - extended after a hardware reset is unchanged.

The Global Mask - extended register applies only to messages using the extended CAN identifier, or message objects with the XTD bit set to "1". 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 the develop an application specific masking strategy. A "0" value means "don't care" on that bit for acceptance filtering. A "1" value means that the 82527 will consider this bit for acceptance filtering.

When a remote frame is sent, an 82527 receiver node will use its Global Mask registers to determine whether the remote frame matches any of its message objects. If the 82527 is programmed to transmit a message in response to a remote frame message identifier, the 82527 will transmit a message with the message identifier of the 82527 message object. The result is the remote message and the responding 82527 transmit message may have different message identifiers because some 82527 Global Mask register bits are "0".

NOTE:

Please see section 4.9, Acceptance Filtering Implications.

4.9 Acceptance Filtering Implications

The 82527 implements two acceptance masks which allow message objects to receive messages with a range of message identifiers (IDs) instead of just a single message ID. This provides the application the flexibility to receive a wide assortment of messages from the bus.

The 82527 observes all messages on the CAN bus and stores any message that matches a message's ID programmed into an "active" message object. It is possible to define which message ID bits must identically match those programmed in the message objects to store the message. Therefore, ID bits of incoming messages are either "must-match" or "don't-care". By selecting bits to be "don't-care", message objects will receive multiple message IDs.

NOTE:
Message objects programmed to transmit are also effected by the Global Masks (standard and extended). The 82527 uses the Global Mask registers to identify which of its message objects transmitted a message. If two 82527 transmit message objects have message IDs that are non-distinct in all "must-match" bit locations, a successful transmission of the higher numbered message object will not be recognized by the 82527. The lower numbered message object will be falsely identified as the transmit message object and its transmit request bit will be reset and its interrupt pending bit set. The actual transmit message object will re-transmit without end because its transmit request bit will not be reset.

This could result in a catastrophic condition since the higher numbered message object may dominate the CAN bus by resending its message without end.

To avoid this condition, applications should require all transmit message objects to use message IDs that are unique with respect to the "must-match" bits. If this is not possible, the application should disable lower numbered message objects with similar message IDs until the higher numbered message object has transmitted successfully.

Another configuration to avoid filtering issues is to dedicate messages 1–14 for transmit and use message 15 for receive. The message 15 mask will have no impact on messages 1–14.

The Acceptance Masks also apply to remote messages. When the 82527 receives a remote message, and a transmit message ID matches after taking into account the global masks, the 82527 will respond by transmitting a data message with its programmed message ID.

4.10 Message 15 Mask Register (0C–0FH)

0CH

7	6	5	4	3	2	1	0
ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
rw	rw	rw	rw	rw	rw	rw	rw

0DH

7	6	5	4	3	2	1	0
ID20	ID19	ID18	ID17	ID16	ID15	ID14	ID13
rw	rw	rw	rw	rw	rw	rw	rw

0EH

7	6	5	4	3	2	1	0
ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5
rw	rw	rw	rw	rw	rw	rw	rw

0FH

7	6	5	4	3	2	1	0
ID4	ID3	ID2	ID1	ID0	Reserved		
rw	rw	rw	rw	r			

Reserved read as 000

The default value of the Message 15 Mask register after a hardware reset is unchanged.

The Message 15 Mask register is a programmable local mask. This feature allows the user to locally mask, or "don't care", any identifier bits of the incoming message for message object 15. Incoming messages are first checked for an acceptance match in message objects 1–14 before passing through to message object 15. Consequently, the Global Mask and the local mask apply to messages received in message object 15.

A "0" value means "don't care" or accept a "0" or "1" for that bit position. A "1" value means that the incoming bit value "must-match" identically to the corresponding bit in the message identifier.

NOTE:
The Message 15 Mask is "ANDed" with the Global Mask. This means that any bit defined as "don't-care" in the Global Mask will automatically be a "don't-care" bit for message 15.

4.11 CLKOUT (Clockout) Register (1FH)

1FH

7	6	5	4	3	2	1	0
0	0	SL1	SL0	CD_V			
r	r	rw	rw	rw			

The CLKOUT register controls the frequency of the CLKOUT signal as well as the slew rate. The default frequency of CLKOUT depends on the CPU interface mode. For Modes 0, 1 and serial mode the default frequency is XTAL. For Modes 2 and 3 the default frequency is XTAL/2. The following are the programmable CLKOUT frequencies and recommended slew rates:

Table 2. Programming CLKOUT and Slew Rates

CDv	CLKOUT Frequency
0	XTAL
1	XTAL/2
10	XTAL/3
11	XTAL/4
100	XTAL/5
101	XTAL/6
110	XTAL/7
111	XTAL/8
1000	XTAL/9
1001	XTAL/10
1010	XTAL/11
1011	XTAL/12
1100	XTAL/13
1101	XTAL/14
1110	XTAL/15
1111	Reserved

SL1	SL0	CLKOUT Conditions
0	0	CLKOUT > 24 MHz
0	1	16 MHz < CLKOUT ≤ 24 MHz
1	0	8 MHz < CLKOUT ≤ 16 MHz
1	1	CLKOUT < 8 MHz

The default value of the CLKOUT register after a hardware reset is 00H (Modes 0 and 1 and serial mode) or 01H (Modes 2 and 3). The slew rate bits enable one to four pullup and pulldown resistors which allows the clockout driver strength to be programmed.

4.12 Bus Configuration Register (2FH)

2FH

7	6	5	4	3	2	1	0
0	CoBy	Pol	0	DcT1	0	DcR1	DcR0
rw	rw	rw	rw	rw	rw	rw	rw

Reserved Bits 7, 4 and 2

one This value must not be programmed by the user.

zero A zero must always be written to this bit.

CoBy Comparator Bypass

one The input comparator is bypassed and the RX0 input is regarded as the valid bus input, (DcR0 must be set to zero).

zero Normal operation: RX0 and RX1 are the inputs to the input comparator, (default after hardware reset).

Pol Polarity

one If the input comparator is bypassed then a logical one is interpreted as dominant and a logical zero is recessive on the RX0 input.

zero If the input comparator is bypassed then a logical one is interpreted as recessive and a logical zero is dominant bit on the RX0 input, (default after hardware reset).

DcT1 Disconnect TX1 output

one Disables the TX1 output driver. This mode is for use with a single wire bus line, or in the case of a differential bus when the two bus lines are shorted together.

zero Enables the TX1 output driver, (default after hardware reset).

DcR1 Disconnect RX1 input

one RX1 is disabled and the RX1 input is disconnected from the inverting comparator input and is replaced by a $V_{CC}/2$ reference voltage.

zero RX1 is enabled and the RX1 input is connected to the inverting input of the input comparator, (default after hardware reset).

DcR0 Disconnect RX0 input

one RX0 is disabled and the RX0 input is disconnected from the non-inverting comparator input and replaced by a $V_{CC}/2$ reference voltage. The MUX bit in the CPU Interface register (02H) must be set to one to activate the $V_{CC}/2$ reference voltage.

zero RX0 is enabled and the RX0 input is connected to the non-inverting input of the input comparator, (default after hardware reset).

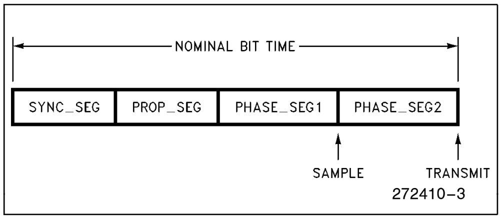
The default value of the bus configuration register after a hardware reset is 00H.

4.13 Bit Timing Overview

A CAN message consists of a series of bits that are transmitted in consecutive bit times. A bit time accounts for propagation delay of the bit, CAN chip input and output delay, and synchronization tolerances. This section describes components of a bit time from the perspective of the CAN Specification and the 82527.

CAN Specification Bit Timing Definitions

The bit timing requirements of the CAN Specification Version 2.0 (September 1991) are defined in section 8. The nominal bit time is composed of four time segments: SYNC__SEG, PROP__SEG, PHASE__SEG1, and a PHASE__SEG2. These time segments are separate and non-overlapping as shown below.



272410–3

SYNC__SEG: This part of the bit time is used to synchronize the various nodes on the bus. An edge is expected to lie within this segment.

PROP__SEG: This part of the bit time is used to compensate for the physical delay times within the network. It is twice the sum of the signal's propagation time on the bus line, the input comparator delay and the output driver delay.

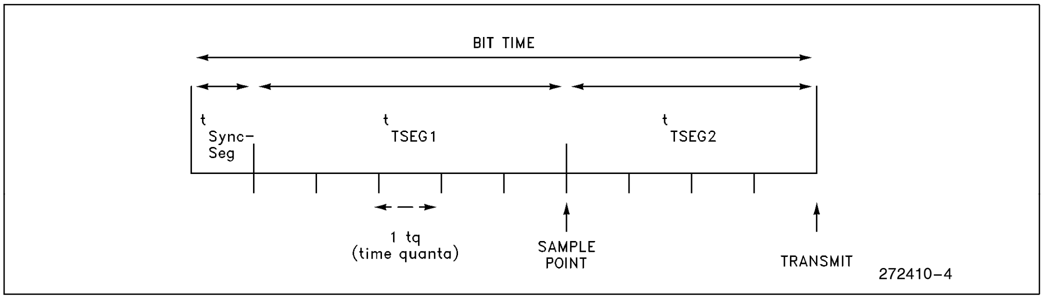
NOTE:
The factor of two accounts for error detection which requires nodes to monitor all bus transmissions.

PHASE__SEG1, PHASE__SEG2: These phases are used to compensate for edge phase errors. These segments can be lengthened or shortened by resynchronization.

SAMPLE POINT: The sample point is the point of time at which the bus level is read and interpreted as the value of that respective bit. Its location is at the end of PHASE__SEG1.

82527 Bit Timing Definitions

The 82527 represents SYNC__SEG, PROP__SEG, PHASE__SEG1, and PHASE__SEG2 by dividing the bit time into three time segments: t_{SYNC_SEG}, t_{TSEG1} and t_{TSEG2}. The "t" prefix indicates the segment is a length of time (i.e. nanoseconds, microseconds). These three time segments are defined by register fields called SYNC__SEG, TSEG1 and TSEG2, respectively. These register fields are digital values programmed into the 82527 Bit Timing Registers.



272410–4

The preceding figure represents a bit time from the perspective of the 82527. A bit time is subdivided into time quanta. One time quantum is derived from the oscillator (SCLK) and the baud rate prescaler (BRP). The length of the bit time results from the addition of the programmable segments: SYNC__SEG, TSEG1 and TSEG2.

TIME QUANTUM (tq): A fixed unit of time derived from the system clock period (t_{SCLK}) equal to t_{SCLK} x (baud rate prescaler + 1).

$t_{SYNC\_\_SEG}$: Synchronizes the various nodes on the bus and an edge from the transmitter is expected to lie within this segment. The SYNC__SEG is 1 time quantum.

t_{TSEG1}: The sum of PROP__SEG and the PHASE__SEG1 as defined by the CAN Specification. TSEG1 is a value programmed into the 82527 CAN device to specify t_{TSEG1}. In three sample mode, 2 time quanta must be added to t_{TSEG1}. This allows the 82527 to sample the bit two additional times prior to sample point at the end of TSEG1.

t_{TSEG2}: This time segment is equivalent to PHASE__SEG2 as defined by the CAN Specification.

Bit Timing Relationships

The following are relationships of the 82527 bit timing:

1. bittime = $t_{SYNC\_\_SEG} + t_{TSEG1} + t_{TSEG2}$
   (see preceding figure)
2. tq = $t_{SCLK} \times$ (BRP + 1) where t_{SCLK} is the period of the system clock.
3. $t_{SYNC\_\_SEG}$ = 1 tq
4. t_{TSEG1} = (TSEG1 + 1) × tq
   (the 82527 adds one to TSEG1 in hardware)
5. TSEG1 = [2 . . . 15]
   (field in Bit Timing Register 1)
6. t_{TSEG2} = (TSEG2 + 1) × tq
   (the 82527 adds one to TSEG2 in hardware)
7. TSEG2 = [1 . . . 7]
   (field in Bit Timing Register 1)
8. t_{SJW} = (SJW + 1) × tq
   (the 82527 adds one to SJW in hardware)
9. SJW = [0 . . . 3]
   (field in Bit Timing Register 0)
10. t_{prop} = two times the maximum of the sum of the delay of the physical bus delay, the output driver delay and the input comparator delay rounded up to the nearest multiple of tq.
11. The maximum oscillator tolerance equals
    [3.84 x $t_{SJW}/t_{bittime}$].

The following conditions must be met to maintain synchronization.

1. $t_{TSEG2} \geq 2tq$
   (minimum tolerance for resynchronization)
2. $t_{TSEG2} \geq t_{SJW}$
   (If $t_{SJW} > t_{TSEG2}$, sampling may occur after the bit time)
3. $t_{TSEG1} \geq 3tq$
   (minimum tolerance for resynchronization with 1tq propagation delay allowance)
4. $t_{TSEG1} \geq t_{SJW} + t_{prop}$
5. For three sample mode (SPL bit = 1, register 4Fh),
   $t_{TSEG1} \geq t_{SJW} + t_{prop} + 2tq$

4.14 Bit Timing Registers (3FH, 4FH)

Bit timing registers are used to define the CAN bus frequency, the sample point within a bit time, and the mode of synchronization.

Bit Timing Register 0 (3FH)

3FH

7	6	5	4	3	2	1	0
SJW		BRP					
rw		rw					

SJW (Re)Synchronization Jump Width

The valid programmed values are 0–3. The SJW defines the maximum number of time quanta a bit time may be shortened or lengthened by one resynchronization. The actual interpretation of this value by the hardware is to use one more than the programmed value.

BRP Baud Rate Prescaler

The valid programmed values are 0–63. The baud rate prescaler programs the length of one time quantum as follows:

tq = $t_{SCLK} \times$ (BRP + 1) where t_{SCLK} is the period of the system clock (SCLK).

The default value of the bit timing register 0 after a hardware reset is unchanged.

Bit Timing Register 1 (4FH)

4FH

7	6	5	4	3	2	1	0
Spl	TSEG2			TSEG1			
rw	rw			rw			

Spl Sampling Mode

Sampling mode = zero may result in faster bit transmissions rates, while sampling mode = one is more immune to noise spikes on the CAN bus.

one Three samples are used for determining the valid bit value using majority logic. The CAN bus is sampled three times per bit time.

zero One sample is used for determining the valid bit value. The CAN bus is sampled once per bit time.

TSEG1 Time Segment 1

The valid programmed values are 2–15. TSEG1 is the time segment before the sample point. The actual interpretation of this value by the hardware is one more than the value programmed by the user.

TSEG2 Time Segment 2

The valid programmed values are 1–7. TSEG2 is the time segment after the sample point. The actual interpretation of this value by the hardware is one more than the value programmed by the user.

NOTE:

In order to achieve correct operation according to the CAN protocol, the total bit length should be a minimum of 8tq with (TSEG1 + TSEG2 ≥ 5).

The default value of the bit timing register 1 after a hardware reset is unchanged.

4.15 Comparison of 82526 and 82527 Bit Timings Calculations

82527

The 82527 timings calculations differ from the 82526 since the 82527 timing equation implicitly accounts for synchronization jump widths.

CAN bus frequency = XTAL frequency/[(DSC + 1) x (BRP + 1) x (3 + TSEG1 + TSEG2)] where the DSC bit is found in the CPU interface register, location 02H.

Example: Reg 02H = 41H, Reg 3FH = 4AH, Reg 4FH = 25H

with resulting bit timing parameters:

BRP = 10	DSC = 1
SJW = 2	XTAL = 16 MHz
TSEG1 = 5	TSEG2 = 2

CAN bus frequency = 16 MHz/[(1 + 1) x (10 + 1) x (3 + 5 + 2)] = 72,727 bits/seconds

82526

CAN bus frequency = XTAL frequency/[(BRP + 1) x 2 x (5 + TSEG1 + TSEG2 + 2 x SJW)]

Example: Reg 03H = C5H, Reg 04H = 25H

with resulting bit timing parameters

INSYNC = 1	BRP = 5
SJW = 3	XTAL = 16 MHz
TSEG1 = 5	TSEG2 = 2

CAN bus frequency = 16 MHz/[(5 + 1) x 2 x (5 + 5 + 2 + 2 x 3)] = 74,074 bits/seconds

4.16 Interrupt Register (5FH)

5FH

7	6	5	4	3	2	1	0
IntId							
r							

IntId Interrupt Identifier

The interrupt register is a read-only register. The value in this register indicates the source of the interrupt. When no interrupt is pending, this register holds the value "0". If the SIE bit in the Control Register (00H) is set and the 82527 has updated the Status Register, the interrupt register will contain a "1". This indicates an interrupt is pending due to a change in the status register. The value 2 + message number indicates the IntPnd bit in the corresponding message object is set. There is an exception in that message object 15 will have the value 2, giving message object 15 the highest priority of all message objects. The default value of the interrupt register after a hardware reset is undefined.

Interrupt	Register Value (hex)
none	0
Status Register	1
message object 15	2
message object 1	3
message object 2	4
message object 3	5
message object 4	6
message object 5	7
message object 6	8
message object 7	9
message object 8	AH
message object 9	BH
message object 10	CH
message object 11	DH
message object 12	EH
message object 13	FH
message object 14	10H

For example, a message is received by message object 13 with the IE (Control register) and RXIE (message object 13 Control 0 register) bits set. The interrupt pin will be pulled low and the value 15 (0FH) will be placed in the interrupt register.

If the value of register 5FH equals "1", then the status register at location 01H must be read to update this interrupt register. The status change interrupt (SIE bit in register 00H) has higher priority than interrupts from message objects. Register 5FH is automatically set to "0" or to the lowest value corresponding to a message with IntPnd set. When the value of this register is two or more, the IntPnd bit of the corresponding message object control register is set.

The 82527 will respond to each status change event independently and will not bundle interrupt events in a single interrupt signal. However, if two status change events occur before the first is acknowledged by the CPU, the next event will not generate a separate interrupt output. Therefore, when servicing status change interrupts, the user code should check all useful status bits upon each status change interrupt.

After resetting the INTPND bit in the Control 0 Register of individual message objects, the minimum delay of the 82527 resetting the interrupt pin and updating the Interrupt Register (5FH) is 3 MCLK cycles and a maximum of 14 MCLK cycles (after the CPU write operation to this register is finished). When a status change interrupt occurs, reading the Status Register (01H) will reset the interrupt pin in a maximum of 4 MCLK cycles + 145 ns. Clearing the INTPND bit of the message object will de-activate the INT# pin.

4.17 Serial Reset Address (FFH)

FFH

7	6	5	4	3	2	1	0
Serial Reset Address							

w

The serial reset address is used to synchronize accesses between the 82527 and the CPU when the CPU cannot provide a chip select. The CPU must write a string of 16 "FFH" bytes to achieve synchronization.

The default value of the serial reset address after a hardware reset is undefined.

4.18 82527 Message Objects

The message object is the means of communication between the host microcontoller and the CAN controller in the 82527. Message objects are configured to transmit or receive messages.

There are 15 message objects located at fixed addresses in the 82527. Each message object starts at a base address that is a multiple of 16 bytes and uses 15 consecutive bytes. For example, message object 1 starts at address 10H and ends at address 1EH. The remaining byte in the 16 byte field is used for other 82527 functions. In the above example the byte at address 1FH is used for the clockout register.

Message object 15 is a receive-only message object that uses a local mask called the message 15 mask register. This mask allows a large number of infrequent messages to be received by the 82527. In addition, message object 15 is buffered to allow the CPU more time to receive messages.

Message Object Structure

Base Address + 0	Control 0
+ 1	Control 1
+ 2	Arbitration 0
+ 3	Arbitration 1
+ 4	Arbitration 2
+ 5	Arbitration 3
+ 6	Mess. Conf.
+ 7	Data 0
+ 8	Data 1
+ 9	Data 2
+ 10	Data 3
+ 11	Data 4
+ 12	Data 5
+ 13	Data 6
+ 14	Data 7

4.19 Control 0 and Control 1 Registers
(Base Address + 0, Base Address + 1)

Control 0 Register

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">Base Address + 0</td></tr>
<tr><td colspan="2">MsgVal</td><td colspan="2">TXIE</td><td colspan="2">RXIE</td><td colspan="2">IntPnd</td></tr>
<tr><td colspan="2">rw</td><td colspan="2">rw</td><td colspan="2">rw</td><td colspan="2">rw</td></tr>
</table>

Control 1 Register

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">Base Address + 1</td></tr>
<tr><td colspan="2">RmtPnd</td><td colspan="2">TxRqst</td><td colspan="2">MsgLst
CPUUpd</td><td colspan="2">NewDat</td></tr>
<tr><td colspan="2">rw</td><td colspan="2">rw</td><td colspan="2">rw</td><td colspan="2">rw</td></tr>
</table>

Each bit in the Control 0 and Control 1 bytes occurs twice; once in true form and once in complement form. This bit representation makes testing and setting these bits as efficient as possible. The advantage of this bit representation is to allow write access to single bits of the byte, leaving the other bits unchanged without the need to perform a read/modify/write cycle. The representation of these two bits is described below:

	MSB	LSB	Meaning
Write	0	0	Not allowed (indeterminate)
	0	1	reset
	1	0	set
	1	1	unchanged
Read	0	1	reset
	1	0	set

These bit pairs eliminate the need to execute a "read-modify-write" operation used to set or reset a bit. The bit pairs allow the software to set or reset any bit without disrupting the other bits using a single write operation.

For example, a CPU would set the TxRqst bit of the Control 1 byte with the following instructions:

```
LDB  Dummy, #0EFH   ;load 11101111 into
                    ;accumulator
                    ;register
STB  Dummy, CTR1    ;write 11101111 to
                    ;Control 1,
                    ;setting TXRqst
```

MsgVal Message Valid

The MsgVal bit is an individual halt bit for each message object. While this bit is reset the 82527 will not access this message object for any reason.

one The message object is valid.

zero The message object is invalid.

The Message Valid bit is set to indicate the message object is configured and is ready for communications transactions. This bit may be reset at any time if the message is no longer required, or if the identifier is being changed. If a message identifier is changed, the message object must be made invalid first, and it is not necessary to reset the chip following this modification. The CPU must reset the MsgVal bit of all unused messages during initialization of the 82527 before the Init bit of the Control Register (00H) is reset. The contents of message objects may be reconfigured dynamically during operation and the MsgVal bit assists reconfiguration in many cases. This bit is written by the CPU.

Two or more message objects may not have the same message identifier and also be valid at the same time.

TXIE Transmit Interrupt Enable

one An interrupt will be generated after a successful transmission of a frame.

zero No interrupt will be generated after a successful transmission of a frame.

The Transmit Interrupt Enable bit enables the 82527 to initiate an interrupt after the successful transmission by the corresponding message object. This bit is written by the CPU.

RXIE Receive Interrupt Enable

one An interrupt will be generated after a successful reception of a frame.

zero No interrupt will be generated after a successful reception of a frame.

This bit enables the 82527 to initiate an interrupt after the successful reception by the corresponding message object. This bit is written by the CPU.

NOTE:

In order for TXIE or RXIE to generate an interrupt, IE in the Control Register must be set.

IntPnd Interrupt Pending

one This message object has generated an interrupt.

zero No interrupt was generated by this message object since the last time the CPU cleared this bit.

This bit is set by the 82527 following a successful transmission or reception as controlled by the RXIE and TXIE bits. The CPU must clear this bit when servicing the interrupt.

RmtPnd Remote Frame Pending

one The transmission of this message object has been requested by a remote node and is not yet done.

zero There is no waiting remote request for the message object.

This bit is only used by message objects with direction = transmit. This bit is set by the 82527 after receiving a remote frame which matches its message identifier, taking into account the global mask register. The corresponding message object will respond by transmitting a message, if the CPUUpd bit = zero. Following this transmission, the 82527 will clear the RmtPnd bit. In other words, when this bit is set it indicates a remote node has requested data and this request is still pending because the data has not yet been transmitted.

NOTE:

Setting RmtPnd will not cause a remote frame to be transmitted. The TxRqst bit is used to send a remote frame from a receive message object.

TxRqst Transmit Request

one The transmission of this message object has been requested and has not been completed.

zero This message object is not waiting to be transmitted.

This bit is set by the CPU to indicate the message object data should be transmitted.

Conditions required to transmit a data frame:

1) Init bit = 0
2) MsgVal bit = 1
3) direction = transmit
4) NewDat bit = 1
5) TxRqst = 1

If direction = receive, (Init = 0 and MsgVal = 1) then a remote frame is sent to request a remote node to send the corresponding data. TxRqst is also set by the 82527 (at the same time as RmtPnd in message objects whose direction = transmit) when it receives a remote frame from another node requesting this data. This bit is cleared by the 82527 along with RmtPnd when the message has been successfully transmitted, if the NewDat bit has not been set.

NOTE:

Setting TxRqst will send a data frame for a transmit message object and a remote frame for a receive message object.

MsgLst Message Lost

This definition is only valid for message objects with direction = receive. For message objects with direction = transmit, the definition is replaced by CPUUpd.

one The 82527 has stored a new message in this message object when NewDat was still set.

zero No message was lost since the last time this bit was reset by the CPU.

This bit is used to signal that the 82527 stored a new message into this message object when the NewDat bit was still set. Therefore, this bit is set if the CPU did not process the contents of this message object since the last time the 82527 set the NewDat bit; this indicates the last message received by this message object overwrote the previous message which was not read and is lost.

CPUUpd CPU Updating

Only valid for message objects with direction = transmit. For message objects with direction = receive it is replaced by MsgLst.

one This message object may not be transmitted.

zero This message object may be transmitted, if direction = transmit.

The CPU sets this bit to indicate it is updating the data contents of the message object and the message should not be transmitted until this bit has been reset. The CPU indicates message updating has been completed by resetting this bit (it is not necessary to use the MsgVal bit to update the message object's data contents). The purpose of this bit is to prevent a remote frame from triggering a transmission of invalid data.

NewDat New Data (This bit has different meanings for receive and transmit message objects.)

one The 82527 or CPU has written new data into the data section of this message object.

zero No new data has been written into the data section of this message object since the last time this bit was cleared by the CPU.

For message objects with direction = receive, the 82527 sets this bit whenever new data has been written into the message object.

For message objects 1–14 please note: When new data is written into message objects, the unused data bytes will be overwritten with non-specified values. The CPU should clear this bit before reading the received data and then check if the bit remained cleared when all bytes have been read. If the NewDat bit is set, the CPU should re-read the received data to prevent working with a combination of old and new data.

For message object 15, new data is written into the shadow register. The foreground register is not over-written with new data. For message object 15 messages, the data should be read first, the IntPnd reset, and then the NewDat and RmtPnd bits are reset. Resetting the NewDat and RmtPnd bits before resetting the IntPnd bit will result in the interrupt line remaining active.

For message objects with direction = transmit, the CPU must set this bit to indicate it has updated the message contents. This is done at the same time the CPU clears the CPUUpd bit. This will ensure that if the message is actually being transmitted during the time the message was being updated by the CPU, the 82527 will not reset the TxRqst bit. In this way, the TxRqst bit is reset only after the actual data has been transferred.

Each bit in the Control 0 and Control 1 registers may be set and reset by the CPU as required.

The default values of the Control 0 and Control 1 registers after a hardware reset are unchanged.

The control registers have been configured with two bits per function to allow software to reduce costly read/modify/write operations. It is possible to modify bits individually by using only write operations.

To program a transfer request, the Control 1 register of the message object should have the TxRqst and NewDat bits set to "1". Therefore, this register may be written with the value 066H to initiate a transmission.

A remote frame may be received, an interrupt flag set, and no transmit sent in response by configuring a message object in the following manner. Set the CPUUpd and RXIE bits in the message object control register to "1". Set the Dir bit in the message configuration register to "1". A remote frame will be received by this message object, the IntPnd bit will be set to "1" and no transmit message will be sent.

Message Object Priority

If multiple message objects are waiting to transmit, the 82527 will first transmit the message from the lowest numbered message object, regardless of message identifier priority.

If two message objects are capable of receiving the same message (possibly due to message filtering strategies), the message will be received by the lowest numbered message object. For example, if all acceptance mask bits were set as "don't care", message object 1 will receive all messages.

4.20 Arbitration 0, 1, 2, 3 Registers (Base Address + 2–Base Address + 5)

Arbitration 0 Base + 2

7	6	5	4	3	2	1	0
ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
rw	rw	rw	rw	rw	rw	rw	rw

Arbitration 1 Base + 3

ID20	ID19	ID18	ID17	ID16	ID15	ID14	ID13
rw	rw	rw	rw	rw	rw	rw	rw

Arbitration 2 Base + 4

ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5
rw	rw	rw	rw	rw	rw	rw	rw

Arbitration 3 Base + 5

ID4	ID3	ID2	ID1	ID0	Reserved
rw	rw	rw	rw	rw	r

Reserved read as 000

ID0–ID28 Message Identifier

ID0–ID28 is the identifier for an extended frame.

ID18–ID28 is the identifier for a standard frame.

NOTE:

When the 82527 receives a message, the entire message identifier, the data length code (DLC) and the Direction bit are stored into the corresponding message object.

4.21 Message Configuration Register (Base Address + 6)

Base + 6

7	6	5	4	3	2	1	0
DLC				Dir	Xtd	Reserved	
rw				rw	rw	r	

DLC Data Length Code

The valid programmed values are 0–8. The data length code of a message object is written with the value corresponding to the data length.

Dir Direction

one Direction = transmit. When TXRqst is set, the message object will be transmitted.

zero Direction = receive. When TXRqst is set, a remote frame will be transmitted. When a message is received with a matching identifier, the message will be stored in the message object.

Xtd Extended or standard identifier

one This message object will use an extended 29-bit message identifier.

zero This message object will use a standard 11-bit message identifier.

The default value of the message configuration register after a hardware reset is unchanged.

If an extended frame message identifier is used (arbitration bits 0–17) and the message configuration register Xtd bit is "0" to specify a standard frame, the 82527 will reset the extended bits in the arbitration registers to "0".

An extended receive message object (XTD = "1") will not receive standard messages.

If a message object receives a message from the bus, the entire message identifier will be stored in the message object. Therefore, if acceptance filtering (masking register) is used, the masked-off "don't care" bits will be rewritten corresponding to the message ID of the incoming message.

4.22 Data Bytes (Base Address + 7–Base Address + 14)

When the 82527 stores a message all 8 data bytes will be written into the message object.

The default value of the data bytes 0–7 after a hardware reset is unchanged. The values of unused data bytes are random and change during operation.

4.23 Special Treatment of Message Object 15

Message object 15 is a receive-only message object with a programmable local mask called the Message 15 Mask Register. Since this message object is a receive-only message object, the TXRqst bit and the TXIE have been hardwired inactive and the CPUUpd bit has no meaning.

The incoming messages for message object 15 will be written into a two-message alternating buffers to avoid the loss of a message if a second message is received before the CPU has read the first message. Once message object 15 is read, it is necessary to reset the NewDat and the RmtPnd bits to allow the CPU to read the shadow message buffer which will receive the next message or which may already contain a new message.

If two messages have been received by message object 15, the first will be accessible to the CPU. The alternate buffer will be overwritten if a subsequent (third receive) receive message is received. Once again, after reading message 15, the user program should reset the IntPnd bit followed by a reset of the NewDat and RmtPnd bits in the Control 1 Register.

The Xtd bit in the message configuration register determines whether a standard or an extended frame will be received by this message object.

5.0 PORT REGISTERS

PORT 1 Registers

P1CONF (9FH)

9FH

7	6	5	4	3	2	1	0
P1CONF 0–7							
rw							

P1CONF 0–7

Port 1 Input/Output Configuration bits

one Port pin configured as a push-pull output.

zero Port pin configured as a high-impedance input.

The default value of the P1CONF register after a hardware reset is 00H.

P1IN (BFH)

BFH

7	6	5	4	3	2	1	0
P1IN 0–7							
rw							

P1IN 0–7

Port 1 Data In

one A one (high voltage) is read from the pin.

zero A zero (low voltage) is read from the pin.

The default value of the P1IN register after a hardware reset is FFH.

P1OUT (DFH)

DFH

7	6	5	4	3	2	1	0
P1OUT 0–7							
rw							

P1OUT 0–7

Port 1 Data Out

one A logical one (high voltage) is written to the pin.

zero A logical zero (low voltage) is written to the pin.

The default value of the P1OUT register after a hardware reset is 00H.

PORT 2 Registers

P2CONF (AFH)

AFH

7	6	5	4	3	2	1	0
P2CONF 0–7							
rw							

P2CONF 0–7

Port 2 Input/Output Configuration bits

one Port pin configured as a push-pull output.

zero Port pin configured as a high-impedance input.

The default value of the P2CONF register after a hardware reset is 00H.

P2IN (CFH)

CFH

7	6	5	4	3	2	1	0
P2IN 0–7							
rw							

P2IN 0–7

Port 2 Data In

one A one (high voltage) is read from the pin.

zero A zero (low voltage) is read from the pin.

The default value of the P2IN register after a hardware reset is FFH.

P2 OUT (EFH)

EFH

7	6	5	4	3	2	1	0
P2OUT 0–7							
rw							

P2OUT 0–7

Port 2 Data Out

one A logical one (high voltage) is written to the pin.

zero A logical zero (low voltage) is written to the pin.

The default value of the P2OUT register after a hardware reset is 00H.

6.0 SERIAL RESET ADDRESS (FFH)

FFH

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="8">Serial Reset Address</td></tr>
<tr><td colspan="8">w</td></tr>
</table>

The serial reset address is used to synchronize accesses between the 82527 and the CPU when the CPU cannot provide a chip select. The CPU must write a string of 16 "FFH" bytes to achieve synchronization.

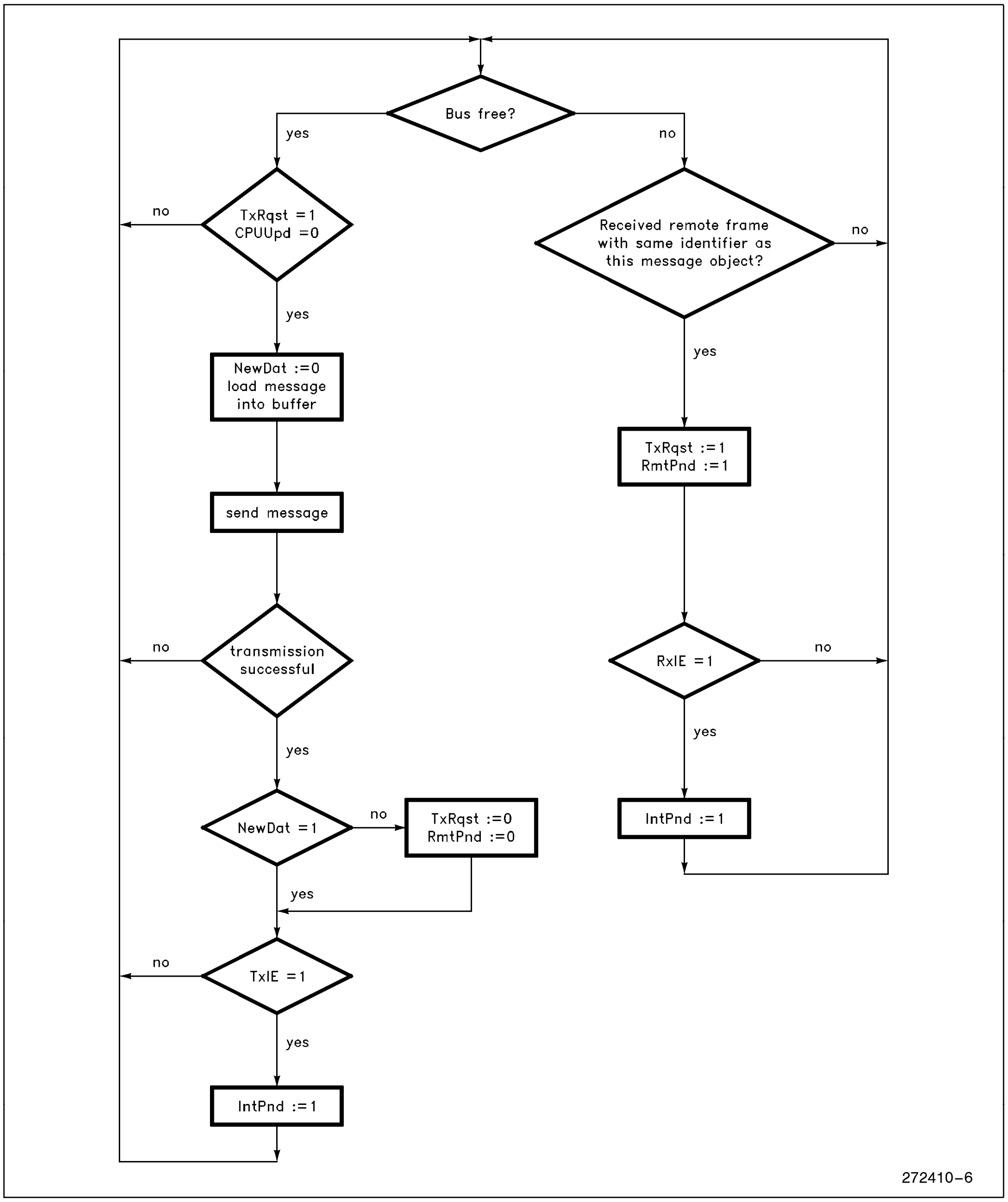
The default value of the serial reset address after a hardware reset is undefined.

7.0 PROGRAM FLOWS

The following flowcharts describe the operation of the 82527 and suggested flows for the host-CPU.

7.1 82527 Handling of Message Objects 1–14 (Direction = Transmit)

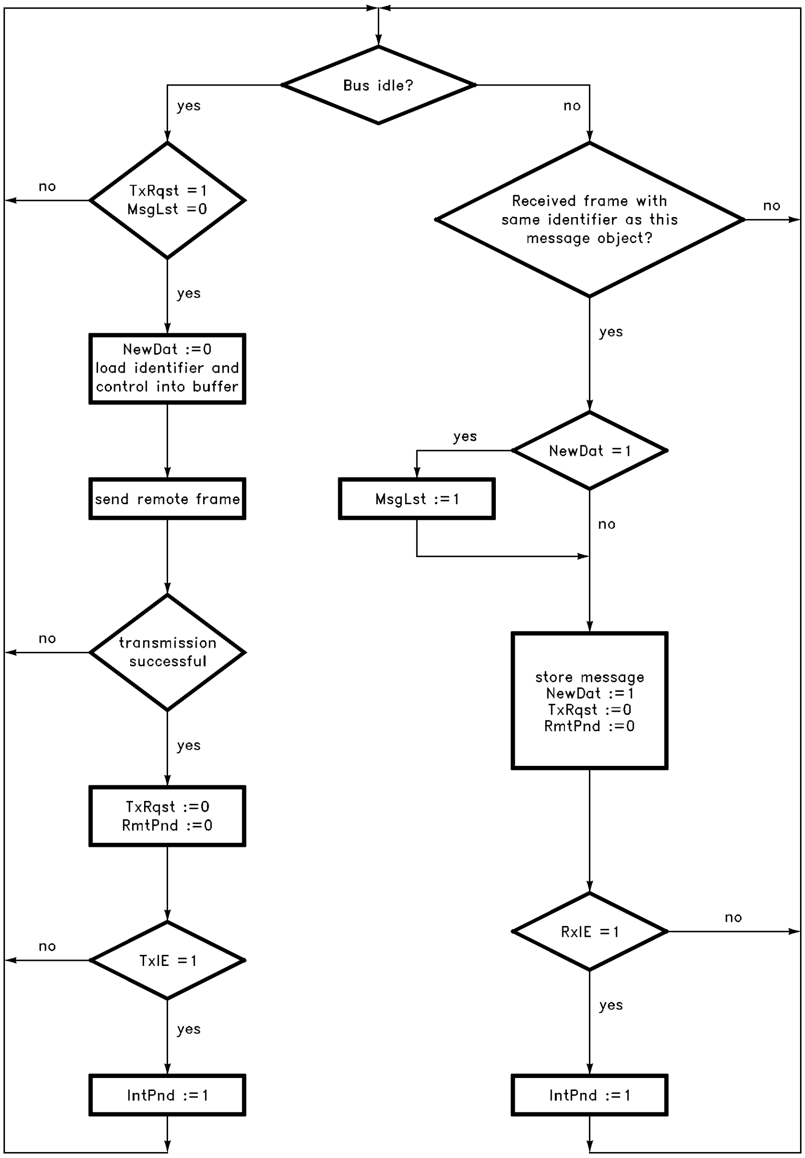
These are the operations the 82527 executes to transmit messages. This diagram is useful to identify when the 82527 sets bits in the control registers.



272410–6

7.2 82527 Handling of Message Objects 1–14 (Direction = Receive)

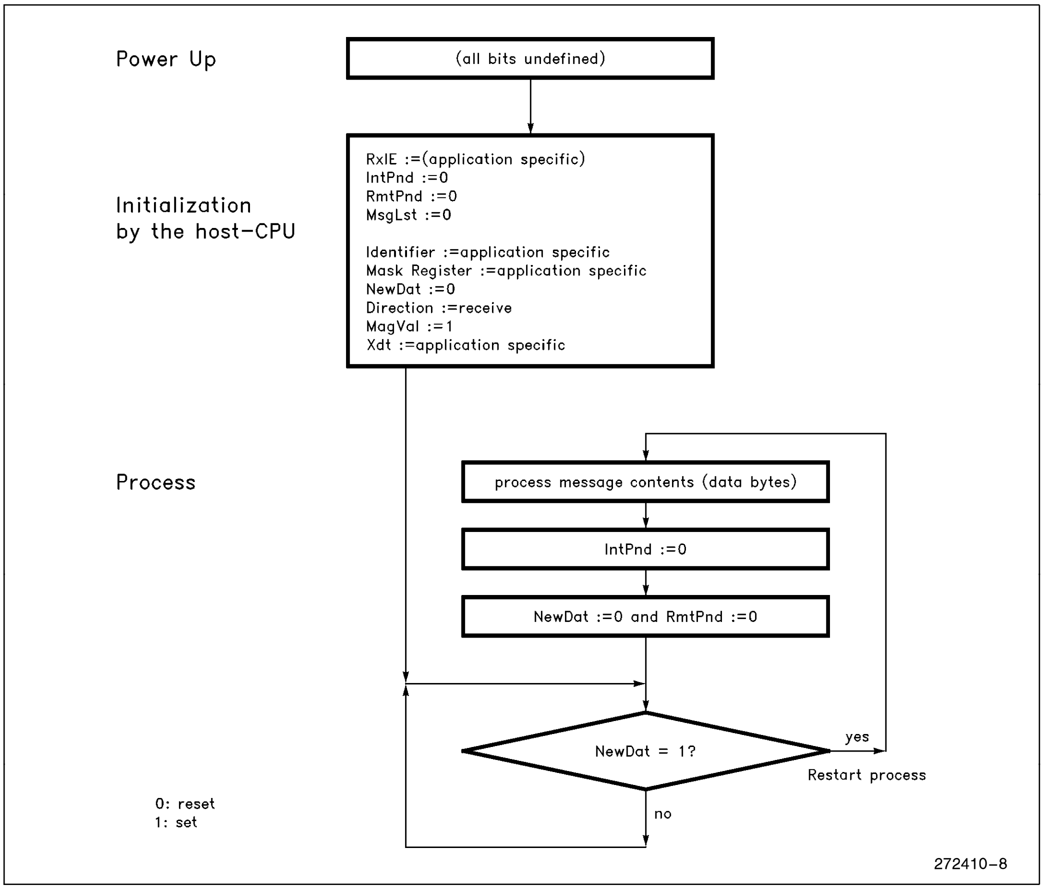
These are the operations the 82527 executes to receive messages. This diagram is useful to identify when the 82527 sets bits in the control registers.



272410–7

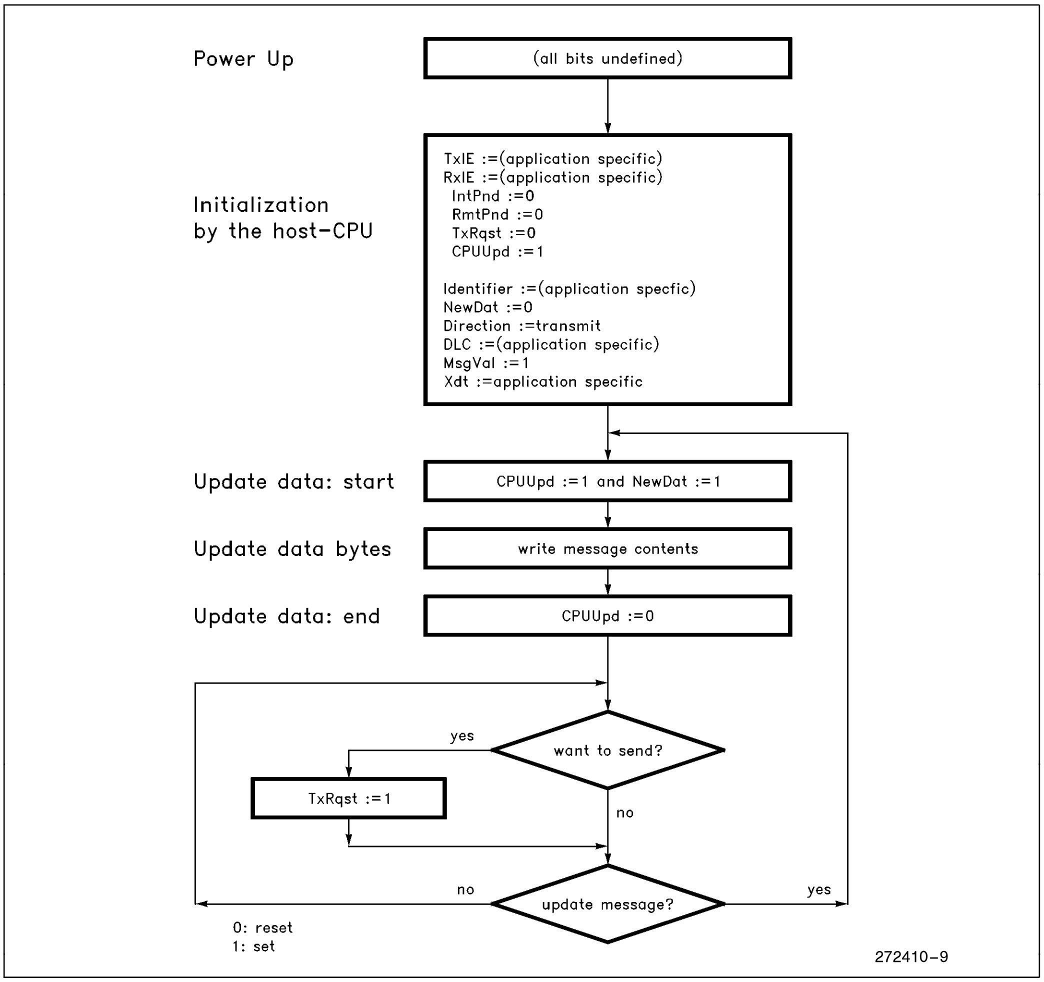
7.3 Host-CPU Handling of Message Object 15 (Direction = Receive)

These are the operations the host-CPU executes to receive message object 15.



7.4 Host-CPU Handling of Message Objects 1–14 (Direction = Transmit)

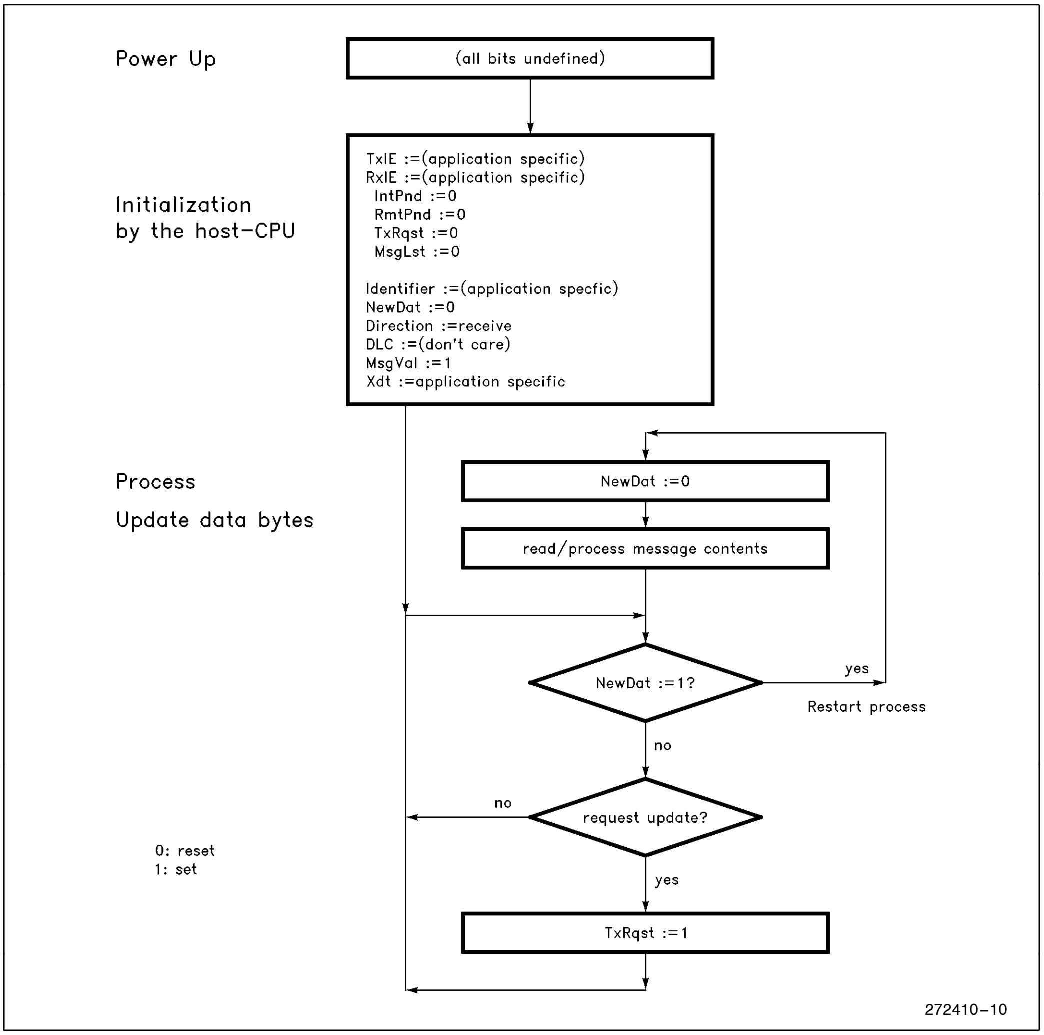
These are the operations the host-CPU executes to transmit message objects 1–14.



272410–9

7.5 Host-CPU Handling of Message Objects 1–14 (Direction = Receive)

These are the operations the host-CPU executes to receive message objects 1–14.



8.0 CPU INTERFACE LOGIC

The CIL (CPU Interface Logic) is a flexible interface between the CPU and the 82527 RAM. The CIL allows a direct serial interface or parallel interface connection to the 82527 for most commonly used CPUs.

The CIL converts address/data/control signals from the CPU to the internal memory bus. The internal memory bus is a 16-bit non-multiplexed bus that is used by both the CPU and the CAN Controller to read and write to the RAM.

CPU Interface Description

There are five CPU interface modes used to interface a CPU to the 82527. These include four parallel interface modes and one serial interface mode.

Two mode pins (MODE0, MODE1) select one of the following parallel interface modes:

Mode1	Mode0	Interface Mode
0	0	Mode 0: 8-bit multiplexed—Intel architecture
0	1	Mode 1: 16-bit multiplexed—Intel architecture
1	0	Mode 2: 8-bit multiplexed—non-Intel architecture
1	1	Mode 3: 8-bit non-multiplexed—non-Intel architecture

The serial interface mode is entered by selecting parallel interface Mode 0 (MODE0 = 0, MODE1 = 0) and connecting RD# and WR# to V_{SS}.

The state of MODE0 and MODE1 as well as RD# and WR# are latched on the rising edge of RESET#. In an application only one of the five CPU interface modes may be entered. Since the CPU interface mode is latched on the rising edge of RESET# it is necessary to enter a hardware reset (RESET# = 0) in order to change CPU interface modes.

Parallel Interfacing Techniques

Mode 0 is intended to interface to Intel architectures (ALE, RD#, WR#) using an 8-bit multiplexed address/data bus. A READY output is provided to force wait states in the CPU.

Mode 1 is intended to interface to Intel architectures (ALE, RD#, WR#) using a 16-bit multiplexed address/data bus. A READY output is provided to force wait states in the CPU.

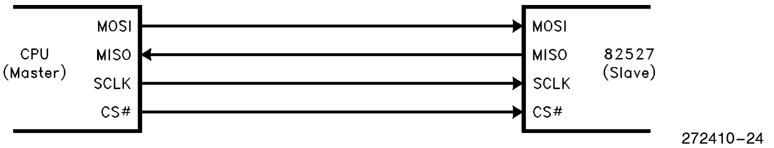
Mode 2 is intended to interface to non-Intel architectures (AS, E, R/W#) using an 8-bit multiplexed address/data bus.

Mode 3 is intended to interface to non-Intel architectures using an 8-bit non-multiplexed address/data bus. The asynchronous mode uses R/W#, CS#, and DSACK0# (E = 1). The synchronous mode uses R/W#, CS#, and E. Mode 3 uses the address/data bus as the address bus and Port 1 as the data bus.

For CPUs which do not provide a READY or DSACK0# input and do not meet the address/data bus timing restrictions, a double read mechanism must be used. When writing to the 82527 the programmer must ensure that the time between two consecutive write accesses is not less than two memory clock (MCLK) cycles. When reading the 82527, a double read is programmed. The first read will be to the message object memory address and the second read will be to the High Speed Read register (04H, 05H). After the first CPU read access, the 82527 stores the data contents to the High Speed Read register for the second read.

Serial Interface Techniques

The serial interface on 82527 is fully compatible to the SPI protocol of Motorola and will interface to most commonly used serial interfaces. The serial interface is implemented in slave mode only, and responds to the master using the specially designed serial interface protocol. This serial interface allows an interconnection of several CPU's and peripherals on the same circuit board.



272410–24

MOSI: Master Out Slave In

The MOSI pin is the data output of the master (CPU) device and the data input of the slave (82527) device. Data is transferred serially from the master to the slave on the signal line, with the most significant bit first and least significant bit last.

MISO: Master In Slave Out

The MISO pin is the data output of the slave (82527) device and the input of the master (CPU) device.

CS#: Chip Select (used as Slave Select for the SPI interface)

An asserted state on the slave select input (CS#) enables the 82527 to accept data on the MOSI pin. The CS# must not toggle between each transmitted byte. The 82527 will only drive data to the serial data register if this pin is asserted.

SCLK: Serial Clock

The master device provides the serial clock for the slave device. Data is transferred synchronously to this clock in both directions. The master and the slave devices exchange a data byte during a sequence of eight clock pulses.

Serial Interface Protocol

The general format of the data exchange from the 82527 to the master is a bit-for-bit exchange on each SCLK clock pulse. Data is read on the rising edge of the SCLK, and is changed on the falling edge of SCLK.

Data is arrranged in the 82527 such that the significance of a bit is determined by its position from the start for output and from the end for input, most significant bit is sent first. The order is such that bit exchanges in multiples of 8 bits and up to 15 bytes of data are allowed. A maximum of 17 bytes can be sent to the 82527-SPI including one address byte, one SPI Control Byte and 15 data bytes.

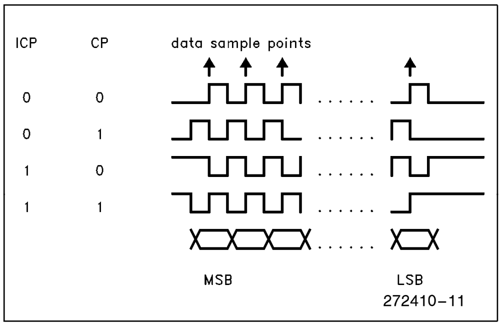
At the beginning of a transmission over the serial interface, the first byte will be the address of the 82527 special function register or the 82527 RAM to be accessed. The next byte transmitted is a Control Byte, which contains the number of bytes to be transmitted and whether this is to be a read or write access to the 82527. The first two bytes are followed by the data bytes (1 to 15).

To ensure the 82527 device is not out of synchronization, the 82527 will transmit the values "AAH" and then "55H" through the MISO pin while the master transmits the Address and Control Byte. This can be enabled and disabled depending on the state of the AD3 pin. This allows the master to know whether the 82527 is synchronized.

If the 82527 is out of sync, the master SPI device can re-sync by transmitting a string of 16 FFH bytes. When the SPI receives a command byte with the value FFH, it will assume the next byte is an address. If it receives an address of FFH (the SPI Reset Address), it will assume that the next byte is also an address.

The states of the pins AD0–AD3 are sampled on the rising edge of RESET#. They have the following functions:

Pin	Function
AD0 (ICP)	Idle Clock Polarity zero SCLK is idle low. one SCLK is idle high.
AD1 (CP)	Clock Phase zero Data is sampled on the rising edge of SCLK (ICP = 0) or data is sampled on the falling edge of SCLK (ICP = 1). one Data sampled on the falling edge of SCLK (ICP = 0) or data is sampled on the rising edge of SCLK (ICP = 1).
AD2 (CSAS)	Chip select active state zero Asserted state of CS# is logic low. one Asserted state of CS# is logic high.
AD3 (STE)	Synchronization Transmission Enable Enables the transmission of the synchronization bytes, while the Address and Control Bytes are transferred. zero The first two bytes which will be sent to the CPU after CS# is asserted are 00H and 00H. one The first two bytes which will be sent to the CPU after CS# is asserted are AAH and 55H.



8.1 Serial Control Byte

The Serial Control Byte is transmitted by the CPU to the 82527 as follows:

7	6	5	4	3	2	1	0
Dir	0	0	0	Serial data length code			

Dir Serial transmission direction

zero The data bytes or SPI Configuration Register will be read, so the 82527 will transfer information to the CPU.

one The SPI Configuration Register or the data bytes will be sent from the CPU to the 82527.

SDLC Serial Data Length Code

The first data byte (third byte of the SPI protocol) will be written to or read from the 82527 address (first byte of the SPI protocol). After this, the address is incremented by the SPI logic and the next data byte is written or read from this address. In one data stream, a maximum of 15 data bytes can be transferred. A DLC of zero is not allowed. After a DLC of zero is received, the SPI must be resynchronized.

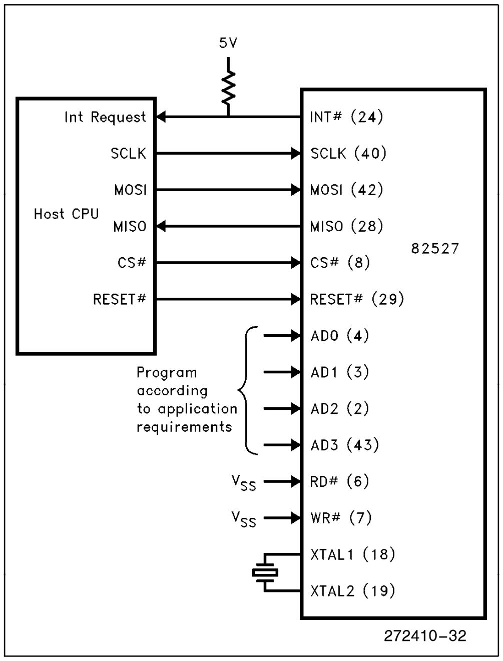
The serial interface is configured from the states of AD0-AD3 on the rising edge of RESET#.

When the CPU conducts a READ, the CPU sends an address byte and a serial ontrol byte. When the 82527 responds back with data, the 82527 ignores the MOSI pin (transmission from the CPU). The CPU may transmit an address and serial control byte after CS# is de-activated and then re-activated. This means the chip select should be activated and de-activated for each read or write transmission.

Synchronization bytes must be monitored carefully. For example, if the 82527 does not transmit the AAH and 55H synchronization bytes correctly, then the previous transmission may be incorrect too.

The MISO pin is tri-stated if CS# is inactive.

82527 SPI Interface Schematic



9.0 82527 FRAME TYPES

The 82527 communications controller supports four different frame types:

- — data frame
- — remote frame
- — error frame
- — overload frame

9.1 Data Frame

A data frame is composed of seven different fields:

- — start bit
- — arbitration field
- — control (identifier) field
- — data field
- — CRC field
- — acknowledge field
- — end of frame

The following describes the standard and extended message formats for data and remote frames shown above.

SOF: Start Of Frame (dominant bit) marks the beginning of a data/remote frame.

Arbitration: One or two fields which contain the message identifier bits. The standard format has one 11-bit field and the extended format has two fields, 11- and 18-bits wide.

RTR: Remote Transmission Request bit is dominant for data frames and recessive for remote frames. This bit is in the arbitration field.

SRR: Substitute Remote Request bit is used in extended messages and is recessive. This bit is a substitute for the RTR bit in the standard format. This bit is in the arbitration field of the extended format.

IDE: Identifier Extension bit is dominant for standard format and recessive for extended format. This bit is in the arbitration field of the extended format and in the control field of the standard format.

Control Field: Reserved bits r0 and r1 are sent as dominant bits. The 4-bit Data Length Code (DLC) indicates the number of bytes in the data field.

Data Field: The data bytes are located in the data frame (0–8 bytes). A remote frame contains zero data bytes.

CRC Field: This field is composed of a 15-bit Cyclical Redundancy Code error code and a recessive CRC delimiter bit.

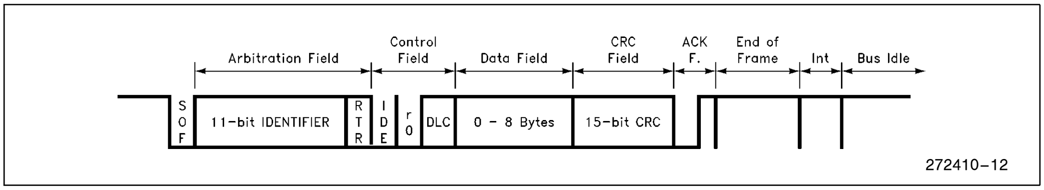
ACK Field: Acknowledge is a dominant bit sent by nodes receiving the data/remote frame and is followed by a recessive ACK delimiter bit.

End of Frame: Seven recessive bits ending the frame.

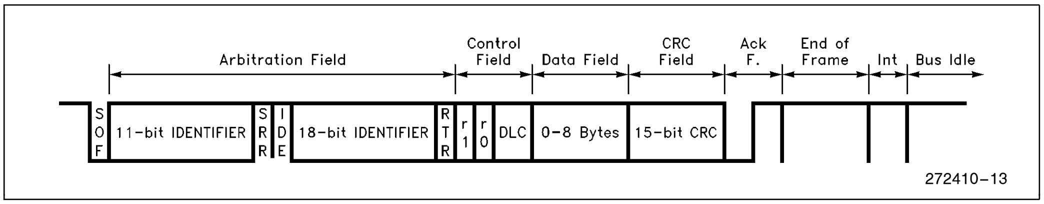
INT: Intermission is the three recessive bits which separate data and remote frames.

The minimum message lengths of standard and extended message formats are summarized for data and remote frames. The actual lengths of these messages may differ because "stuff" bits are added to the message. Stuff bits assist synchronization by adding transitions to the message. A stuff bit is inserted in the bit stream after five consecutive-equal value bits are transmitted; the stuff bit is the opposite polarity of the five consecutive bits. All message fields are stuffed except the CRC delimiter, the ACK field and the End of Frame.

Standard Format



Extended Format



CAN Message Formats

Standard Format

Message Field	Number of Bits
SOF	1
Arbitration (ID)	11
RTR	1
IDE	1
r0	1
DLC	4
Data Field	0–64
CRC Field	16
ACK Field	2
End of Frame	7
Total	44–108 bits

Extended Format

Message Field	Number of Bits
SOF	1
Arbiration (ID)	11
SRR	1
IDE	1
Arbitration (ID)	18
RTR	1
r1	1
r0	1
DLC	4
Data Field	0–64
CRC Field	16
ACK Field	2
End of Frame	7
Total	64–128 bits

9.2 Remote Frame

A data frame is composed of six different fields:

— start bit

— arbitration field

— control (identifier) field

— data field

— CRC field

— acknowledge field

— end of frame

Contrary to the Data Frame, the RTR-bit of the Remote Frame is “recessive” and no data segment is transmitted independent of the Data Length Code set by the Descriptor of the corresponding Communication Object.

The RTR-bit allows Remote Transmission Requests from any node to the system. This provides the capability to request information in addition to the standard broadcast characteristics. It also supports powerful diagnostic capability by being able to determine if the primary transmitter (data source) of a specific parameter(s) is on the bus and functional.

9.3 Error Frame

The Error Frame contains a sequence of variable length dominant bits as a result of error flags being transmitted by different system-nodes. This is an important aspect of the 82527 communication protocol with regards to data consistency within a communication network. The error frame is followed by an error delimiter.

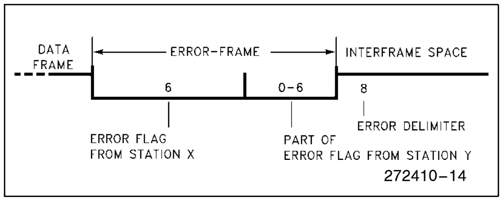


Figure 1. Error Frame Format

ERROR FLAG consists of six consecutive dominant bits. Since this “violates” bit stuffing rules, it is used as an error indicator to the system (see Coding/Decoding).

An Error Flag is transmitted if an 82527 operates as an error active node and has detected an error condition during or after a message transfer. If an Error Flag is generated by a transmitter, or a receiver, all other nodes interpret the Error Flag as a bit stuffing rule violation. As a consequence, they, in turn, transmit an error flag. A variable sequence of dominant bits result from the superposition of the different Error Flags transmitted by individual nodes. The total length of the Error Flag sequence varies between six bits minimum to twelve bits maximum.

An error condition is signaled by the transmission of six recessive bits while in the error passive operation mode. This way an error passive node with a temporary local receiver problem will not destroy messages received correctly by other nodes. The recessive bits may be overwritten by an Error Flag generated by one or more error active system nodes, but the error passive 82527 waits for at least six bits of equal polarity before entering into the next internal receive or transmit mode. (See Error Handling for error active/passive mode.)

NOTE:
The 82527 will not perform storage of a message (positive acceptance filtering) into the communication buffer, if reception of the message was followed by an Error Flag on the serial bus.

The error-delimiter consists of eight recessive bits generated by the 82527 after the end of an Error Flag on the serial bus line. This is monitored by detection of a transition from the dominant to recessive bit level.

Detected errors during the transmission of a data or remote frame can be signaled within the transmission time of the respective frame. This procedure associates an Error Flag to the corresponding frame, and initiates a retransmission of the frame. As the 82527 monitors, any deviation of its error frame will start retransmitting an error frame. If this occurs several times in a sequence the 82527 will become error passive.

9.4 Overload Frame

The overload frame consists of two bit fields, the overload flag and the overload delimiter.

There are two cases of overload conditions which result in the transmission of an overload flag:

1. Internal conditions of the receiver circuitry of a CAN chip which require a delay time before receiving the next frame (receiver not ready). The 82527 will not generate overload frames when CAN bus transmission rates are 1 Mbit/sec or less.
2. Detection of a "dominant" bit during Interframe Space.

The overload frame consists of six dominant bits that correspond to the Error Flag and destroy the fixed form of the Interframe Space Field. As a consequence, all other nodes see the dominant bit during the Interframe Space time and interpret the recessive to dominant edge as a start of frame and transmit an overload flag because of the overload condition.

The overload delimiter consists of seven recessive bits generated by the CAN chip.

After transmission of an overload frame, each 82527 within the system monitors the bus line until a transition from a dominant to a recessive level occurs. This indicates to each 82527 the end of overload frames and each node simultaneously starts the transmission of six more recessive bits.

NOTE:
The earliest time an overload frame can be transmitted is at the first bit time of the Interframe Space Field. This is contrary to the Error Frame and allows the 82527 to differentiate between the Error Frame and Overload Frame.

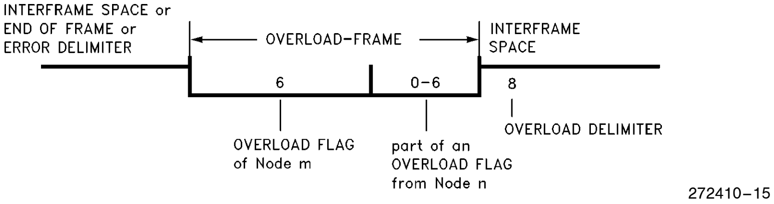


Figure 2. Overload Frame Format

Interframe-Space

Data Frame and Remote Frame are separated from preceding frames by an Interframe Space consisting of the Intermission bit field and a possible Bus Idle time. An error frame is not preceded by an Interframe Space.

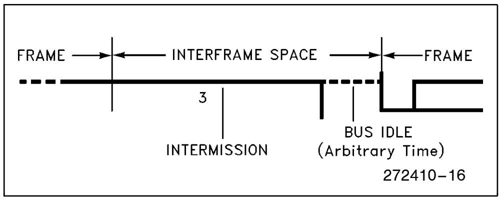


Figure 3. Interframe Space Format

INTERMISSION consists of three recessive bits. During Intermission time the 82527 will not start transmission of a frame. Intermission is a fixed time period for the 82527 to execute internal processes prior to the next receive or transmit task.

Data received within a data frame will be stored in the communication buffer and the control bits are updated if no error condition has occurred through the last bit of the end of frame field.

The bus idle time may be of arbitrary length. After the Interframe Space period, the 82527 looks for bus idle before initiating transmission, if requested by a CPU. The detection of a dominant bit after Intermission or bus idle is interpreted by the 82527 as Start of Frame.

9.5 Coding/Decoding

Coding

The frame segments (start of frame, arbitration field, control field, data field and CRC sequence) are coded using bit stuffing. Whenever the transmit logic of the 82527 detects five consecutive bits of identical levels to be transmitted, the logic inserts a complement bit in the transmitted bit stream.

Bit stuffing is used to guarantee enough edges in the NRZ Bit Stream to maintain synchronization.

Decoding

Whenever the 82527 has received five identical consecutive bit levels in the received bit stream the logic automatically deletes the next bit from the data stream (destuffing). Some field formats do not use bit stuffing. In these cases the bit stuffing and destuffing logic is turned off.

9.6 Arbitration

In the case when two or more 82527s start transmission concurrently, the bus access conflict is solved by a bitwise arbitration method during transmission of the arbitration field.

The transmit logic compares the bit level transmitted to the level monitored on the serial bus. Both the transmitter and receiver are on the bus at the same time. The transmit logic stops message transfer if a recessive bit was sent but a dominant bit was monitored. This method guarantees transmission of the message with the highest priority even if there is a collision during the arbitration field of one or more message Identifier(s).

The 82527 protocol architecture requires each message used in the communication network to have a unique Identifier characterizing the type of data within the data field. Using this method, the Identifier assigns a name to the data frame and automatically implies the priority of the message.

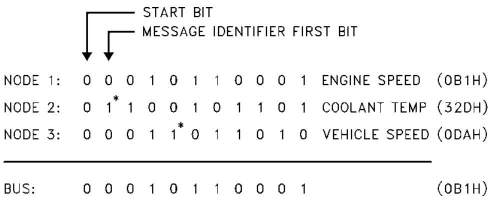
As a result, the Identifier during bus access represents not only the message name but, more important, the priority of each specific message. Since the most significant bit (MSB) of an Identifier is transmitted first, the Identifier with the smallest digital value has the highest priority for bus access.

An Identifier should not be used for more than one specific message to ensure that two or more nodes never simultaneously start a transmission of a data frame with the same message priority. Following this rule, bus access conflicts are resolved during the transmission of the Identifier.

One exception would be the simultaneous transmitter and receiver initiated frame transfer for the same message. If one 82527 generates a request for actual data of a certain type by transmitting a remote frame and simultaneously, the 82527 responsible for this type of data starts the transmission, arbitration can not be solved by the Identifier itself and actually is not required.

To deal with this exception, the RTR-bit is included in the arbitration field. The RTR-bit of the transmitter is always set dominant and, therefore, has a higher priority than the requesting 82527 (RTR-bit recessive). This way the remote frame request by the receiver gets an immediate response by the transmitter.

Example: Non-Destructive prioritized bitwise arbitration



```
NODE 1:   0 0 0 1 0 1 1 0 0 0 1  ENGINE SPEED  (0B1H)
NODE 2:   0 1* 1 0 0 1 0 1 1 0 1 COOLANT TEMP  (32DH)
NODE 3:   0 0 0 1 1* 0 1 1 0 1 0 VEHICLE SPEED (0DAH)
-----------------------------------------------
BUS:      0 0 0 1 0 1 1 0 0 0 1                (0B1H)
```

272410–17

NOTES:
Node 1 wins the arbitration with the lowest identifier (0B1H)
- Zero is the dominant bit on the bus

*Node detected the loss of the arbitration and stopped transmitting but continues to receive the message.

The CAN protocol architecture defines that each Communication Object used must have a unique Identifier characterizing the priority of the message. This allows bitwise arbitration of the bus if a conflict arises. The transmit logic compares the level monitored on the serial bus with that transmitted. The transmit logic immediately stops transmission if there is a conflict. This guarantees the data transfer of the Communication Object with the highest priority even if there is a collision.

10.0 ERROR DETECTION AND CONFINEMENT

The Error Detection mechanism is implemented in hardware for efficiency.

10.1 Bit Error

During a transmit operation, the 82527 monitors the bus on a bit-by-bit basis. If the bit level monitored is different from the transmitted bit, a bit error is signaled.

Exceptions: Arbitration and ACK-SLOT. During arbitration, a recessive bit can be overwritten by a dominant bit. In this case, the 82527 interprets a bit error as an arbitration loss. During the ACK-SLOT, a transmitter may detect a falsified bit (recessive to dominant) meaning that at least one receiver has received the message correctly.

NOTE:
Except during transmission of the arbitration field and during the time window of the ACK-SLOT, all global and local errors at the transmitter are detected.

10.2 Bit Stuffing Error

As described earlier, the frame segments are coded by a method of bit stuffing.

There are two possibilities where bit stuffing errors may occur:

1. A disturbance generates more consecutive bits of equal level than allowed by the rule of bit stuffing. These errors are detected by all nodes.
2. A disturbance falsifies one or more of the five bits preceeding the stuff bit. This error is not recognized by a receiver; however, if the error also appears at the transmitter, it will be detected as a bit error (transmitter monitors bus as it transmits).

In any case, the error is detected by a receiver either by the bit stuffing mechanism (the stuff bit of the transmitter is not dropped but taken as an information bit) or by the CRC check.

10.3 CRC Error

To ensure the validity of a transmitted message, all receivers perform a CRC check. In addition to the information bits, the CRC includes control bits used for error detection.

Description of the CRC Code

The code used for the 82527 is a (shortened) BCH Code, extended by a parity check and the following attributes:

— 127 bits as maximum length of the code word

— length of the CRC sequence is 15 bits

— Hamming distance d = 6
d = min A (x EXOR y) / x, y different code words
A(x) = number of "recessive" bits in the code word x

$f(x) = (x^{14} + x^9 + x^8 + x^6 + x^5 + x^4 + x^2 + x + 1) (x + 1)$

f(x) = 1100 0101 1001 1001

f(x) = C599 Hex

Burst errors are detected up to a length of 15 (degree of f(x)). Multiple errors (number of disturbed bits at least d = 6) are not detected with a residual error probability of 3×10^{-5}.

10.4 Form Error

Form Errors result from the violation of the fixed form of the following bit fields:

— end of frame
— interframe space

- ACK delimiter
- CRC delimiter

During the transmission of these bit fields, an error condition is recognized if a “dominant” bit level is detected.

10.5 Error Detection Capabilities

Global errors, which occur at all fully functional nodes, are 100% detectable.

For local errors, e.g. errors which may appear at some nodes only, the shortened BCH Code extended by the parity check has the following error detection capabilities:

- Up to 5 single bit errors are detected 100% even if those errors are being distributed randomly within the code word.
- All single bit errors are detected if their total number within the code word is odd.
- The residual error probability of the CRC check is $2^{-15} = 3 \times 10^{-5}$. As an error may be detected by the CRC check, and/or by additional implemented error detection mechanism, the residual error probability is significantly less than 3×10^{-5}.

10.6 Error Confinement

Error Confinement is implemented on the 82527 as a self-checking mechanism for distinguishing “temporary errors” from “permanent failures”. A permanent failure is noted when an average of one in eight messages is corrupted. This type of error condition can be caused by a defective connector, transmitter, receiver or a long lasting disturbance from outside the network. If the node continues to observe a failure over a period of time the node will be removed from the bus. A disconnected node is not placed again on the serial bus until the CPU has issued a software reset to the 82527, and an 82527 internal delay time has elapsed.

The implementation of the error confinement consists of two counters (RECEIVE-ERROR-COUNT and TRANSMIT-ERROR-COUNT) and some control logic. These counters are modified according to a number of rules, which may be considered as the core of the error confinement. If a message is transmitted or received without an error the error-counter is decremented by a fixed number, if it is not already 0. The error-counter is increased by a fixed number, if an error is detected on the serial bus. No access is provided to the ERROR-COUNTERS; however, two flags are provided (Error Status and Bus Status) as a summary of error events.

The count added to the error-counter depends on the type of the error detected. For instance, whenever a node detects and reports an error condition (error flag), all system nodes will also detect an error condition due to that error flag, even if the information up to that time was received error-free.

NOTE:
In case an error condition is not detected by all nodes at the exact same bit time, the node reporting the error first is more likely to be responsible for such an error condition compared to those nodes reacting to the error flag. Therefore, a node that is often responsible for error conditions, as mentioned above, is the origin of the error as a result of a defect.

In general, a defective node exchanges information for a short time so as to prevent it from loading the bus and slowing down other nodes on the bus.

Three error states are flagged in the STATUS-REGISTER as follows:

An 82527 in the busoff state will neither transmit nor receive messages. In order to restart the 82527 it is necessary to reset the init bit in the control register (00H).

NOTE:
After this sequence, the CAN node will be active on the serial bus after 128 x 11 consecutive recessive bits.

10.7 82527 States with Respect to the Serial Bus

The 82527 may be in one of the three following states:

- Error Active
- Error Passive
- Bus Off

Error-Active is the normal mode of operation, it is characterized by the 82527 transmitting an ERROR-FLAG of 6 dominant bits in case a receive or a transmit error is detected.

An “Error Passive” 82527 does not send an ACTIVE ERROR FLAG (6 dominant bits). In this mode, the 82527 communicates on the bus but on detecting a receive or a transmit error, it sends a PASSIVE ERROR FRAME (6 recessive bits). After transmitting a message, an “Error Passive” 82527 does not initiate another transmission immediately; instead, after the INTER-FRAME SPACE, it transmits 7 “recessive bits”. If during this time period (7 recessive bits), another 82527 starts transmission on the bus, the “error passive” 82527 becomes the receiver.

A busoff 82527 does not transmit or receive any information; its output drivers are put in a float state. This state is indicated to the user by the BOff bit in the Status Register (01H).

For Error Confinement two error counters are implemented in the 82527 architecture:

1. TRANSMIT AND RECEIVE ERROR COUNTER

These counters are modified by the 82527 with no read or write access to the user.

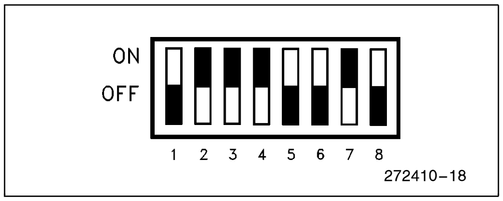
At the system start-up, there may be only one 82527 active on the bus. If this node transmits a message, it will not get an acknowledgement and hence detect an error; it will correspondingly repeat the message. Because of repeated transmissions, 82527 will become "Error Passive" but not busoff.

Similarly, an 82527 sending a wake-up message may become "Error Passive."

11.0 SAMPLE PROGRAM

The following pages contain a software program written for the Intel 87C196KR microcontroller which is used in the Intel EV82527 Evaluation Kit for the 82527 chip. This code may serve as a guide to configure the 82527. This code configures a transmit, receive, and remote messages. Interrupts are used to receive messages. The program uses Ports 1 and 2 extensively to read dip switches and the control an LED display. For illustrative purposes, the dip switch control features are shown below and these functions are all implemented in the following code.

The DIP Switch Definitions:



272410-18

Switch 1

0 standard format message objects (11-bit ID)

1 extended format message objects (29-bit ID)

Switch 2

0 display transmitted data

1 display received data

Switch 3/4

0 0 receive message on ID 0 (switch 8 = 0) or ID 2048 (switch 8 = 1)

0 1 receive message on ID 1 (switch 8 = 0) or ID 2049 (switch 8 = 1)

1 0 receive message on ID 2 (switch 8 = 0) or ID 2050 (switch 8 = 1)

1 1 receive message on ID 3 (switch 8 = 0) or ID 2051 (switch 8 = 1)

Switch 5/6

0 0 transmit message on ID 0 (switch 8 = 0) or ID 2048 (switch 8 = 1)

0 1 transmit message on ID 1 (switch 8 = 0) or ID 2049 (switch 8 = 1)

1 0 transmit message on ID 2 (switch 8 = 0) or ID 2050 (switch 8 = 1)

1 1 transmit message on ID 3 (switch 8 = 0) or ID 2051 (switch 8 = 1)

Switch 7

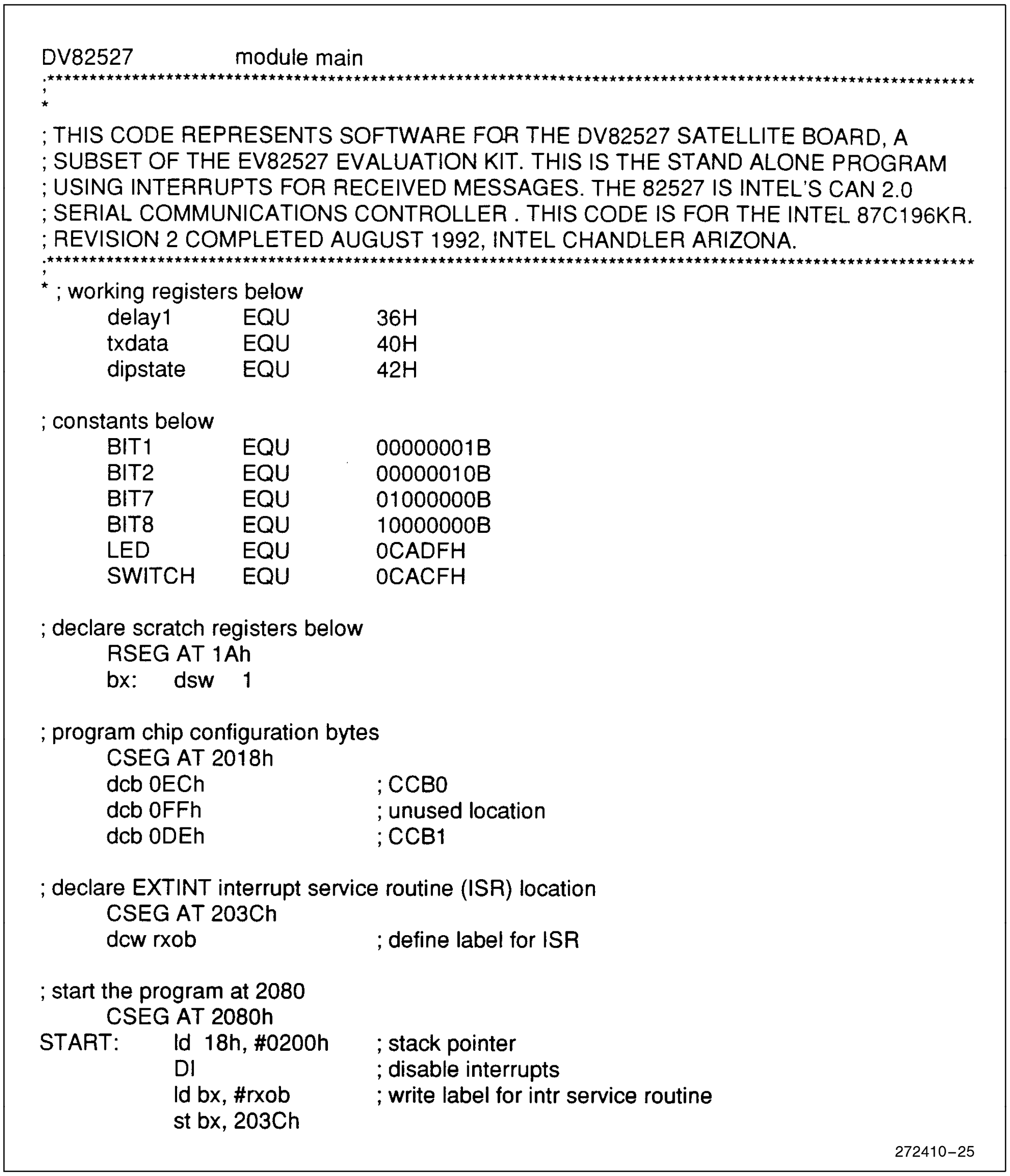
0 receive message using receive message object

1 send remote message from receive message object

Switch 8

0 transmit slow counter values

1 transmit value currently on DIP switches

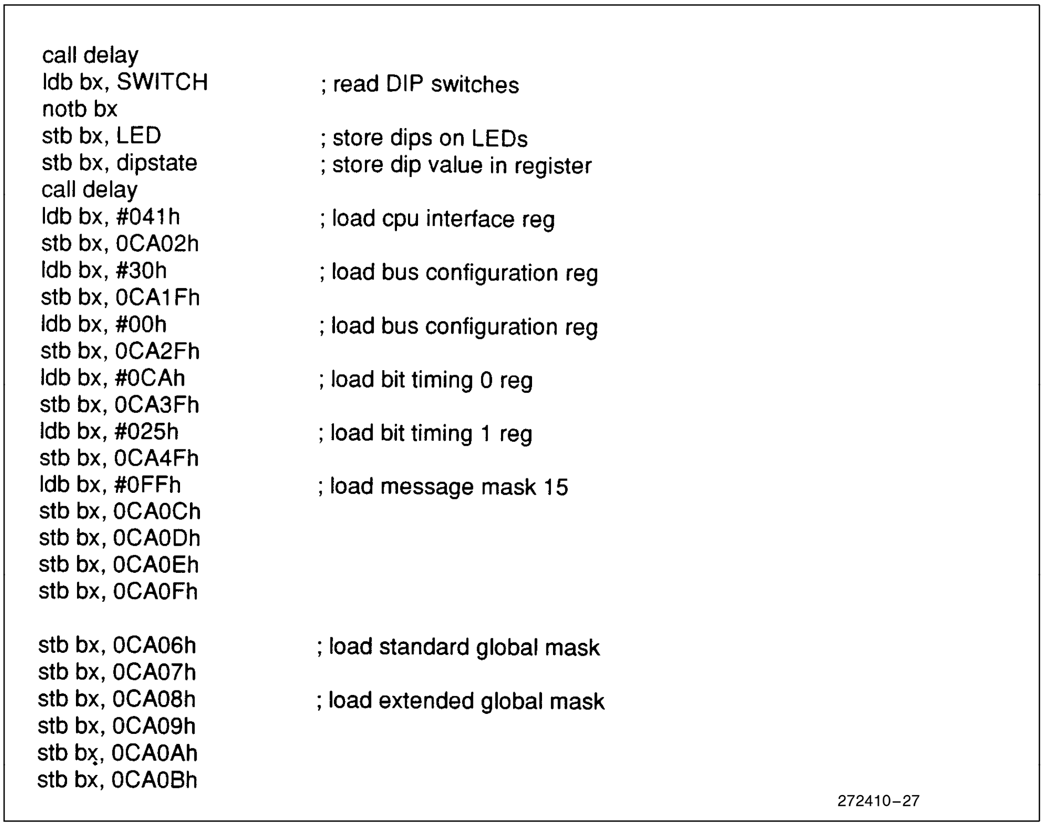


```
DV82527            module main
;*********************************************************************************************************
;
*
; THIS CODE REPRESENTS SOFTWARE FOR THE DV82527 SATELLITE BOARD, A
; SUBSET OF THE EV82527 EVALUATION KIT. THIS IS THE STAND ALONE PROGRAM
; USING INTERRUPTS FOR RECEIVED MESSAGES. THE 82527 IS INTEL'S CAN 2.0
; SERIAL COMMUNICATIONS CONTROLLER . THIS CODE IS FOR THE INTEL 87C196KR.
; REVISION 2 COMPLETED AUGUST 1992, INTEL CHANDLER ARIZONA.
;*********************************************************************************************************
;
* ; working registers below
        delay1       EQU        36H
        txdata       EQU        40H
        dipstate     EQU        42H

; constants below
        BIT1         EQU        00000001B
        BIT2         EQU        00000010B
        BIT7         EQU        01000000B
        BIT8         EQU        10000000B
        LED          EQU        0CADFH
        SWITCH       EQU        0CACFH

; declare scratch registers below
        RSEG AT 1Ah
        bx:     dsw    1

; program chip configuration bytes
        CSEG AT 2018h
        dcb 0ECh                ; CCB0
        dcb 0FFh                ; unused location
        dcb 0DEh                ; CCB1

; declare EXTINT interrupt service routine (ISR) location
        CSEG AT 203Ch
        dcw rxob                ; define label for ISR

; start the program at 2080
        CSEG AT 2080h
START:          ld  18h, #0200h ; stack pointer
                DI              ; disable interrupts
                ld bx, #rxob    ; write label for intr service routine
                st bx, 203Ch
```

272410-25

```
; program port 2 configuation registers - define P2.2 as a special
; function input for EXTINT (external interrupt) of 87C196KR controller
        ldb bx, 1FCBh              ; configure pin P2.2 P2IO register
        orb bx, #04h
        stb bx, 1FCBh
        ldb bx, 1FCDh              ; configure pin P2.2 P2REG register
        orb bx, #04h
        stb bx, 1FCDh
        ldb bx, 1FC9h              ; configure pin P2.2 P2SSEL register
        orb bx, #04h
        stb bx, 1FC9h

; program interrupt mask
        ldb bx, 13h
        orb bx, #40h               ; enable EXTINT interrupts in interrupt mask
        stb bx, 13h

; program port 5 for special function do that bus timing signals are enabled
        ldb bx, #04Dh              ; program P5.1, P5.6 and P5.7 for LSIO
        stb bx, 1FF1h[0]           ; program P5SSEL for special bus functions

; hard reset of 82527 device using pin 5.1
        ldb bx, #00h               ; enable push pull outputs
        stb bx, 1ff3h              ; write P5IO register
        ldb bx, #00h
        stb bx, 1ff5h              ; write P5REG to toggle P5.1
        call delay                 ; wait  for 15 mS
                                           - only need 1 delay for B-step
        ldb bx, #0ffh
        stb bx, 1ff5h              ; toggle P5.1

; start programming the 82527
begin527:
        ldb bx, 0ca02h             ; load CPU interface reg
        andb bx, #80h              ; mask RstSt (reset status) bit
        jne begin527               ; go back if not reset# pin not high

; begin initialization
init527:ldb bx, #041h              ; set initialization bit, CCE bit
        stb bx, 0ca00h             ; store in control register
        ldb bx, #0ffh
        stb bx, 0ca9fh             ; configure port1 for output
        ldb bx, #00h
        stb bx, 0caafh             ; configure port2 for input
        ldb bx, #055h
        stb bx, LED                ; write to LEDs
```

272410-26



```
call delay
ldb bx, SWITCH          ; read DIP switches
notb bx
stb bx, LED             ; store dips on LEDs
stb bx, dipstate        ; store dip value in register
call delay
ldb bx, #041h           ; load cpu interface reg
stb bx, 0CA02h
ldb bx, #30h            ; load bus configuration reg
stb bx, 0CA1Fh
ldb bx, #00h            ; load bus configuration reg
stb bx, 0CA2Fh
ldb bx, #0CAh           ; load bit timing 0 reg
stb bx, 0CA3Fh
ldb bx, #025h           ; load bit timing 1 reg
stb bx, 0CA4Fh
ldb bx, #0FFh           ; load message mask 15
stb bx, 0CA0Ch
stb bx, 0CA0Dh
stb bx, 0CA0Eh
stb bx, 0CA0Fh

stb bx, 0CA06h          ; load standard global mask
stb bx, 0CA07h
stb bx, 0CA08h          ; load extended global mask
stb bx, 0CA09h
stb bx, 0CA0Ah
stb bx, 0CA0Bh
```

272410–27

```
; write to message object control 0 registers
        ldb bx, #055h               ; write control 0 reg for messages 2-15
        stb bx, 0CA10h              ; all messages initialized as invalid
        stb bx, 0CA20h
        stb bx, 0CA30h
        stb bx, 0CA40h
        stb bx, 0CA50h
        stb bx, 0CA60h
        stb bx, 0CA70h
        stb bx, 0CA80h
        stb bx, 0CA90h
        stb bx, 0CAA0h
        stb bx, 0CAB0h
        stb bx, 0CAC0h
        stb bx, 0CAD0h
        stb bx, 0CAE0h
        stb bx, 0CAF0h

; configure message objects 1 and 2
        ldb bx, SWITCH              ; get dip switch
        notb bx                     ; invert data
        andb bx, #BIT8              ; check for extended or standard format
        je std

extend:ldb bx, #01Ch                ; load message 1 - transmit/extended format
        stb bx, 0CA16h
        ldb bx, #014h               ; load message 2 - receive/extended format
        stb bx, 0CA26h
        ldb bx, #00h                ; write arbitration registers - message 1
        stb bx, 0CA12h
        stb bx, 0CA13h
        stb bx, 0CA22h
        stb bx, 0CA23h
; program message identifier value of 2048, 2049, 2050, or 2051
        ldb bx, #040h               ; loads value 2048
        stb bx, 0CA14h
        stb bx, 0CA24h
        ldb bx, SWITCH              ; load DIP
        notb bx
        and bx, #0Ch                ; get the transmit message bits
        shl bx, #1                  ; shift left one bit
        stb bx, 0ca15h              ; will program 2048, 2049, 2050 or 2051
        ldb bx, SWITCH              ; load DIP
        notb bx
        and bx, #30h                ; mask receive message bits
        shr bx, #1                  ; shift right one bit
```

272410–28

```
        stb bx, 0ca25h          ; will program 2048, 2049, 2050 or 2051
        sjmp control
std:    ldb bx, #018h           ; load message 1 - transmit/standard format
        stb bx, 0CA16h
        ldb bx, #01h            ; load message 1 transmit data
        stb bx, 0CA17h
        ldb bx, #010h           ; load message 2 - receive/standard format
        stb bx, 0CA26h
        ldb bx, #00h            ; write arbitration registers - message 1
        stb bx, 0CA12h
        stb bx, 0CA15h
        stb bx, 0CA14h
        stb bx, 0CA22h
        stb bx, 0CA25h
        stb bx, 0CA24h
; program message identifier value of 0, 1, 2, or 3
        ldb bx, SWITCH          ; load DIP
        notb bx                 ; invert data
        and bx, #0Ch            ; mask transmit bits
        shl bx, #3              ; shift left three bits
        stb bx, 0ca13h          ; will program 0, 1, 2 or 3
        ldb bx, SWITCH          ; load DIP
        notb bx
        and bx, #30h            ; mask receive bits
        shl bx, #1              ; shift left one bit
        stb bx, 0ca23h          ; will program 0, 1, 2 or 3

; program remote message for both extended or standard frame messsages
        control:ldb bx, #095h   ; message 0 valid
        stb bx, 0CA10h
        ldb bx, #099h           ; message 1 valid and set receive enable
        stb bx, 0ca20h

; get and mask remote request dip switch bit
        ldb bx, SWITCH          ; load DIP
        notb bx
        and bx, #BIT2           ; mask remote bit
        je noremote             ; check value of remote switch
        ldb bx, #065h           ; set TxRqst bit to send remote message
        stb bx, 0CA21h
        br done                 , remote receive set
        noremote: ldb bx, #055h ; no remote frame message
        stb bx, 0ca21h
```

272410-29

```
done: ldb bx, #02h              ; out of reset, enable message object
      stb bx, 0ca00h            ; interrupts by setting IE bits
      call delay                ; wait
      EI                        ; enable 87C196KR interrupts

; check whether to transmit slow counter or read dip switch
      ldb bx, switch
      andb bx, #BIT1
      jne counter

; MAIN LOOPS - transmit data
readdip:ldb bx, SWITCH          ; load dip switches
      notb bx
      stb bx, txdata            ; store data from dips in scratch register
      cmpb bx, dipstate         ; check if dip state changed
      jne doinit                ; re-initialize 82527 because new dipstate?
      andb bx, #BIT7            ; check whether to store data on LEDs
      jne nodisp
      stb dipstate, LED         ; store transmit on LEDs

nodisp:stb dipstate, 0ca17h     ; store data in message object
      ldb bx, #066h             ; set transmit request bit
      stb bx, 0CA11h

; send remote frame if enabled
      ldb bx, SWITCH            ; load DIP
      notb bx
      and bx, #BIT2             ; mask remote bit
      je next1                  ; check value of remote switch
      ldb bx, #065h             ; set TxRqst bit to send remote message
      stb bx, 0CA21h

next1: call delay
      call delay
      sjmp readdip

doinit:ljmp init527             ; jump to init527 because dipstate changed

; transmit slow counter
counter:addb txdata, #1h        ; increment KR counter
      stb txdata, 0CA17h        ; store transmit data
      ldb bx, #066h             ; set transmit request bit
      stb bx, 0CA11h
      ldb bx, SWITCH            ; check whether to display transmit LEDs
      notb bx
      cmpb bx, dipstate         ; check if dip state changed
```

272410–30

```
        jne doinit              ; re-initialize 82527 because new dipstate?
        andb bx, #BIT7
        jne z1
        stb txdata, LED         ; store data on LEDs

; send remote frame if enabled
z1:     ldb bx, SWITCH          ; load DIP
        notb bx
        and bx, #BIT2           ; mask remote bit
        je next2                ; check value of remote switch
        ldb bx, #065h           ; set TxRqst bit to send remote message
        stb bx, 0CA21h
next2:  call delay              ; wait
        call delay
        call delay
        call delay
        je counter              ; go to top of loop

; delay subroutine
delay:  ld delay1, #01AFFh      ;load 1AFFh into delay1 register
sub1:   sub delay1, #01h        ;subtract 1 from 1AFFh
        cmp delay1, #00h        ;is delay1 memory value 0 yet?
        bne sub1                ;branch to sub1 if not 0
        ret
; interrupt service routine for arriving messages
rxob:
        pusha
        push bx                 ; save delay counter
        DI                      ; disable interrupts
        ldb bx,0ca5fh           ; read interrupt pointer
        ldb bx, SWITCH          ; read dip switches
        notb bx                 ; invert data
        andb bx, #BIT7          ; check whether to display receive data
        je b1
        ldb bx, 0ca27h
        stb bx, LED
b1:     ldb bx, #055h
        stb bx, 0ca21h
        ldb bx, #11111101b      ; reset IntPnd bit
        stb bx, 0ca20h
        pop bx
        popa
        EI                      ; enable interrupts
        ret
        end
```

272410–31