CIS 721 - Real-Time Systems

Lecture 18: Higher Layer CAN Protocols

Mitch Neilsen
neilsen@ksu.edu

Outline

- **Real-Time Communication**
  - Controller Area Network
  - **Higher Layer CAN Protocols**
    - **Implementation**
  - Time-Triggered Protocols (TTP/C, TTP/A,TTCAN)
- Safety Critical Systems

Higher Layer Protocols

- CAN Kingdom - Kvaser
- **SAE J1939** - for in-vehicle, heavy truck, and agricultural networking
- DeviceNet - ODVA, (Allen-Bradley, etc.)
- SDS - Honeywell
- NMEA 2000 - for networking marine craft
- CANopen - CiA (CAN in Automation)
- CDA 101 - DoD - implemented using CAN Kingdom

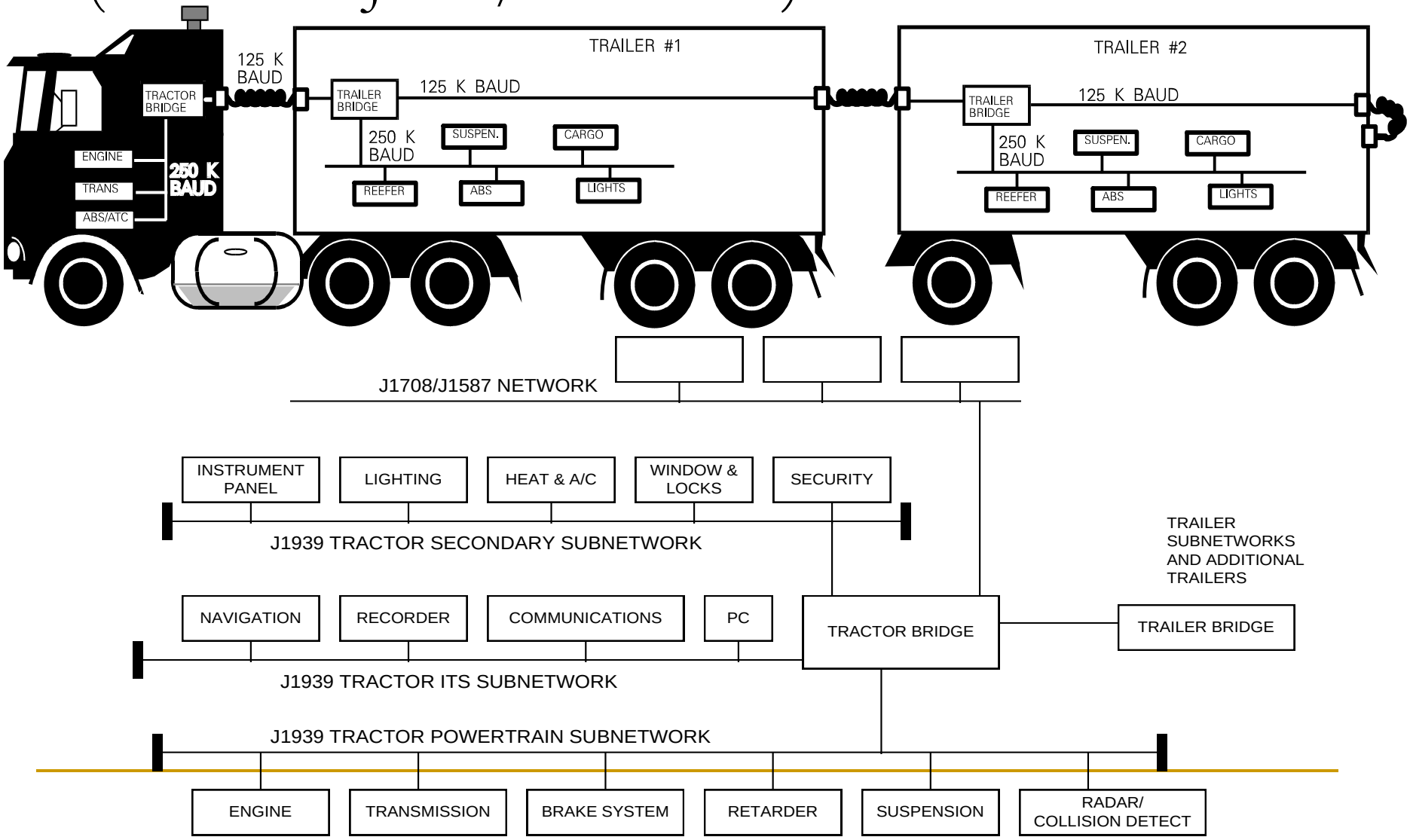
SAE J1939

- SAE J1939 was designed to allow electronic devices from different vendors to communicate with each other through a standard architecture.
- Only data frames are transmitted at a bit rate of 250 Kbps.
- Most CAN identifiers are defined by the standard.
- This results in system performance that may be far below the capability of CAN.

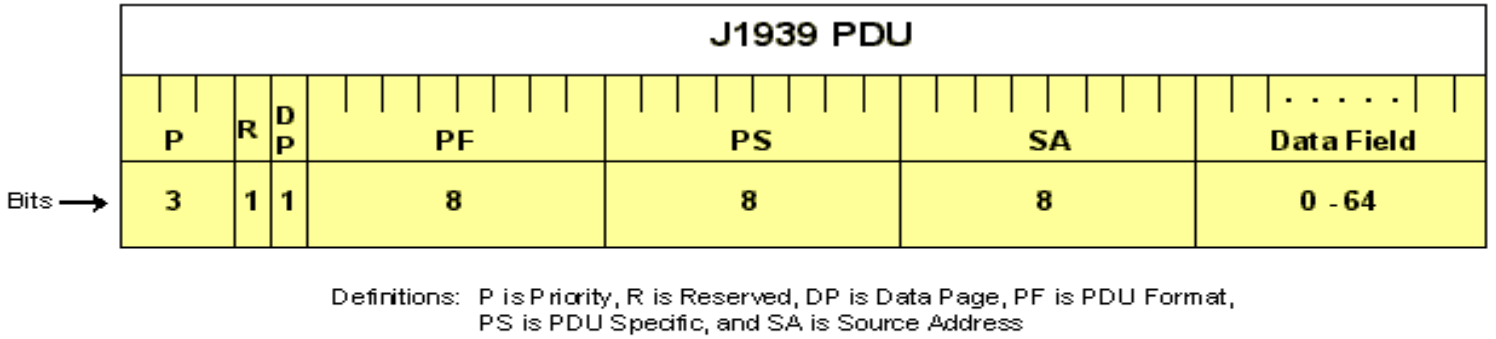
J1939 in OSI Reference Model

- J1939/11 - Physical Layer - twisted pair, twisted quad, ...
- J1939/21 - Data Link Layer - define frame (Protocol Data Unit (PDU)) format, point-to-point and broadcast (BLAST) protocols
- J1939/31 - Network Layer
- J1939/71 - Application Layer
- J1939/81 - Network Management
- J1939/0* - General documentation:
  - 01 - Truck and Bus
  - 02 - Agricultural Equipment
  - ...

Typical On-Highway Truck Implementation (From SAE J1939/01 DRAFT)



J1939 Protocol Data Unit



- The CAN 29-bit identifier is broken up into:
  - Priority (3 bits)
  - Reserved Bit (1 bit)
  - Data Page (1 bit)
  - PDU (Protocol Data Unit) Family (PF) (8 bits)
  - PDU Specific (PS) (8 bits)
  - Source Address (SA) (8 bits)

Higher Layer Protocols

- CAN Kingdom - Kvaser
- SAE J1939 - for in-vehicle, heavy truck, and agricultural networking
- **DeviceNet - ODVA, (Allen-Bradley, etc.)**
- SDS - Honeywell
- NMEA 2000 - for networking marine craft
- CANopen - CiA (CAN in Automation)
- CDA 101 - DoD - implemented using CAN Kingdom

ODVA DeviceNet (DN)

- ODVA Organization (Allen-Bradley, ...)
- Predefined **Master/Slave** connections and message identifiers
- Used for networking low-level Programmable Logic Controllers (PLC) for industrial automation
- Based on CAN 2.0A (11-bit identifiers)
- A wide range of products from more than 270 vendors

Higher Layer Protocols

- CAN Kingdom - Kvaser
- SAE J1939 - for in-vehicle, heavy truck, and agricultural networking
- DeviceNet - ODVA, (Allen-Bradley, etc.)
- **SDS - Honeywell**
- NMEA 2000 - for networking marine craft
- CANopen - CiA (CAN in Automation)
- CDA 101 - DoD - implemented using CAN Kingdom

Smart Distributed System (SDS)

- Honeywell standard based on CAN
- CAN identifier is related to node (host) identifier
- PC alternative to PLC
- A wide range of modules from 50 vendors
- SDS CAN identifier (short CAN 2.0A form):
  - Direction (1 bit) - Host to Guest or Guest to Host
  - Device Address (7 bits) - Host Identifier
  - PDU Type (3 bits) - 0 = change state to off, 1 = change state to on, 2 = off ack, 3 = on ack, 4 = write off, 5 = write on, 6 = write off ack, 7 = write on ack.

Mixing Higher Layer Protocols

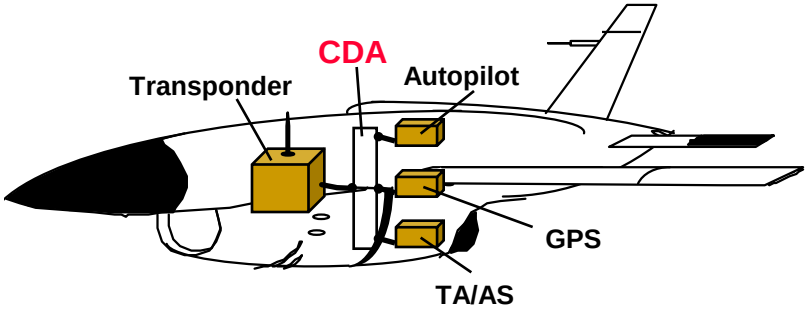
- For CAN Modules to co-exist on the same bus, they must:
  - use the same bit rate
  - use the same physical medium
  - use compatible, unique CAN identifiers
- Bit rates supported:
  - 1Mbps: CK, SDS
  - 500Kbps: CK, SDS, DN, CANopen
  - 250Kbps: CK, SDS, DN, J1939, NMEA2000
  - 125Kbps: CK, SDS, DN

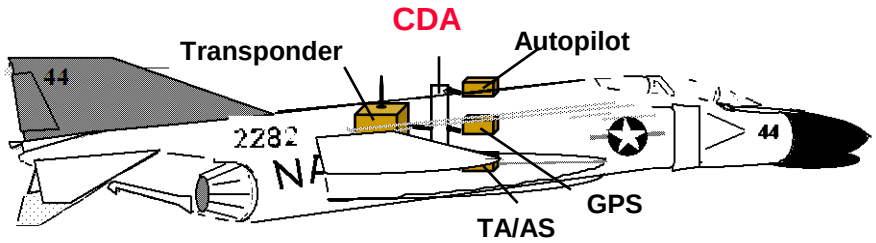
Higher Layer Protocols

- CAN Kingdom - Kvaser
- SAE J1939 - for in-vehicle, heavy truck, and agricultural networking
- DeviceNet - ODVA, (Allen-Bradley, etc.)
- SDS - Honeywell
- NMEA 2000 - for networking marine craft
- CANopen - CiA (CAN in Automation)
- **CDA 101 - DoD - implemented using CAN Kingdom**

Common Digital Architecture (CDA 101)

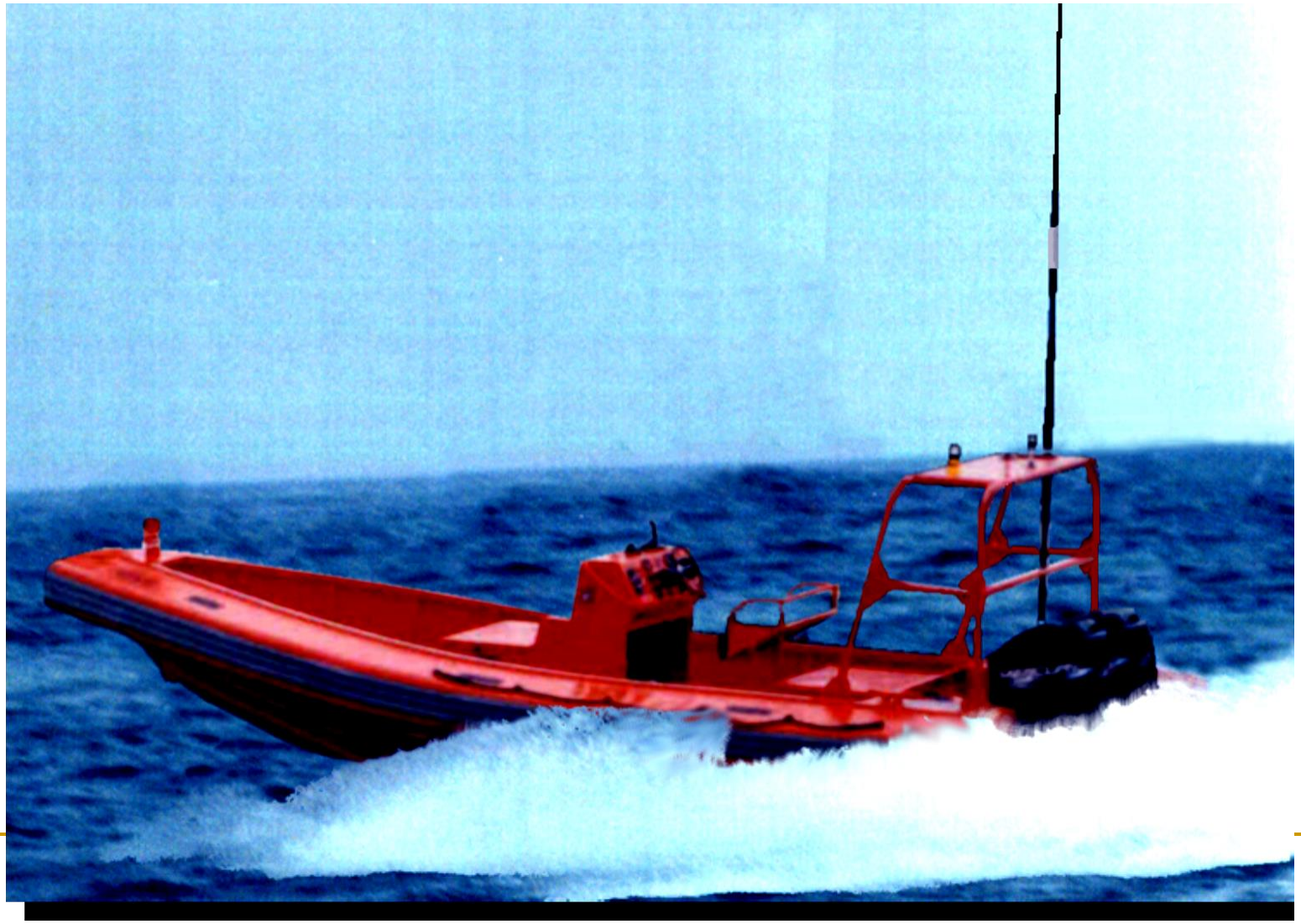
- A standard architecture for interconnecting target vehicle electronics



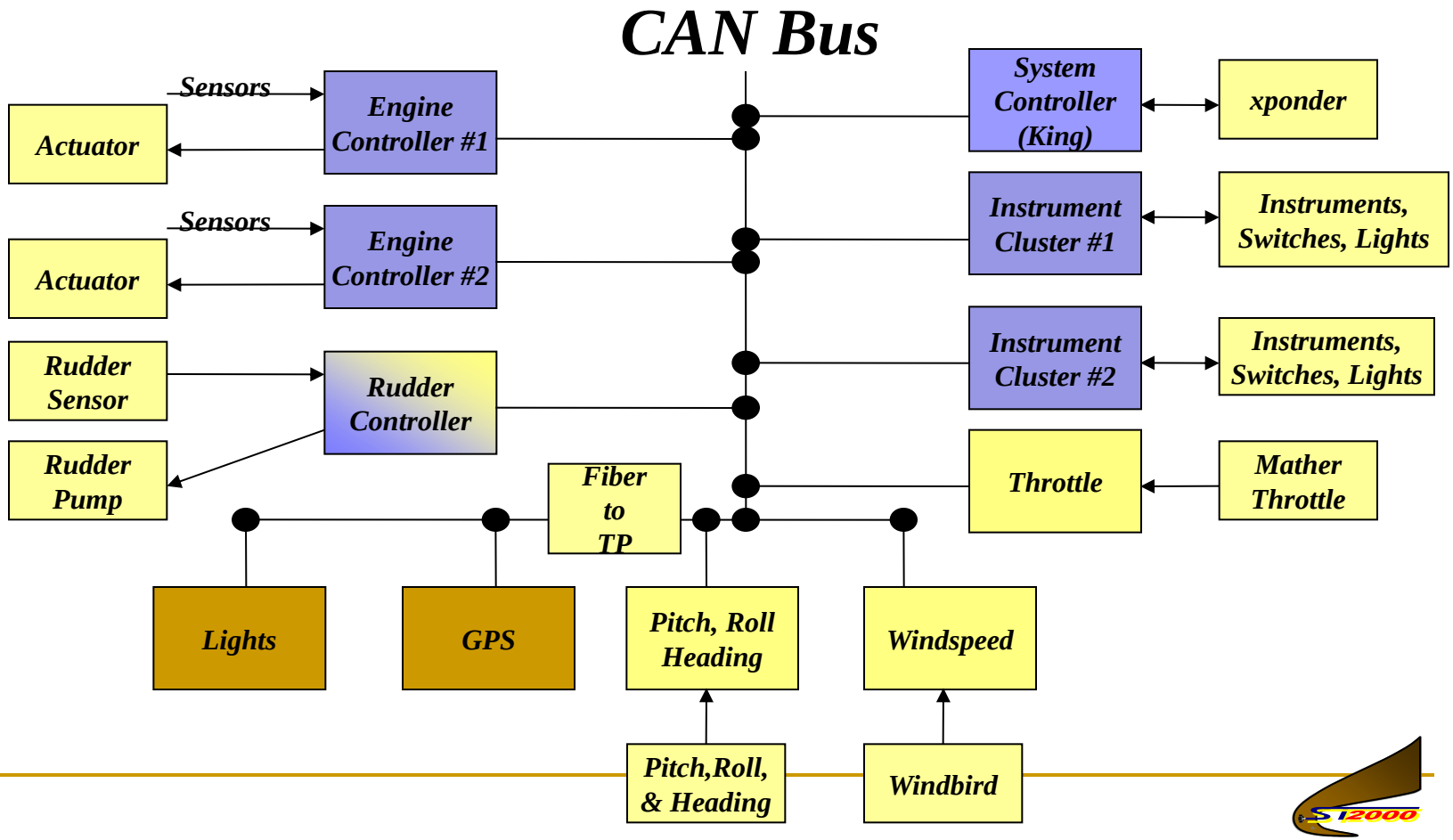


- Interface standards
  - Hardware
  - Software

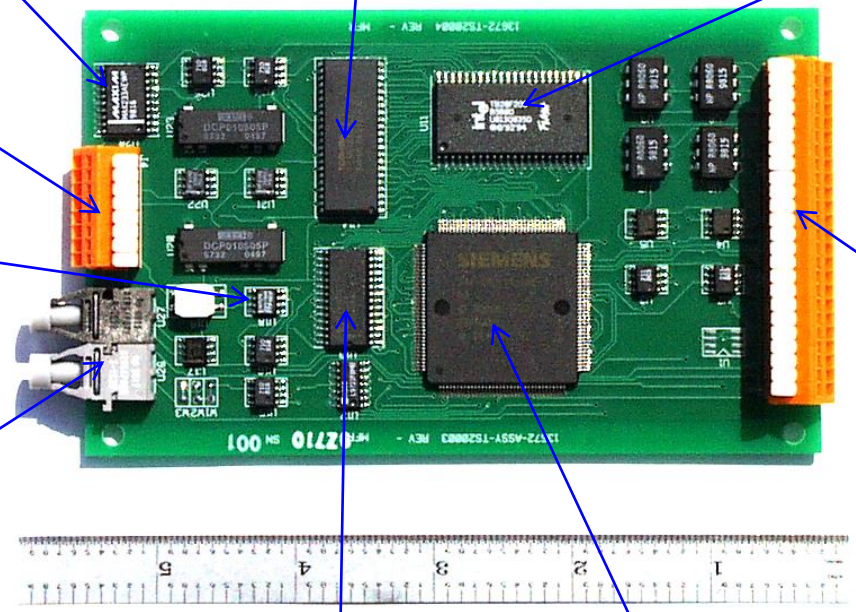
Seaborne Target ST2000



Seaborne Target ST2000



Typical CAN Node



Implementation

- **C167CR On-Chip CAN Module**
- **CAN Library (AppNote 2922)**
- **CAN Interrupt Structure (AppNote 1621)**
- **Tasking Development Environment**
  - **C/C++ Compiler**
  - **On-board Debugger**
  - **Phytec Flash Programmer**

C167CR On-Chip CAN Interface

- The C167CR supports Full CAN 2.0B functionality:
  - Data transfer rates up to 1Mbps
  - Data integrity (built in error checking)
  - Host processor unloading – the CAN controller handles most of the tasks autonomously
  - Flexible and powerful message passing
  - Fifteen message objects

Message Objects (15)

- All message objects can be updated **independently**.
- Maximum message length is **8 bytes**.
- Each message objects has a **unique identifier** and its own set of control and status bits.
- Each object can be configured with its **direction** as **either transmit or receive**, except the last message (object 15) which is only a **double receive buffer** with a special mask register.

Registers and Message Objects

- All registers and message objects of the CAN controller are located in the special CAN address area of 256 bytes, which is mapped into segment 0 and uses addresses 00'**EF00**H through 00'**EFFF**H.
- All registers are organized as 16-bit registers, located on word addresses. However, all registers may be accessed bytewise in order to select special actions without effecting other mechanisms.

CAN Module Address Map

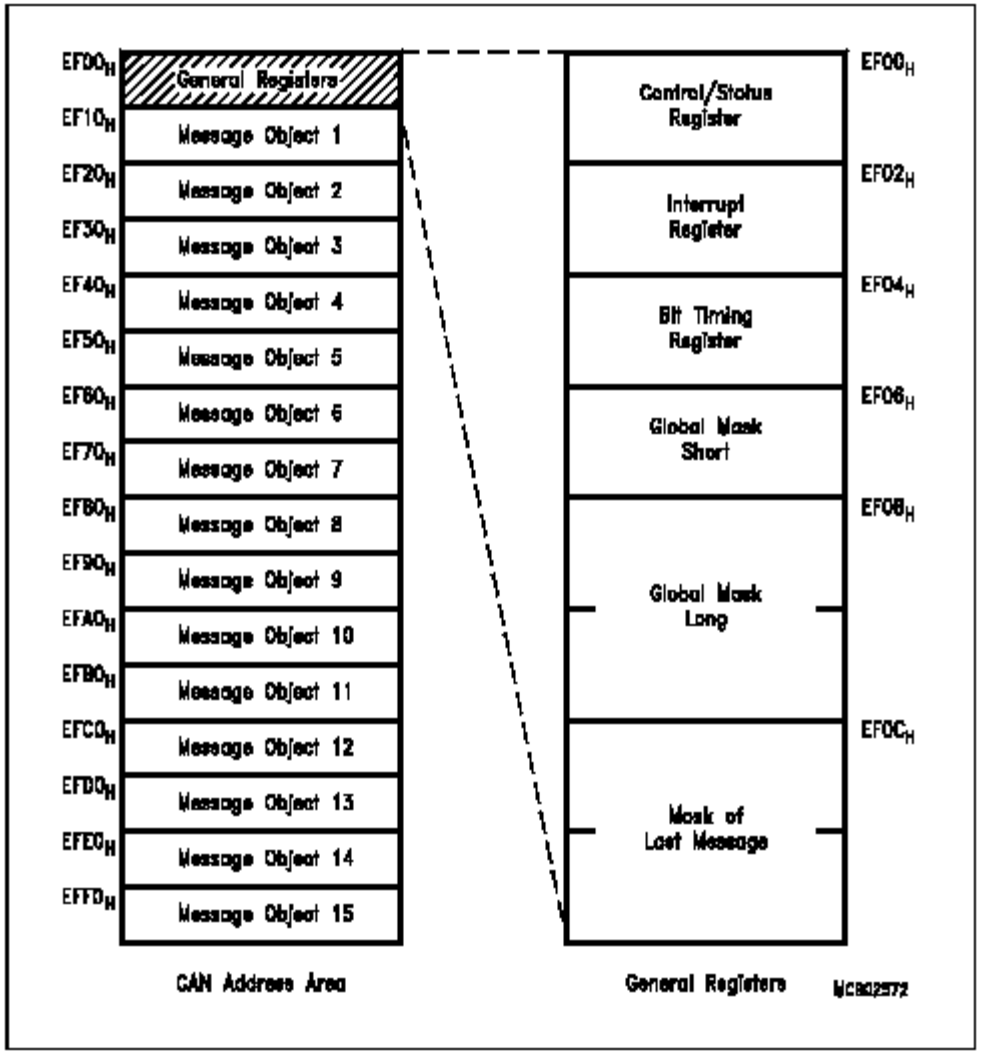


Figure 23-2
CAN Module Address Map

#include <canr16x.h> (c:\c166\include\canr16x.h)

/* Define CAN module control registers */

```
#define   CR          *(unsigned char*) 0xef00
#define   SR          *(unsigned char*) 0xef01
#define   IR          *(unsigned char*) 0xef02
#define   BTR         *(unsigned int *) 0xef04
#define   GMS  *(unsigned int *) 0xef06
#define   UGML        *(unsigned int *) 0xef08
#define   LGML*(unsigned int *) 0xef0a
#define   UMLM        *(unsigned int *) 0xef0c
#define   LMLM*(unsigned int *) 0xef0e
```

CAN Module Address Map

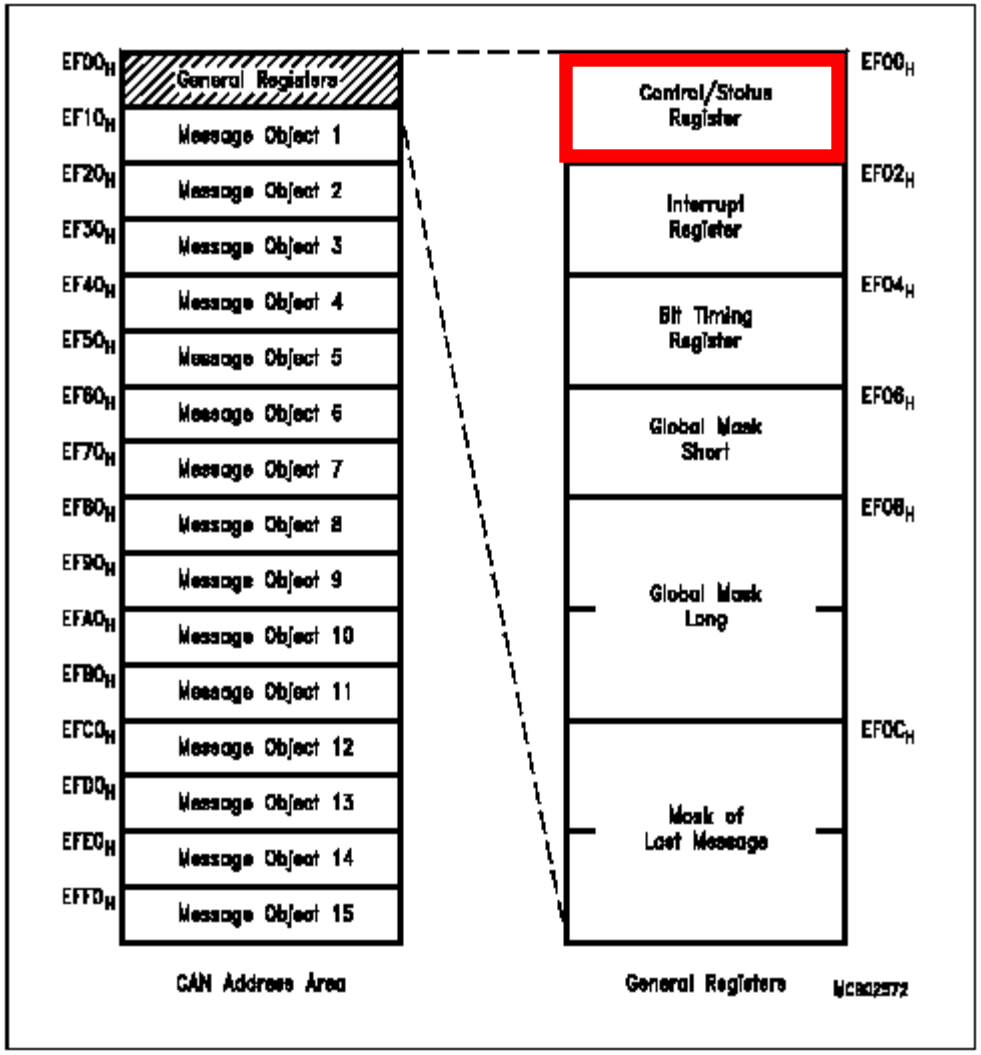


Figure 23-2
CAN Module Address Map

Control / Status Register ($EF00_H$) **XReg** **Reset Value: $XX01_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BOFF	EWRN	-	RXOK	TXOK	LEC			0 [1)]	CCE	0	0	EIE	SIE	IE	INIT
r	r	r	rw	rw		rw		rw	rw	r	r	rw	rw	rw	rw

Status Bits (SR) | Control Bits (CR)

	Starts the initialization of the CAN controller, when set.
IE	**Interrupt Enable** Enables or disables interrupt generation from the CAN Module via the signal $\overline{\text{XINTR}}$. Does not affect status updates.
SIE	**Status Change Interrupt Enable** Enables or disables interrupt generation when a message transfer (reception or transmission) is successfully completed or a CAN bus error is detected (and registered in the status partition).
EIE	**Error Interrupt Enable** Enables or disables interrupt generation on a change of bit BOFF or EWARN in the status partition).
CCE	**Configuration Change Enable** Allows or inhibits CPU access to the Bit Timing Register.
1)	**Test Mode** (Bit 7) Make sure that bit 7 is cleared when writing to the Control Register, as this bit controls a special test mode, that is used for production testing. During normal operation, however, this test mode may lead to undesired behaviour of the device.

Bit	Function (Status Bits)
LEC	**Last Error Code** This field holds a code which indicates the type of the last error occurred on the CAN bus. If a message has been transferred (reception or transmission) without error, this field will be cleared. 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:** A fixed format part of a received frame has the wrong format. 3 **AckError:** The message this CAN controller transmitted was not acknowledged by another node. 4 **Bit1Error:** During the transmission of a message (with the exception of the arbitration field), the device wanted to send a *recessive* level ("1"), but the monitored bus value was *dominant*. 5 **Bit0Error:** During the transmission of a message (or acknowledge bit, active error flag, or overload flag), the device wanted to send a *dominant* level ("0"), but the monitored bus value was *recessive*. During *busoff* recovery this status is set each time a sequence of 11 *recessive* bits has been monitored. This enables the CPU to monitor the proceeding of the busoff recovery sequence (indicating the bus is not stuck at *dominant* or continously disturbed). 6 **CRCError:** The CRC check sum was incorrect in the message received.
TXOK	**Transmitted Message Successfully** Indicates that a message has been transmitted successfully (error free and acknowledged by at least one other node), since this bit was last reset by the CPU (the CAN controller does not reset this bit!).
RXOK	**Received Message Successfully** Indicates that a message has been received successfully, since this bit was last reset by the CPU (the CAN controller does not reset this bit!).
EWRN	**Error Warning Status** Indicates that at least one of the error counters in the EML has reached the error warning limit of 96.
BOFF	**Busoff Status** Indicates when the CAN controller is in busoff state (see EML).

Note: Reading the upper half of the Control Register (status partition) will clear the Status Change Interrupt value in the Interrupt Register, if it is pending. Use byte accesses to the lower half to avoid this.

CAN Module Address Map

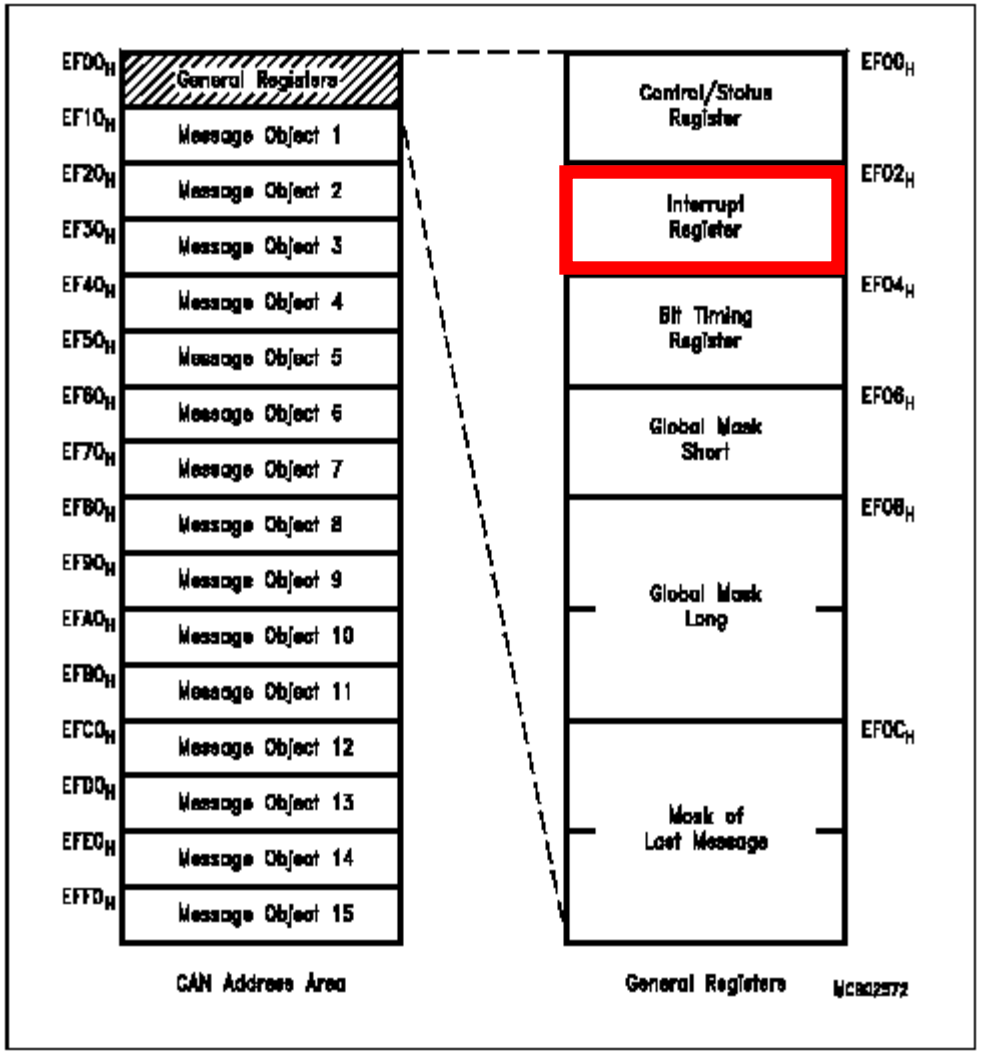
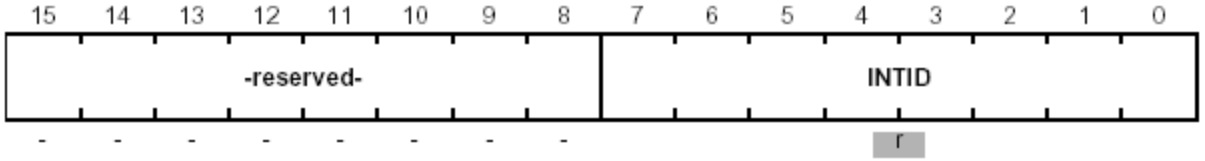


Figure 23-2
CAN Module Address Map

Interrupt Register ($EF02_H$) **XReg** **Reset Value: - - XX_H**



Bit	Function
INTID	**Interrupt Identifier** This number indicates the cause of the interrupt. When no interrupt is pending, the value will be "00".

INTID	Cause of the Interrupt
00	**Interrupt Idle:** There is no interrupt request pending.
01	**Status Change Interrupt:** The CAN controller has updated (not necessarily changed) the status in the Control Register. This can refer to a change of the error status of the CAN controller (EIE is set and BOFF or EWRN change) or to a CAN transfer incident (SIE must be set), like reception or transmission of a message (RXOK or TXOK is set) or the occurrence of a CAN bus error (LEC is updated). The CPU may clear RXOK, TXOK, and LEC, however, writing to the status partition of the Control Register can never generate or reset an interrupt. To update the INTID value the status partition of the Control Register must be read.
02	**Message 15 Interrupt:** Bit INTPND in the Message Control Register of message object 15 (last message) has been set. The last message object has the highest interrupt priority of all message objects. [1)]
(2+N)	**Message N Interrupt:** Bit INTPND in the Message Control Register of message object 'N' has been set (N = 1...14). Note that a message interrupt code is only displayed, if there is no other interrupt request with a higher priority. [1)]

CAN Module Address Map

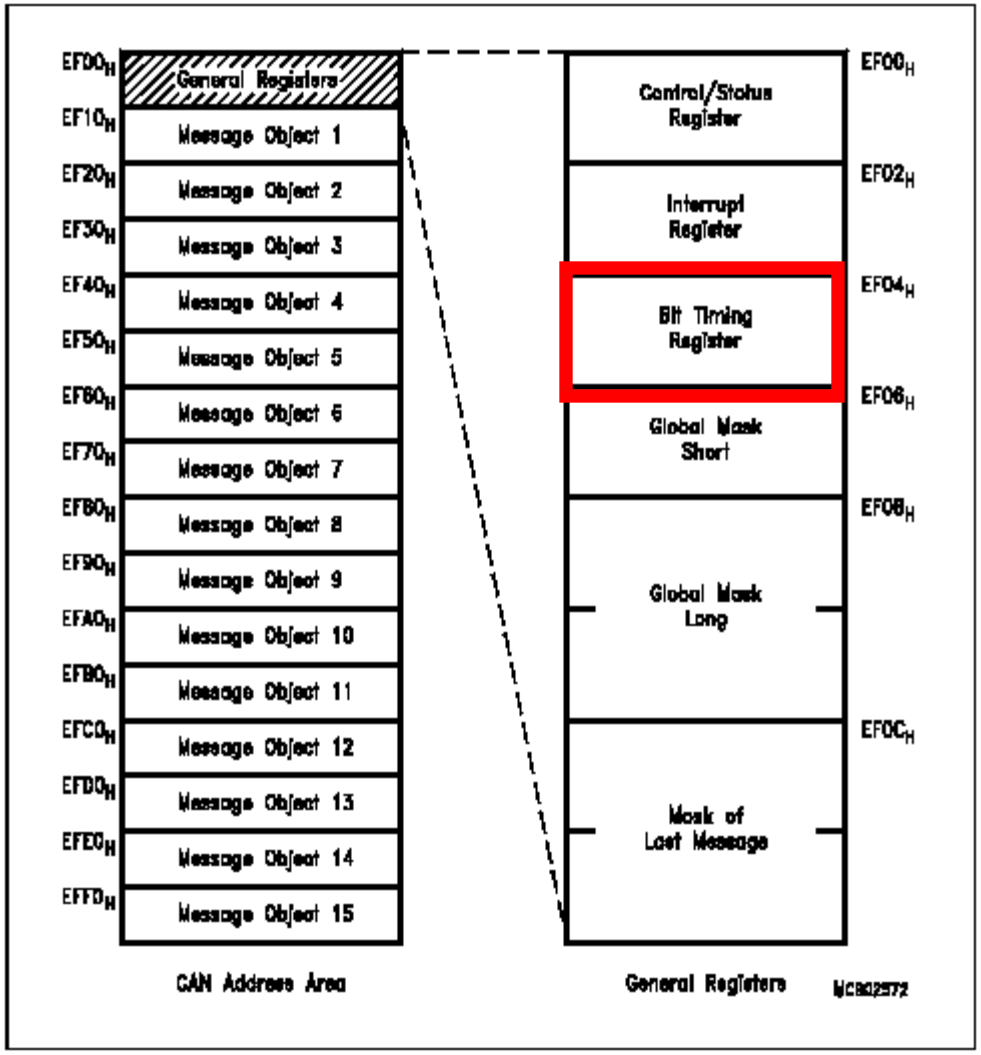


Figure 23-2
CAN Module Address Map

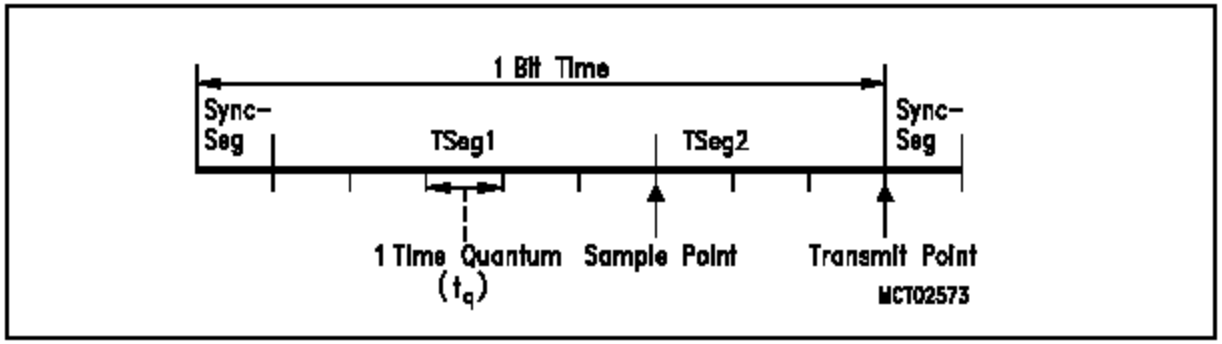


Figure 23-3
Bit Timing Definition

Bit Timing Register ($EF04_H$) **XReg** **Reset Value: $UUUU_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	TSEG2			TSEG1				SJW		BRP					
r	rw			rw				rw		rw					

Bit	Function
BRP	**Baud Rate Prescaler** For generating the bit time quanta the CPU frequency is divided by 2 * (BRP+1).
SJW	**(Re)Synchronization Jump Width** Adjust the bit time by maximum (SJW+1) time quanta for resynchronization.
TSEG1	**Time Segment before sample point** There are (TSEG1+1) time quanta before the sample point. Valid values for TSEG1 are "2...15".
TSEG2	**Time Segment after sample point** There are (TSEG2+1) time quanta after the sample point. Valid values for TSEG2 are "1...7".

CAN Module Address Map

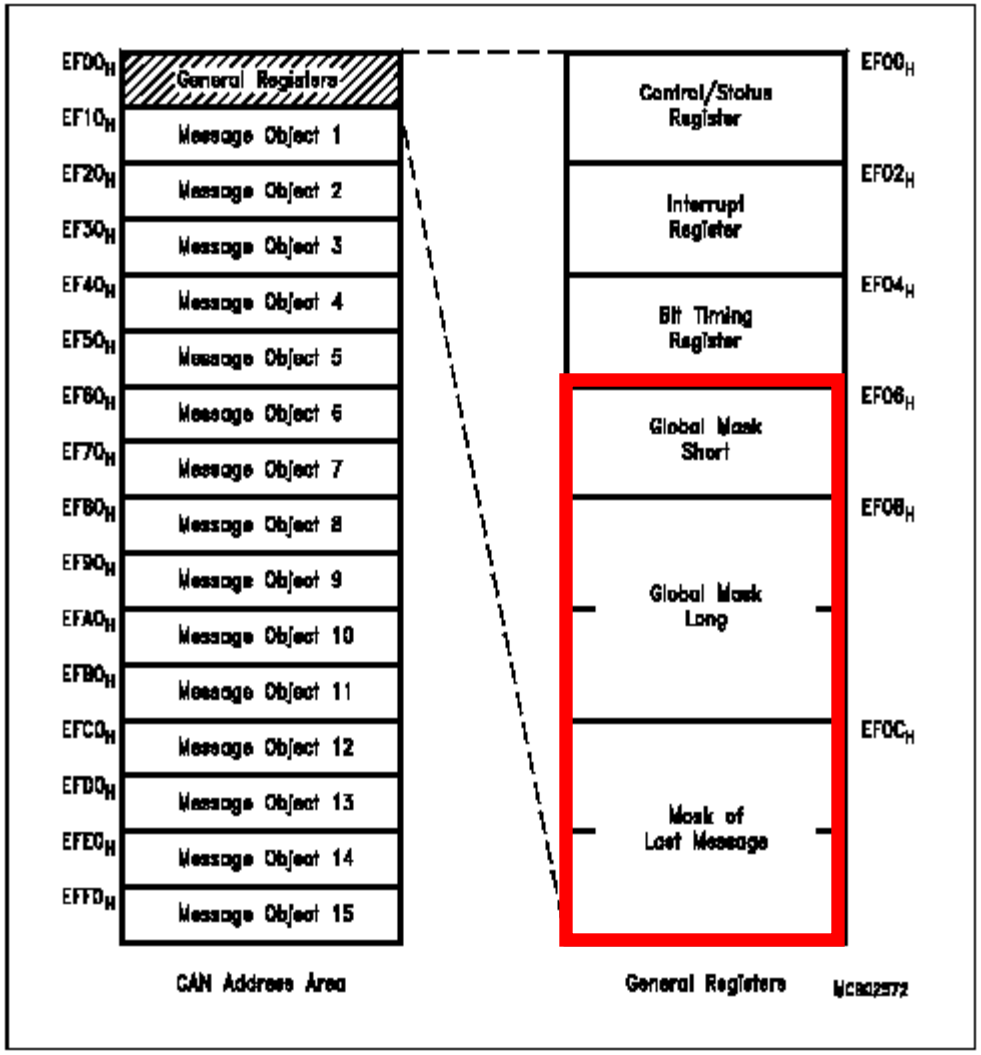
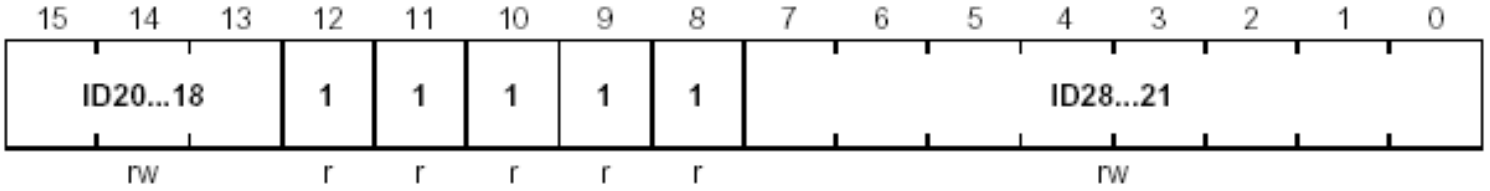


Figure 23-2
CAN Module Address Map

Mask Registers

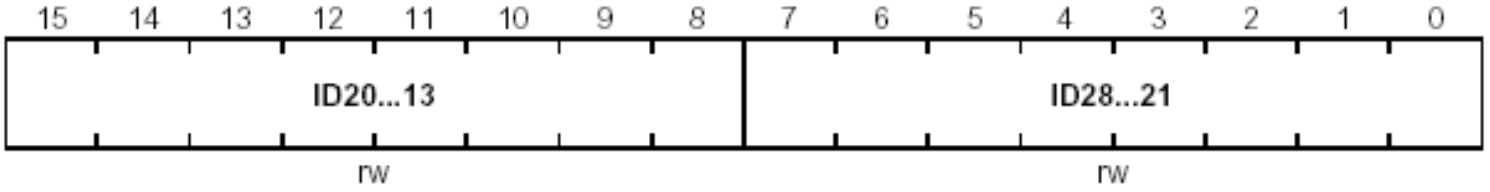
- Messages can use standard or extended identifiers. Incoming frames are masked with their appropriate global masks.
- Bit IDE of the incoming message determines, if the standard 11-bit mask in Global Mask Short is to be used, or the 29-bit extended mask in Global Mask Long.
- Bits holding a “0” mean “don’t care”, ie. do not compare the message’s identifier in the respective bit position.
- The last message object (15) has an additional individually programmable acceptance mask.

Global Mask Short ($EF06_H$) **XReg** **Reset Value: $UFUU_H$**

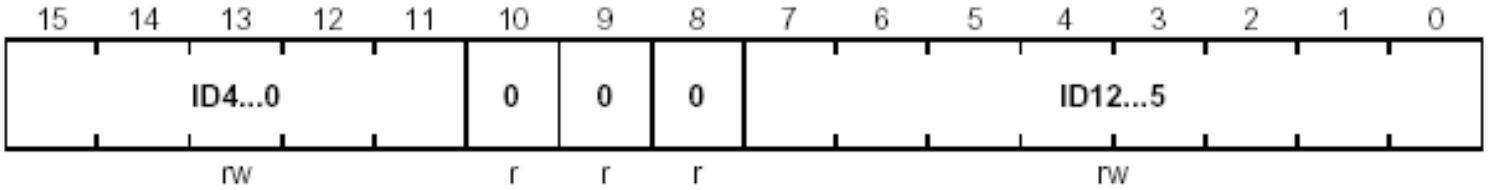


Bit	Function
ID28...18	**Identifier (11-bit)** Mask to filter incoming messages with standard identifier.

Upper Global Mask Long ($EF08_H$) **XReg** **Reset Value: $UUUU_H$**



Lower Global Mask Long ($EF0A_H$) **XReg** **Reset Value: $UUUU_H$**



CAN Module Address Map

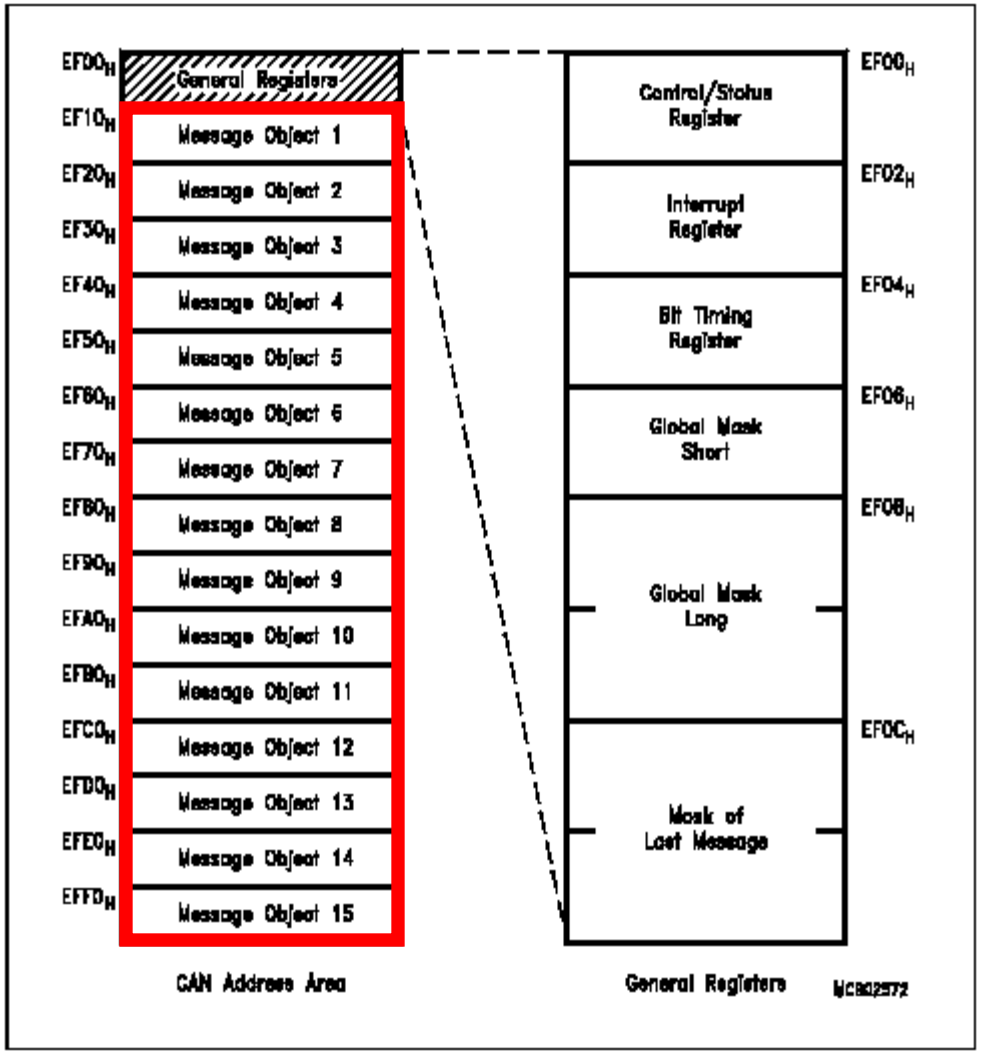


Figure 23-2
CAN Module Address Map

Message Objects

- The message object is the primary means of communication between CPU and CAN controller.
- Each of the 15 message objects uses 15 consecutive bytes (see below) and starts at an address that is a multiple of 16.

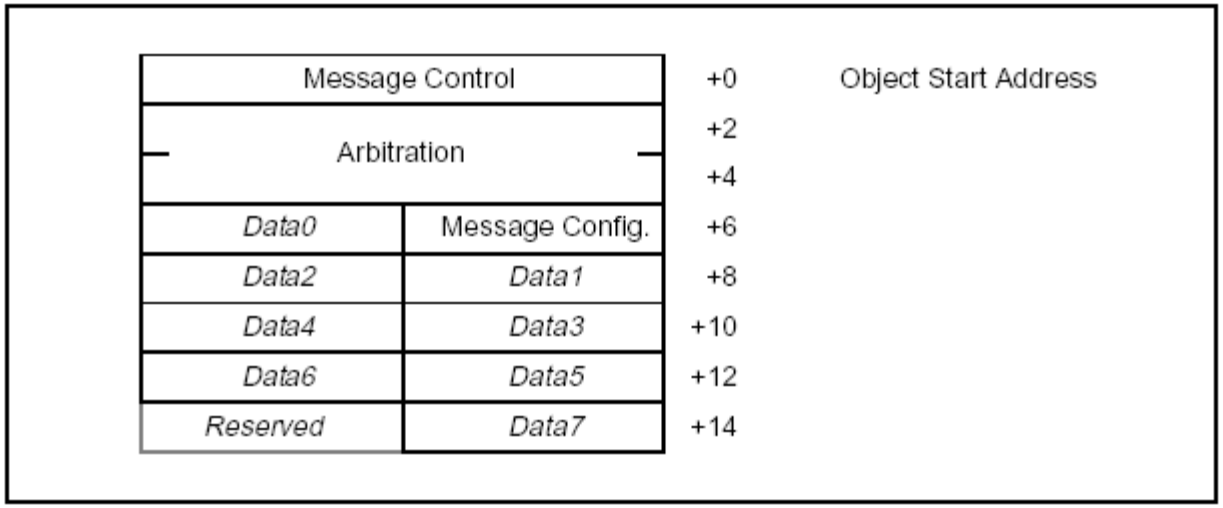


Figure 23-4
Message Object Address Map

Message Control Register

- Each element of the Message Control Register is made of two complementary bits.
- This special mechanism allows for the selective setting or resetting of specific elements (leaving others unchanged) without requiring read-modify-write cycles. None of these elements will be affected by reset. The table below shows how to use and interpret these 2-bit fields.

Value	Function on Write	Meaning on Read
0 0	-reserved-	-reserved-
0 1	Reset element	Element is reset
1 0	Set element	Element is set
1 1	Leave element unchanged	-reserved-

Message Control Register ($EFn0_H$) XReg Reset Value: $UUUU_H$

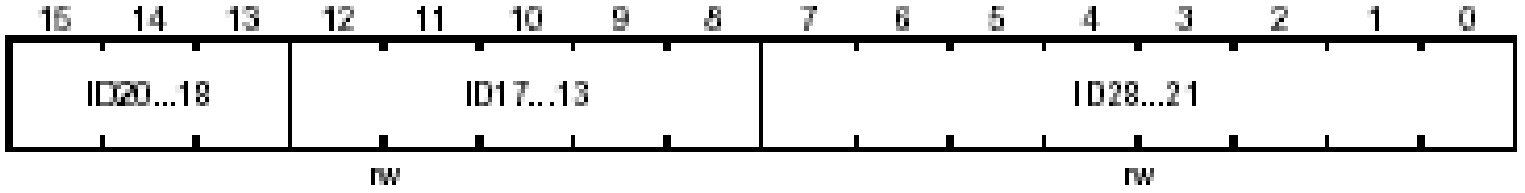
15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
RMTPND	TXRQ	MSGLST CPUUPD	NEWDAT	MSGVAL	TXIE	RXIE	INTPND
rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
INTPND	Interrupt Pending Indicates, if this message object has generated an interrupt request (see TXIE and RXIE), since this bit was last reset by the CPU, or not.
RXIE	Receive Interrupt Enable Defines, if bit INTPND is set after successful reception of a frame.
TXIE	Transmit Interrupt Enable Defines, if bit INTPND is set after successful transmission of a frame. 1)
MSGVAL	Message Valid Indicates, if the corresponding message object is valid or not. The CAN controller only operates on valid objects. Message objects can be tagged invalid, while they are changed, or if they are not used at all.
NEWDAT	New Data Indicates, if new data has been written into the data portion of this message object by CPU (transmit-objects) or CAN controller (receive-objects) since this bit was last reset, or not. 2)
MSGLST	Message Lost (This bit applies to receive-objects only!) Indicates that the CAN controller has stored a new message into this object, while NEWDAT was still set, ie. the previously stored message is lost.
CPUUPD	CPU Update (This bit applies to transmit-objects only!) Indicates that the corresponding message object may not be transmitted now. The CPU sets this bit in order to inhibit the transmission of a message that is currently updated, or to control the automatic response to remote requests.
TXRQ	Transmit Request Indicates that the transmission of this message object is requested by the CPU or via a remote frame and is not yet done. TXRQ can be disabled by CPUUPD. 1) 3)
RMTPND	Remote Pending (Used for transmit-objects) Indicates that the transmission of this message object has been requested by a remote node, but the data has not yet been transmitted. When RMTPND is set, the CAN controller also sets TXRQ. RMTPND and TXRQ are cleared, when the message object has been successfully transmitted.

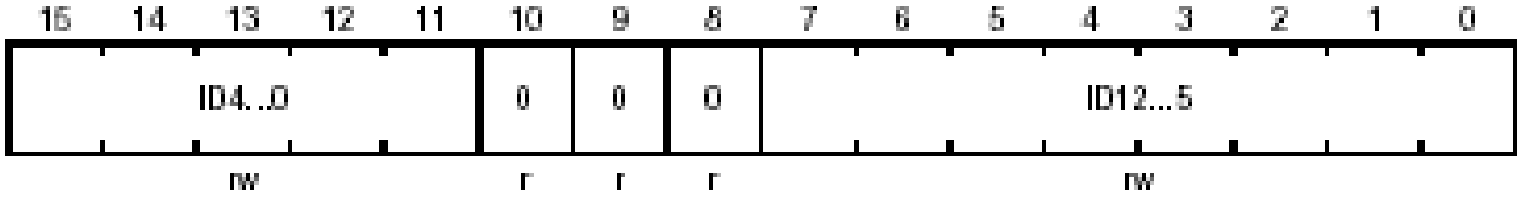
Arbitration Registers

- The Arbitration Registers are used for acceptance filtering of incoming messages and to define the identifier of outgoing messages.
- A received message is stored into the valid message object with a matching identifier and DIR=”0” (data frame) or DIR=”1” (remote frame).
- Extended frames can be stored only in message objects with XTD=”1”, standard frames only in message objects with XTD=”0”.
- For matching, the corresponding global mask has to be considered (in case of message object 15 also the Mask of Last Message).
- If a received message (data frame or remote frame) matches with more than one valid message object, it is stored into that with the lowest message number.

Upper Arbitration Register ($EFn2_H$) XReg Reset Value: $UUUU_H$



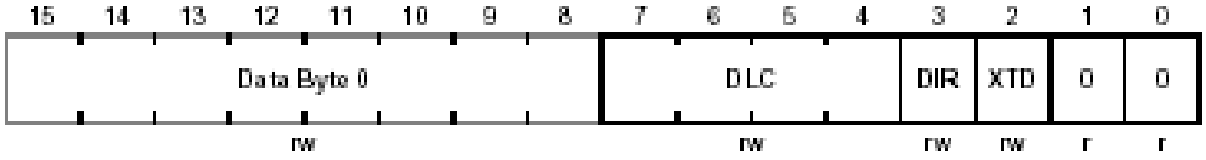
Lower Arbitration Register ($EFn4_H$) XReg Reset Value: $UUUU_H$



Bit	Function
ID28...0	Identifier (29-bit) Identifier of a standard message (ID28...18) or an extended message (ID28...0). For standard identifiers bits ID17...0 are "don't care".

Message Configuration Register

Message Configuration Register ($EFn6_H$) XReg — Reset Value: - - UU_H



Bit	Function
XTD	Extended Identifier Indicates, if this message object will use an extended 29-bit identifier or a standard 11-bit identifier.
DIR	Message Direction DIR="1": transmit. On TXRQ, the respective message object is transmitted. On reception of a remote frame with matching identifier, the TXRQ and RMTPND bits of this message object are set. DIR="0": receive. On TXRQ, a remote frame with the identifier of this message object is transmitted. On reception of a data frame with matching identifier, that message is stored in this message object.
DLC	Data Length Code Valid values for the data length are 0...8.

Data Area

The data area of message object n covers locations $00'EFn7_H$ through $00'EFnE_H$. Location $00'EFnF_H$ is reserved.

Initialization

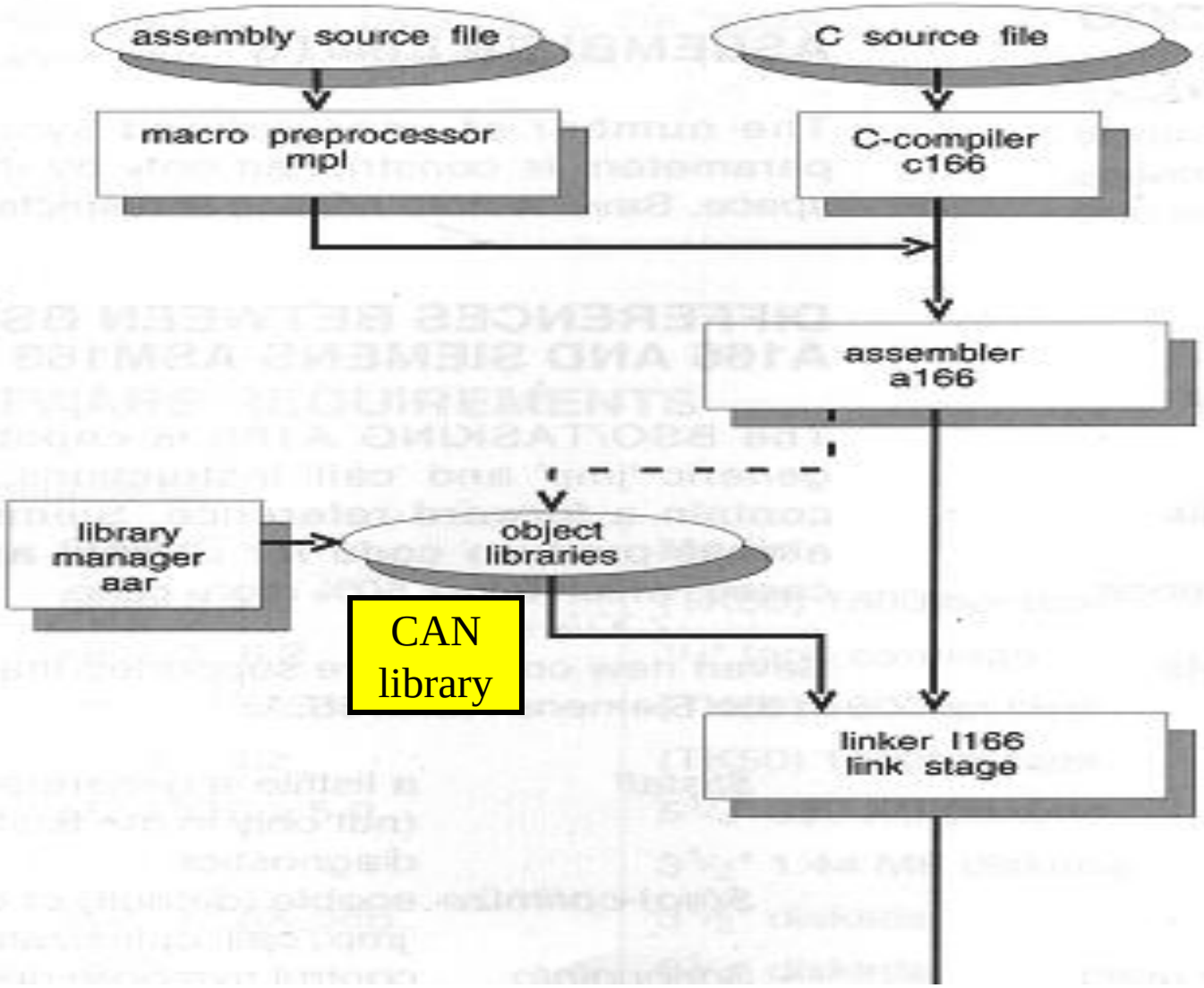
To initialize the CAN Controller, the following actions are required:

- configure the Bit Timing Register
- set the Global Mask Registers
- initialize each message object.

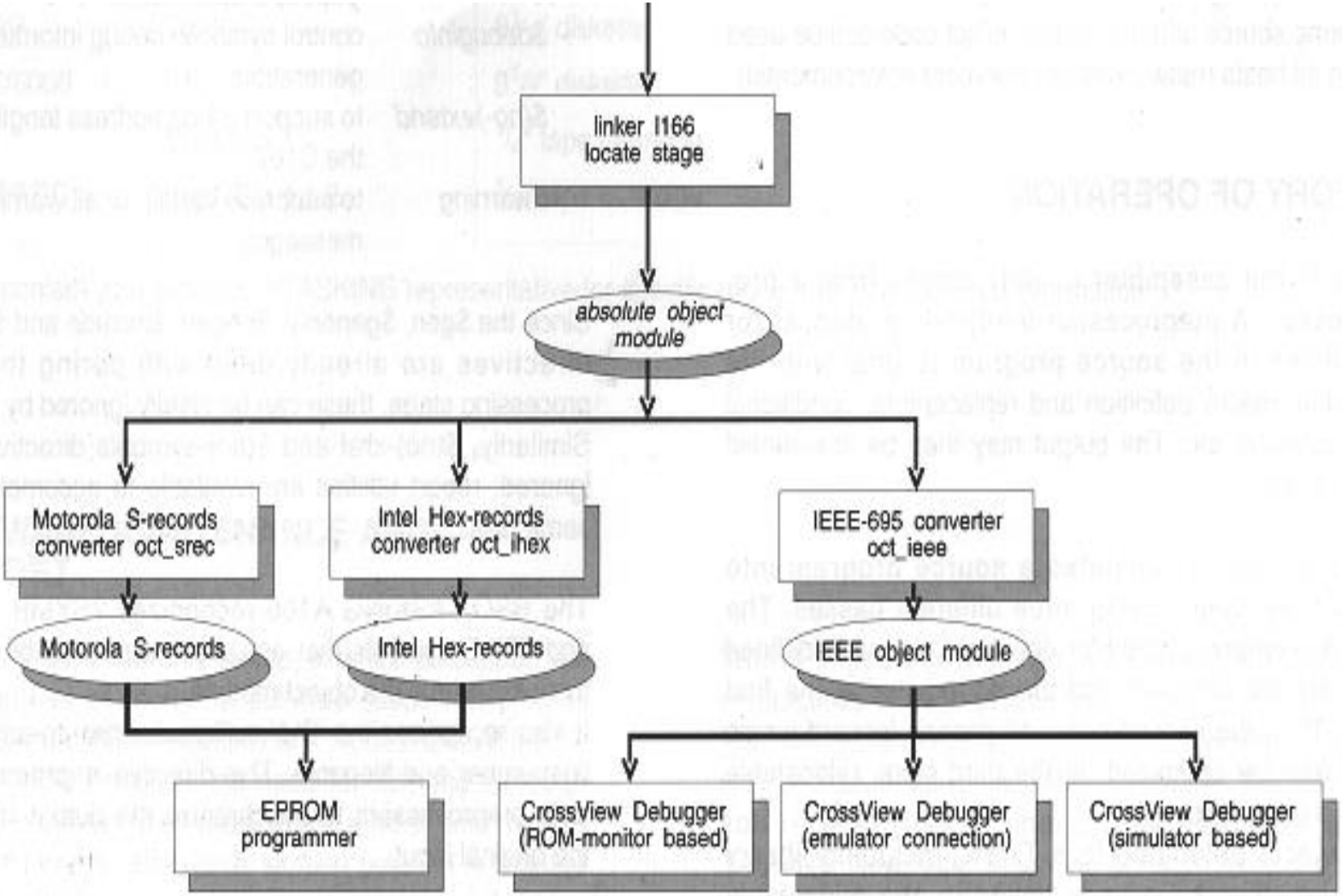
CAN Library:

- init_can_16x(baud rate, eie, sie, ie)
  - to initialize CAN Controller global settings
  - source code in appnotes/AP292201.EXE

Tasking 80C166 Tool-Chain



Tool-Chain (continued)



‘C’ CAN Driver Routines
Siemens Application Note AP292201.PDF

Initialization routine for the CAN module: init_can_16x(..)

Define a message object in the CAN module:
def_mo_16x(..)

Load the data bytes of a message object:
ld_modata_16x(..)

Read the data bytes of a message object:
rd_modata_16x(..)

Read the contents of message object 15:
rd_mo15_16x(..)

CAN Driver Routines (cont.)

Send message object:

send_mo_16x(..)

Check for new data in a message object:

check_mo_16x(..)

Check for new data or remote frame in message object 15: check_mo15_16x(..)

Check if a bus off situation has occurred and recover from bus off: check_busoff_16x(..)

Table 2-1:
Procedure overview

Procedure name:	`init_can_16x(P1, P2, P3, P4)`
Task:	initialize the global registers of the CAN module
Input parameters:	P1..P4 (see below)
Returns:	---
Name of C-source file:	`INCAN16X.C`

Table 2-2:
Input parameters

No	Meaning	Type	Possible values	Effect
P1	baud rate [kbit/s]	unsigned int	50, 125, 250, 500, 1000	Bit timing register will be loaded with the values corresponding to the selected baud rate
P2	EIE bit	unsigned char	0: 1:	• No error interrupts are generated from the CAN module to the C16x CPU. • Error interrupts are enabled.
P3	SIE bit	unsigned char	0: 1:	• No status interrupts are generated from the CAN module to the C16x CPU. • Status interrupts are enabled.
P4	IE bit	unsigned char	0: 1:	• Interrupt line from the CAN module to the C16x CPU is disabled. • Interrupt line enabled.

CAN Examples

- can204\example.c
  - send message 0x204, receive message 0x205
  - can204r\example.c - ROM monitor version
- can205\example.c
  - send message 0x205, receive message 0x204

CAN Module Address Map

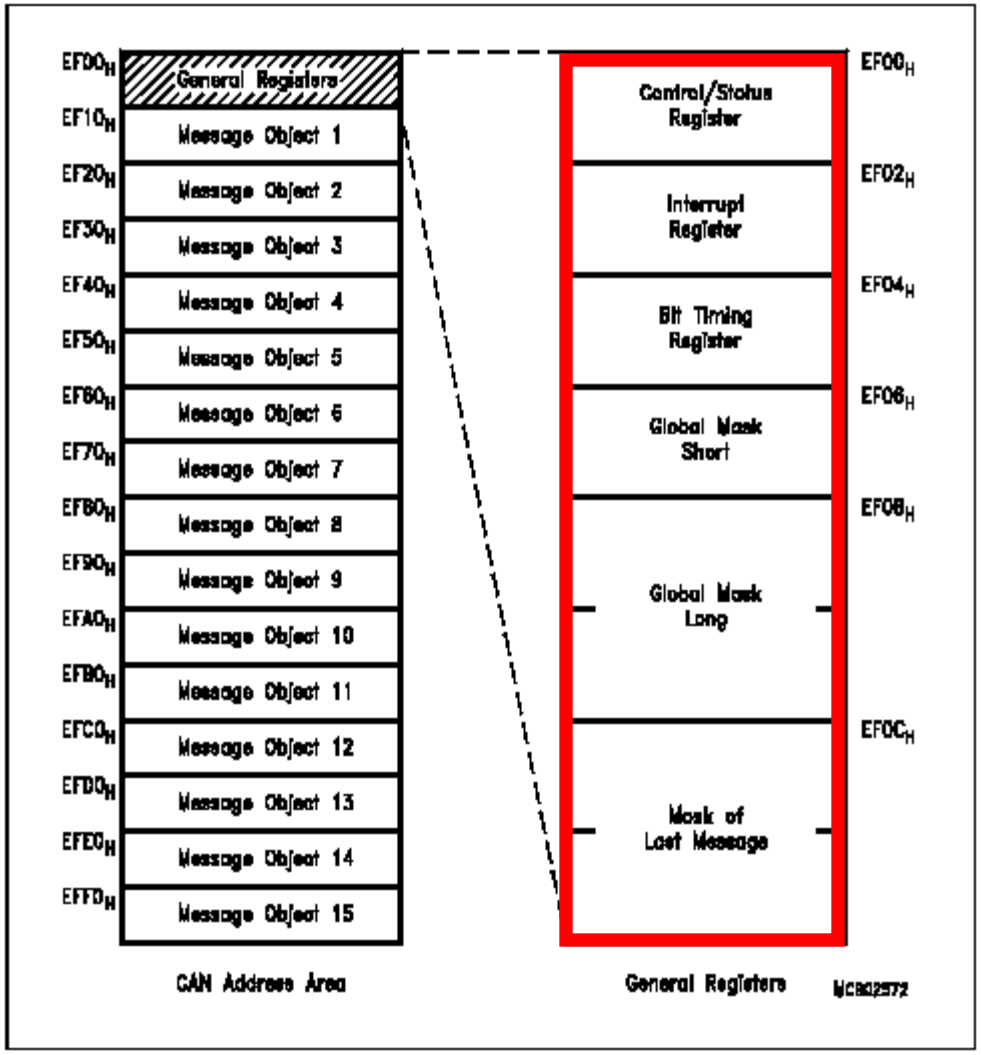


Figure 23-2
CAN Module Address Map

General CAN Registers (C:\c166\include\canr_16x.h)

```
/* Define CAN module control registers */

#define   CR      *(unsigned char*) 0xef00
#define   SR      *(unsigned char*) 0xef01
#define   IR      *(unsigned char*) 0xef02
#define   BTR     *(unsigned int *) 0xef04
#define   GMS     *(unsigned int *) 0xef06
#define   UGML    *(unsigned int *) 0xef08
#define   LGML    *(unsigned int *) 0xef0a
#define   UMLM    *(unsigned int *) 0xef0c
#define   LMLM    *(unsigned int *) 0xef0e
```

CAN Module Address Map

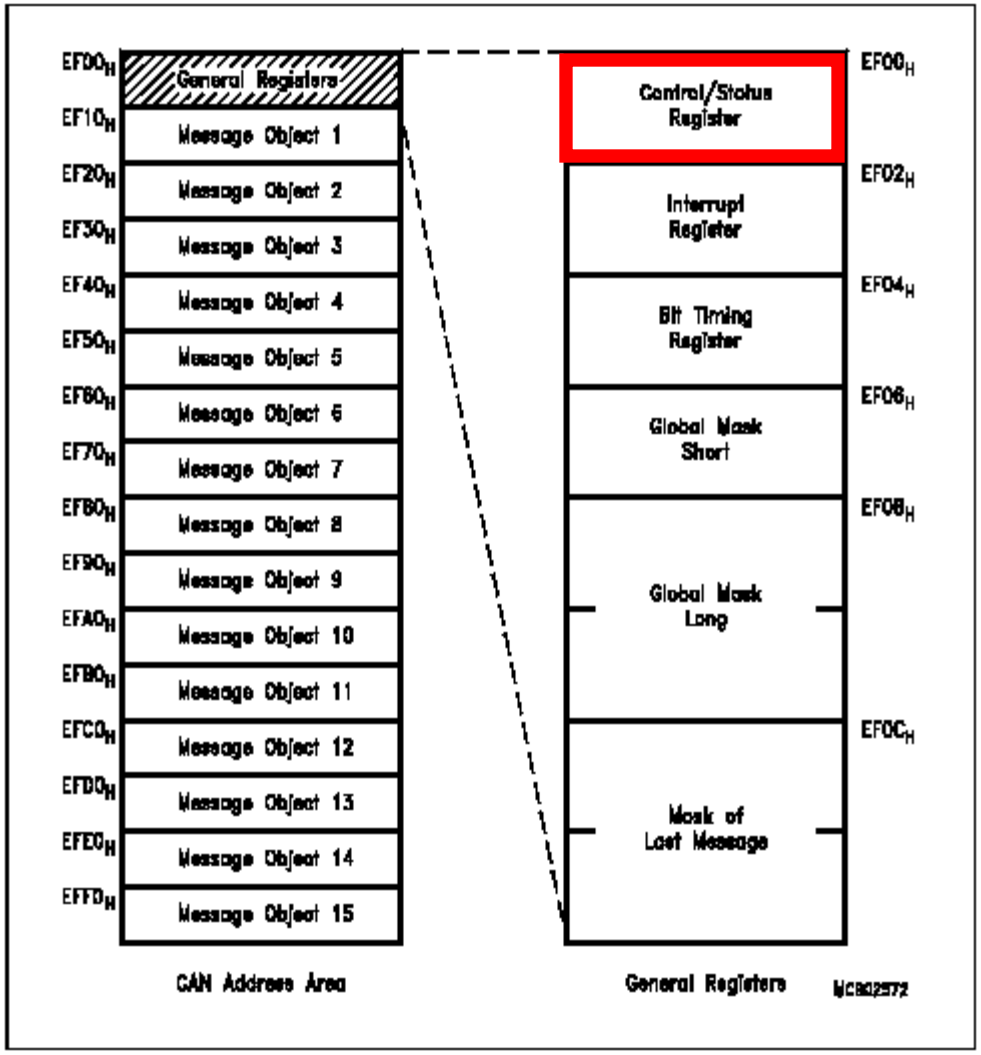


Figure 23-2
CAN Module Address Map

Control / Status Register ($EF00_H$) **XReg** **Reset Value: $XX01_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BOFF	EWRN	-	RXOK	TXOK	LEC			0 [1)]	CCE	0	0	EIE	SIE	IE	INIT
r	r	r	rw	rw	rw			rw	rw	r	r	rw	rw	rw	rw

Control Bits (CR)

Bit	Function (Control Bits)
INIT	**Initialization** Starts the initialization of the CAN controller, when set.
IE	**Interrupt Enable** Enables or disables interrupt generation from the CAN Module via the signal $\overline{\text{XINTR}}$. Does not affect status updates.
SIE	**Status Change Interrupt Enable** Enables or disables interrupt generation when a message transfer (reception or transmission) is successfully completed or a CAN bus error is detected (and registered in the status partition).
EIE	**Error Interrupt Enable** Enables or disables interrupt generation on a change of bit BOFF or EWARN in the status partition).
CCE	**Configuration Change Enable** Allows or inhibits CPU access to the Bit Timing Register.
1)	**Test Mode** (Bit 7) Make sure that bit 7 is cleared when writing to the Control Register, as this bit controls a special test mode, that is used for production testing. During normal operation, however, this test mode may lead to undesired behaviour of the device.

Example: can204i\can_msg.h

```
#define MY_IEN_BIT       1      /* General Interrupt Enable = yes                */
#define MY_BAUD_RATE 250        /* 250 kBit/s on the CAN bus                     */
#define IE_BIT           1      /* ENABLE interrupts from CAN module             */
#define EIE_BIT          0      /* No error interrupts from the CAN module       */
#define SIE_BIT          0      /* No status interrupts from the CAN module    */
                                /* Specify Message Object (MO) Features          */
#define MO1_XTD_BIT              0          /* MO1 uses 11 bit ID               */
#define MO1_ID                   0x204      /* 11-bit identifier                */
#define MO1_DIR_BIT      1       /* MO1 transmits data     */
#define MO1_DLC                  8          /* MO1 data length                  */
#define MO1_TXIE_BIT             0          /*  no transmit interrupts */
#define MO1_RXIE_BIT             0          /*  no receive interrupts   */
#define MO2_XTD_BIT              0          /* MO2 uses 11 bit ID               */
#define MO2_ID                   0x205      /* 11-bit identifier                */
#define MO2_DIR_BIT      0       /* MO2 receives data        */
#define MO2_DLC                  8          /* MO2 data length                  */
#define MO2_TXIE_BIT             0          /*  no transmit interrupts */
#define MO2_RXIE_BIT             1          /*  receive interrupts      */
```

Bit	Function (Status Bits)
LEC	**Last Error Code** This field holds a code which indicates the type of the last error occurred on the CAN bus. If a message has been transferred (reception or transmission) without error, this field will be cleared. 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:** A fixed format part of a received frame has the wrong format. 3 **AckError:** The message this CAN controller transmitted was not acknowledged by another node. 4 **Bit1Error:** During the transmission of a message (with the exception of the arbitration field), the device wanted to send a *recessive* level ("1"), but the monitored bus value was *dominant*. 5 **Bit0Error:** During the transmission of a message (or acknowledge bit, active error flag, or overload flag), the device wanted to send a *dominant* level ("0"), but the monitored bus value was *recessive*. During *busoff* recovery this status is set each time a sequence of 11 *recessive* bits has been monitored. This enables the CPU to monitor the proceeding of the busoff recovery sequence (indicating the bus is not stuck at *dominant* or continously disturbed). 6 **CRCError:** The CRC check sum was incorrect in the message received.
TXOK	**Transmitted Message Successfully** Indicates that a message has been transmitted successfully (error free and acknowledged by at least one other node), since this bit was last reset by the CPU (the CAN controller does not reset this bit!).
RXOK	**Received Message Successfully** Indicates that a message has been received successfully, since this bit was last reset by the CPU (the CAN controller does not reset this bit!).
EWRN	**Error Warning Status** Indicates that at least one of the error counters in the EML has reached the error warning limit of 96.
BOFF	**Busoff Status** Indicates when the CAN controller is in busoff state (see EML).

Note: Reading the upper half of the Control Register (status partition) will clear the Status Change Interrupt value in the Interrupt Register, if it is pending. Use byte accesses to the lower half to avoid this.

CAN Module Address Map

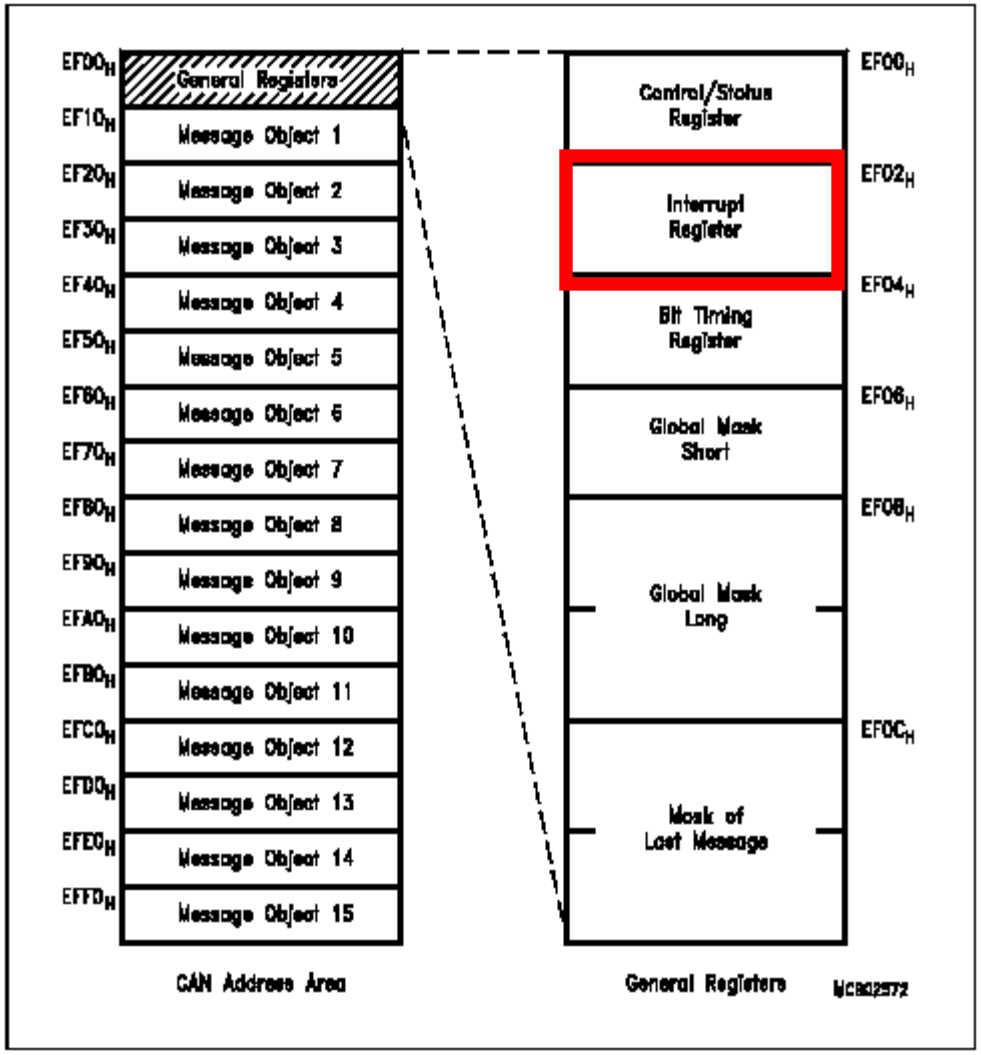
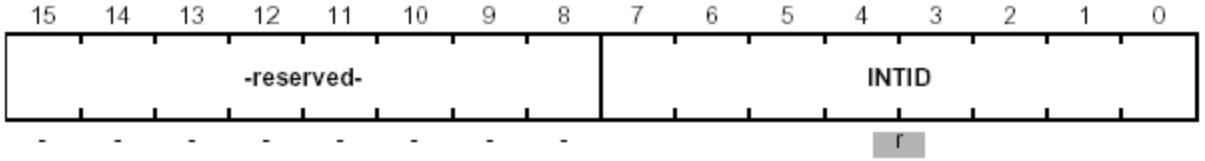


Figure 23-2
CAN Module Address Map

Interrupt Register ($EF02_H$) **XReg** **Reset Value: - - XX_H**



Bit	Function
INTID	**Interrupt Identifier** This number indicates the cause of the interrupt. When no interrupt is pending, the value will be "00".

INTID	Cause of the Interrupt
00	**Interrupt Idle:** There is no interrupt request pending.
01	**Status Change Interrupt:** The CAN controller has updated (not necessarily changed) the status in the Control Register. This can refer to a change of the error status of the CAN controller (EIE is set and BOFF or EWRN change) or to a CAN transfer incident (SIE must be set), like reception or transmission of a message (RXOK or TXOK is set) or the occurrence of a CAN bus error (LEC is updated). The CPU may clear RXOK, TXOK, and LEC, however, writing to the status partition of the Control Register can never generate or reset an interrupt. To update the INTID value the status partition of the Control Register must be read.
02	**Message 15 Interrupt:** Bit INTPND in the Message Control Register of message object 15 (last message) has been set. The last message object has the highest interrupt priority of all message objects. [1)]
(2+N)	**Message N Interrupt:** Bit INTPND in the Message Control Register of message object 'N' has been set (N = 1...14). Note that a message interrupt code is only displayed, if there is no other interrupt request with a higher priority. [1)]

CAN Module Address Map

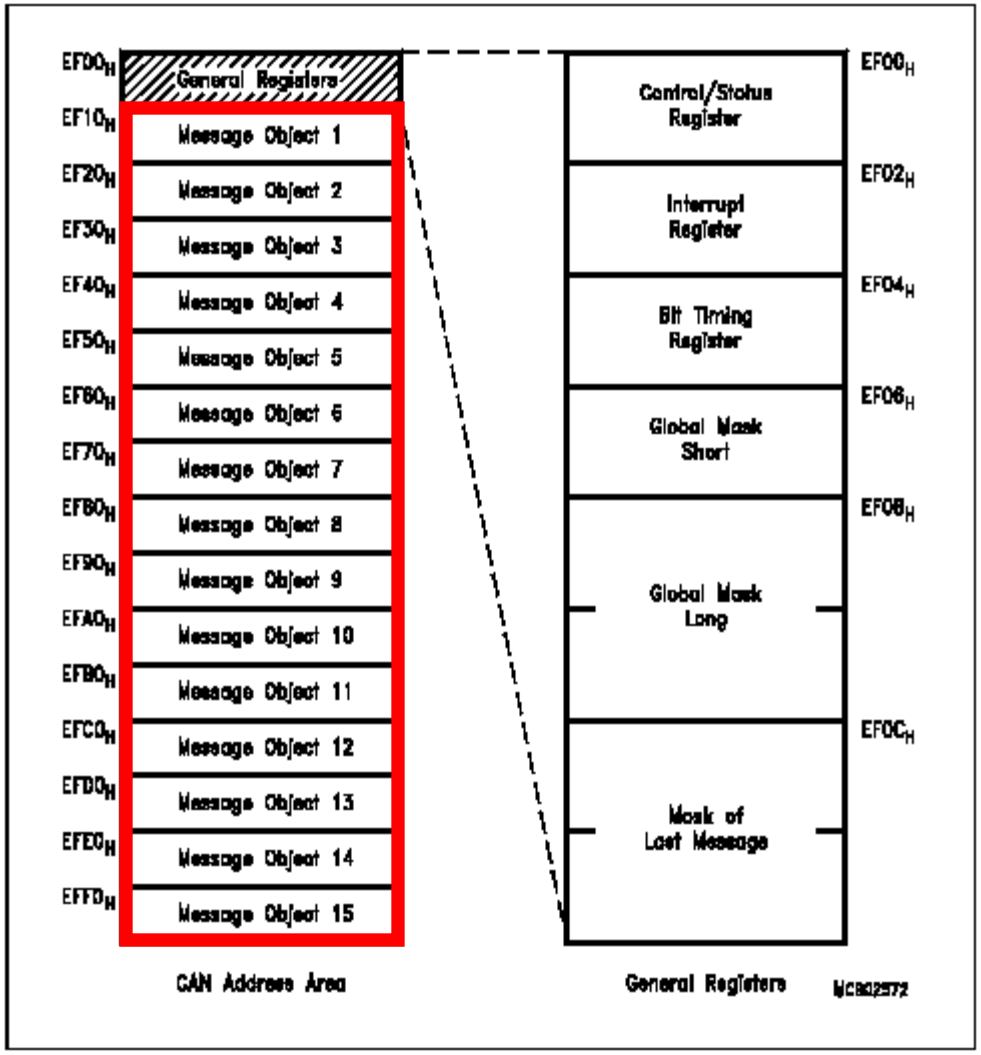


Figure 23-2
CAN Module Address Map

Message Objects

- The message object is the primary means of communication between CPU and CAN controller.
- Each of the 15 message objects uses 15 consecutive bytes (see below) and starts at an address that is a multiple of 16.

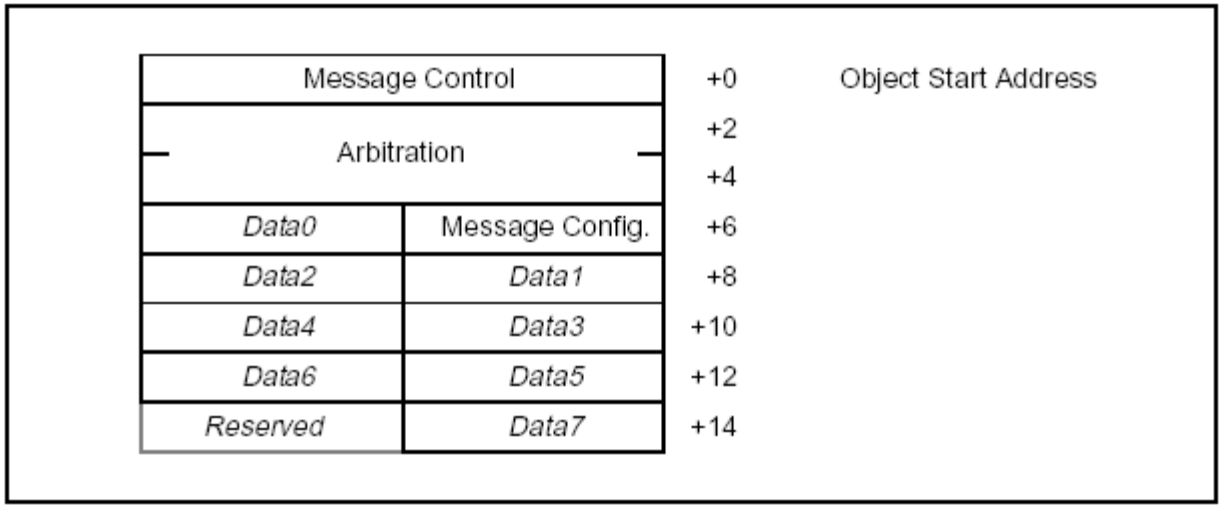


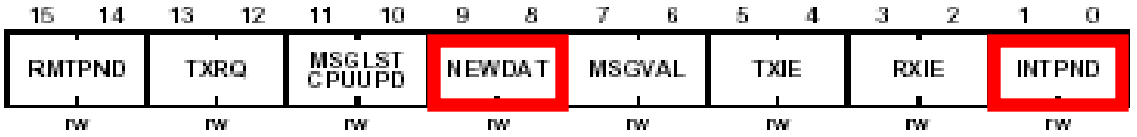
Figure 23-4
Message Object Address Map

Message Control Register

- Each element of the Message Control Register is made of two complementary bits.
- This special mechanism allows for the selective setting or resetting of specific elements (leaving others unchanged) without requiring read-modify-write cycles. None of these elements will be affected by reset. The table below shows how to use and interpret these 2-bit fields.

Value	Function on Write	Meaning on Read
0 0	-reserved-	-reserved-
0 1	Reset element	Element is reset
1 0	Set element	Element is set
1 1	Leave element unchanged	-reserved-

Message Control Register ($EFn0_H$) XReg Reset Value: $UUUU_H$



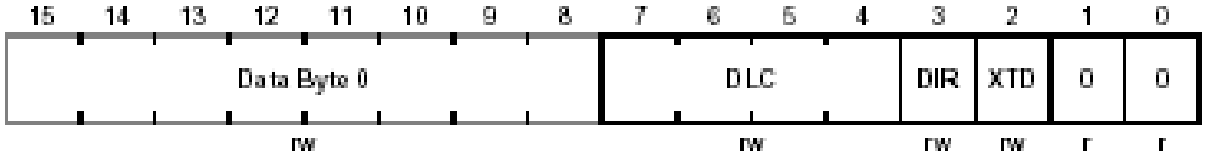
Bit	Function
INTPND	Interrupt Pending Indicates, if this message object has generated an interrupt request (see TXIE and RXIE), since this bit was last reset by the CPU, or not.
RXIE	Receive Interrupt Enable Defines, if bit INTPND is set after successful reception of a frame.
TXIE	Transmit Interrupt Enable Defines, if bit INTPND is set after successful transmission of a frame. 1)
MSGVAL	Message Valid Indicates, if the corresponding message object is valid or not. The CAN controller only operates on valid objects. Message objects can be tagged invalid, while they are changed, or if they are not used at all.
NEWDAT	New Data Indicates, if new data has been written into the data portion of this message object by CPU (transmit-objects) or CAN controller (receive-objects) since this bit was last reset, or not. 2)
MSGLST	Message Lost (This bit applies to receive-objects only!) Indicates that the CAN controller has stored a new message into this object, while NEWDAT was still set, ie. the previously stored message is lost.
CPUUPD	CPU Update (This bit applies to transmit-objects only!) Indicates that the corresponding message object may not be transmitted now. The CPU sets this bit in order to inhibit the transmission of a message that is currently updated, or to control the automatic response to remote requests.
TXRQ	Transmit Request Indicates that the transmission of this message object is requested by the CPU or via a remote frame and is not yet done. TXRQ can be disabled by CPUUPD. 1) 3)
RMTPND	Remote Pending (Used for transmit-objects) Indicates that the transmission of this message object has been requested by a remote node, but the data has not yet been transmitted. When RMTPND is set, the CAN controller also sets TXRQ. RMTPND and TXRQ are cleared, when the message object has been successfully transmitted.

Read Message Object (appnotes\RDMOD16x.c)

```
void rd_modata_16x(unsigned char nr, unsigned char *downl_data_ptr)
{
    unsigned char i, dummy_char;
    unsigned char *dummy_dbptr;
    if ((nr<15) && (nr)) {
      do {
          dummy_char=*msgconf_ptr_16x[nr];
          dlc_16x[nr]=(dummy_char>>4);  /* store actual data length code */
          *msgctrl_ptr_16x[nr]=0xfdfd;   /* clear NEWDAT and INTPND */
          dummy_dbptr=db0_ptr_16x[nr];  /* load dummy ptr (db 0) */
          for (i=0;i<dlc_16x[nr];i++) *downl_data_ptr++ = *dummy_dbptr++;
          /* move data bytes from MO's data bytes to download buffer */
      } while (*msgctrl_ptr_16x[nr] & 0x0200);        /* while NEWDAT=1 */
    }
}
```

Message Configuration Register

Message Configuration Register ($EFn6_H$) XReg — Reset Value: - - UU_H



Bit	Function
XTD	Extended Identifier Indicates, if this message object will use an extended 29-bit identifier or a standard 11-bit identifier.
DIR	Message Direction DIR="1": transmit. On TXRQ, the respective message object is transmitted. On reception of a remote frame with matching identifier, the TXRQ and RMTPND bits of this message object are set. DIR="0": receive. On TXRQ, a remote frame with the identifier of this message object is transmitted. On reception of a data frame with matching identifier, that message is stored in this message object.
DLC	Data Length Code Valid values for the data length are 0...8.

Data Area

The data area of message object n covers locations $00'EFn7_H$ through $00'EFnE_H$. Location $00'EFnF_H$ is reserved.

Read Message Object (appnotes\RDMOD16x.c)

```
void rd_modata_16x(unsigned char nr, unsigned char *downl_data_ptr)
{
    unsigned char i, dummy_char;
    unsigned char *dummy_dbptr;
    if ((nr<15) && (nr)) {
      do {
          dummy_char=*msgconf_ptr_16x[nr];
          dlc_16x[nr]=(dummy_char>>4);  /* store actual data length code */
          *msgctrl_ptr_16x[nr]=0xfdfd;   /* clear NEWDAT and INTPND */
          dummy_dbptr=db0_ptr_16x[nr];  /* load dummy ptr (db 0) */
          for (i=0;i<dlc_16x[nr];i++) *downl_data_ptr++ = *dummy_dbptr++;
          /* move data bytes from MO's data bytes to download buffer */
      } while (*msgctrl_ptr_16x[nr] & 0x0200);        /* while NEWDAT=1 */
    }
}
```

Initialize CAN Controller (appnotes\INCAN16x.c)

```
void init_can_16x(unsigned int baud_rate, unsigned char eie,
      unsigned char sie, unsigned char ie)
{
      unsigned char i, n;
      unsigned char *dummy_dbptr;
      /* Initialization PORT4 (CAN) (P4.6 to output; P4.5 to input): */
      _bfld (P4, 0x0060, 0x0060);
      _bfld (DP4, 0x0060, 0x0040);
      /* Load C167 pointers: */
      for (i=1;i<16;i++) {
         db0_ptr_16x[i] = (unsigned char *)(0xef07+i*16); /* ptrs. to data bytes 0 of MOs */
         id_ptr_16x[i] = (unsigned int *)(0xef02+i*16);      /* ptrs.  to id's of MO 1..15      */
         msgconf_ptr_16x[i] = (unsigned char *)(0xef06+i*16); /* ptrs. to msg cfg regs.*/
         msgctrl_ptr_16x[i] = (unsigned int *)(0xef00+i*16); /* ptrs. to msg cntrl regs. */
         dir_bit_16x[i] = 0;                /* clear DIR bit array */
         xtd_bit_16x[i] = 0;                /* clear XTD bit array */
         dlc_16x[i] = 0;                    /* clear data length code array */
      }
```

Initialize CAN Controller (cont.)

```
/* Load General CAN-Registers: */
CR=0x41;          /* set CCE and INIT in Control Register (EF00h) */
SR=0x00;          /* Clear Status Partition (EF01h) */

switch (baud_rate)
{
  case 50:        BTR=BTR_VALUE_50KBAUD;
                  break;
  case 125:       BTR=BTR_VALUE_125KBAUD;
                  break;
  (etc…)
}
GMS=0xe0ff;       /* Global Mask Short (EF06h) */
                  /* each bit of standard ID must match to store mess. */
UGML=0xffff;      /* Upper Global Mask Long (EF08h) */
LGML=0xf8ff;      /* Lower Global Mask Long (EF0Ah) */
                  /* each bit of extended ID must match to store mess. */
```

Initialize CAN Controller (cont.)

```
    UMLM=0x0000;     /* Upper Mask of Last Message (EF0Ch) */
    LMLM=0x0000;     /* Lower Mask of Last Message (EF0Eh) */
                     /* every message into MO 15 (Basic CAN Feature)*/

    /* reset all elements incl MSGVAL in all Message Object Ctrl. Reg.: */
    for (i=1;i<16;i++) *msgctrl_ptr_16x[i] = 0x5555;

    /* reset all data bytes in all Message Objects: */
    for (i=1;i<16;i++)
    {
      dummy_dbptr=db0_ptr_16x[i];
      for (n=0;n<8;n++) *dummy_dbptr++ = 0x00;
    }

    /* end initialization (CCE=0, INIT=0); Interrupts EIE, SIE, IE=user: */
    CR = (0x00 | (eie<<3) | (sie<<2) | (ie<<1));
}
```

‘C’ CAN Driver Routines
Siemens Application Note AP292201.PDF

Initialization routine for the CAN module: init_can_16x(..)

Define a message object in the CAN module:
def_mo_16x(..)

Load the data bytes of a message object:
ld_modata_16x(..)

Read the data bytes of a message object:
rd_modata_16x(..)

Read the contents of message object 15:
rd_mo15_16x(..)

CAN Driver Routines (cont.)

Send message object:

send_mo_16x(..)

Check for new data in a message object:

check_mo_16x(..)

Check for new data or remote frame in message object 15: check_mo15_16x(..)

Check if a bus off situation has occurred and recover from bus off: check_busoff_16x(..)

CAN Interrupt Sources

- Many different interrupt sources generate one global CAN interrupt request.
- The INTID code in the CAN Interrupt Register (EF02H) indicates which source has activated the request.
- Bit IE in the CAN Control/Status Register (EF00H) globally enables (IE=1) or disables all interrupt sources for the CAN module.

Types Of Interrupts

- Status Interrupts
- Error Interrupts
- Message Specific Interrupts

Status Interrupts

- **Status Interrupts** are generated after a status change:
  - a successful transmission (TXOK is set) or
  - reception (RXOK is set) of any message,
  - or the occurrence of an error during transmission (LEC).
- Interrupt type is indicated by the flags in the status part (high byte) of the CAN Control/Status Register
- Status Interrupts are enabled by setting bit SIE in the CAN Control/Status Register.

Error Interrupts

- **Error Interrupts** are generated after each change of the flags EWRN or BOFF which indicate the level of the two internal error counters (transmit error counter and receive error counter).
- If one of these counters reaches the value of 96, the error warning flag **EWRN** is set.
- If the send error counter exceeds the value of 255, the bus off flag **BOFF** is set. Furthermore, bit INIT is automatically set and the device stops all action on the CAN bus, it goes bus-off. A resynchronisation on the CAN bus can be achieved by resetting bit INIT by software.
- Error Interrupts are enabled by setting bit EIE.

Message Specific Interrupts

- **Message Specific Interrupts** are generated by each message object after successful transmission or reception.
- They are enabled by setting bits TXIE and/or RXIE in the corresponding CAN Message Control Register located at address EFn0H, with n (1..15) being the number of the corresponding message object.

Define Message Objects (appnotes\DEFMO16x.c)

```
void def_mo_16x(unsigned char nr, unsigned char xtd, unsigned long id,
     unsigned char dir, unsigned char dlc, unsigned char txie, unsigned char rxie)
{
     unsigned int dummy_int;
     unsigned int *dummy_idptr;
     if ((nr<16) && (nr))
     {
        // Load Arbitration Register(s) for Message Object nr
         // Prepare Message Control Register
        /* prepare Message Control Register: */
        if (txie==1) (txie=0x20); else (txie=0x10);
        if (rxie==1) (rxie=0x08); else (rxie=0x04);
        if (dir==1) dummy_int = (0x5981 | txie | rxie);  /* CPUUPD set       */
        else dummy_int = (0x5581 | txie | rxie);     /* MSGLST reset         */
        *msgctrl_ptr_16x[nr]=dummy_int;   /* Load Mess. Contr. Reg.      */
```

Define Message Object (cont.)

```
    /* prepare Message Configuration Register: */
    if (dlc>8) dlc=8;
    dlc_16x[nr]=dlc;
    dir_bit_16x[nr]=dir;
    xtd_bit_16x[nr]=xtd;
    *msgconf_ptr_16x[nr] = (dlc<<4) + (dir<<3) + (xtd<<2);
}
```

INTID Code

- An INTID code of **0** indicates that all requested interrupts have been correctly serviced and no more interrupts are pending. This should be the condition to leave the CAN interrupt service routine.
- An INTID value of **1** indicates a status interrupt (if enabled by SIE) or an error interrupt (if enabled by EIE), which cause the interrupt with the highest priority. In the case of a status change due to a successful message transfer, one of the flags TXOK or RXOK in the CAN Status Register is set.
- An erroneous message transfer is indicated by the LEC bit field. In the case of an error interrupt, at least one of the error flags EWRN and BOFF has changed.

INTID Code (cont.)

- An INTID code of **2** indicates the reception of a message by msg. object 15.
- An INTID code of **3..16** indicates a message specific transmit (if TXIE set) or receive (if RXIE set) interrupt concerning the message objects 1 to 14 (e.g., INTID - 2). In addition, the global flags TXOK and RXOK can be checked to decide if it was an interrupt on a received or on a transmitted message.
- A successful message transfer sets bit INTPND in the corresponding Message Control Register, which must be cleared by software to reset this interrupt request (recall read message object code).

General Interrupt Handling

- The status part (high byte) of the CAN Control/Status Register must be read in the interrupt service procedure in order to identify the interrupt source and to reset the pending interrupt request. Flags TXOK and RXOK in this part of the register must be cleared by software.
- The priority of the internal CAN interrupt sources decreases with an increasing INTID code. This structure must also be taken into account for the identification of the interrupt source. Successful transmission of only one message object can cause two independent interrupt requests if bit SIE and the corresponding bit TXIE have been set. If the status interrupt (highest priority) is serviced and bit INTPND of this message object is not cleared, the message specific interrupt remains pending.

```
// CAN interrupt service routine
interrupt(0x40)
void int_can (void)
{
  unsigned char status, intid;
  while (intid = IR)
  {
    status = SR; SR = 0; // read and reset CAN status register
    switch (intid)
    {
      case 1: // status and error interrupt
              if (status & 0x04) // status interrupts
              {
                if (status & 0x08) {...} // transmit interrupt
                if (status & 0x10) {...} // receive interrupt
                if (status & 0x07) {...} // error interrupt
              }
              if (status & 0x08) // error interrupts
              {
                if (status & 0x40) {...} // EWRN has changed
                if (status & 0x80) {...} // BUSOFF has changed
              }
              break;
      case 2: // message 15 interrupt
              {...} // see special chapter
              break;
```

```
    case 3: // message 1 interrupt
            ..
            if (status & 0x08) {...} // transmit interrupt
            if (status & 0x10) {...} // receive interrupt
            break;

    case 16: // message 14 interrupt
            ..
            if (status & 0x08) {...} // transmit interrupt
            if (status & 0x10) {...} // receive interrupt
            break;
    }
  }
}
```

CAN Examples

- can204\example.c
  - send message 0x204, receive message 0x205
- can205\example.c
  - send message 0x205, receive message 0x204
- can204r\example.c
  - ROM monitor version
- **can204i\example.c**
  - **interrupt-driven receives version**

Time-Triggered vs. Event-Triggered

- Time-triggered control system
  - All activities are carried out at certain points in time know a priori.
  - All nodes have a common notion of time, based on approximately synchronized clocks.
- Event-triggered control system
  - All activities are carried out in response to relevant events external to the system.

Time-Triggered Protocols

- Using Hardware and Software
  - TTCAN - Time-Triggered CAN
  - TTP/C - Time-Triggered Protocol (Vers. C)
- Using Software
  - TTP/A - Time-Triggered Protocol (Vers. A)
  - Time Herald in CAN Kingdom

Summary

- Next Time: Global Clocks and Time-Triggered Protocols