CIS 721 - Real-Time Systems

Lecture 17: CANKingdom and Common Digital Architecture

Mitch Neilsen
neilsen@cis.ksu.edu

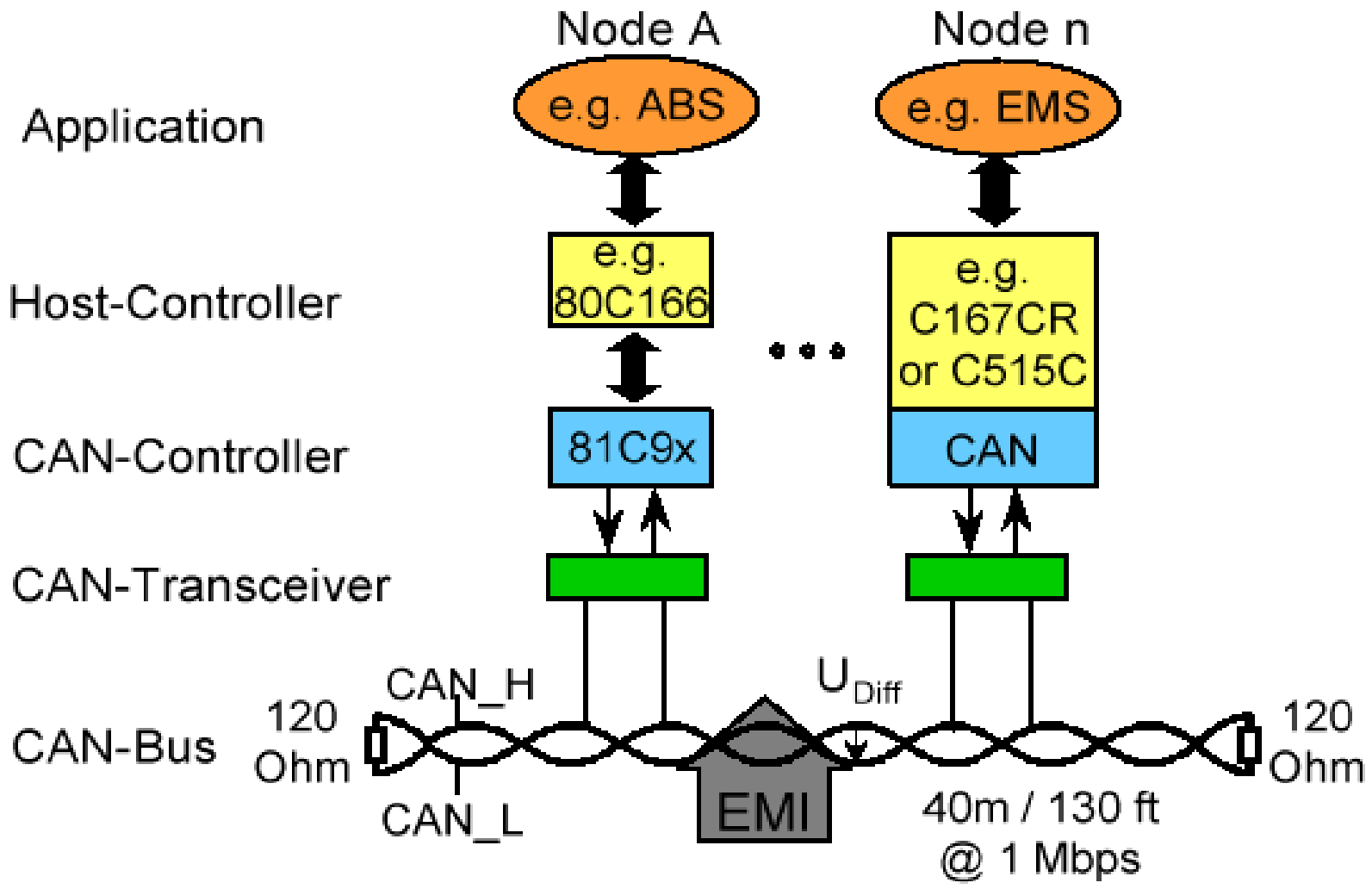
Outline

- Controller Area Network (CAN)
  - Infineon C167CR
- Higher-Level CAN Protocols
  - CANKingdom (Kvaser, CKI)
  - Common Digital Architecture (CDA 101)
- Implementation
  - Tasking Development Environment
  - CDA Example: City Lights
- Summary

Controller Area Network (CAN)

- A Controller Area Network is an advanced serial bus protocol that efficiently supports distributed, real-time **control**.
- Originally developed for use in automobiles by a German company, **Bosch** GmbH, in the late 1980s.
- CAN is internationally standardized by the International Standardization Organization (ISO) in **ISO 11898**.
- Several Higher-Level Standards are based on CAN including **CANKingdom** and the Common Digital Architecture (**CDA101**) which is based on CANKingdom.

Infineon C167CR CAN Controller



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** set as **either transmit or receive --** the last message object (object 15) only has a **double receive buffer** with a special mask register.

Registers and Message Objects

- Applications communicate with the CAN controller by accessing a set of **hardware registers**.
- 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. All registers may be accessed bytewise in order to select special actions without affecting other mechanisms.

CAN Module Address Map

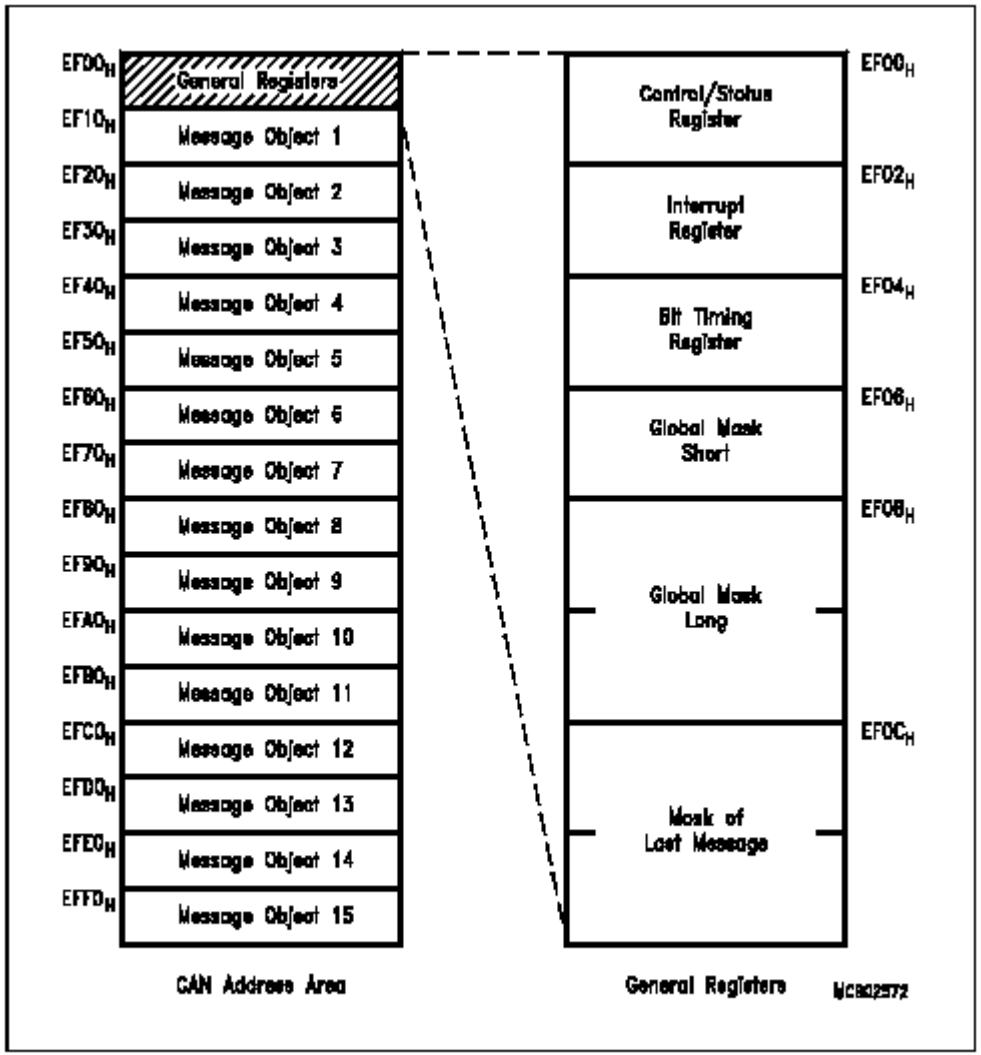


Figure 23-2
CAN Module Address Map

#include <canr16x.h> (\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

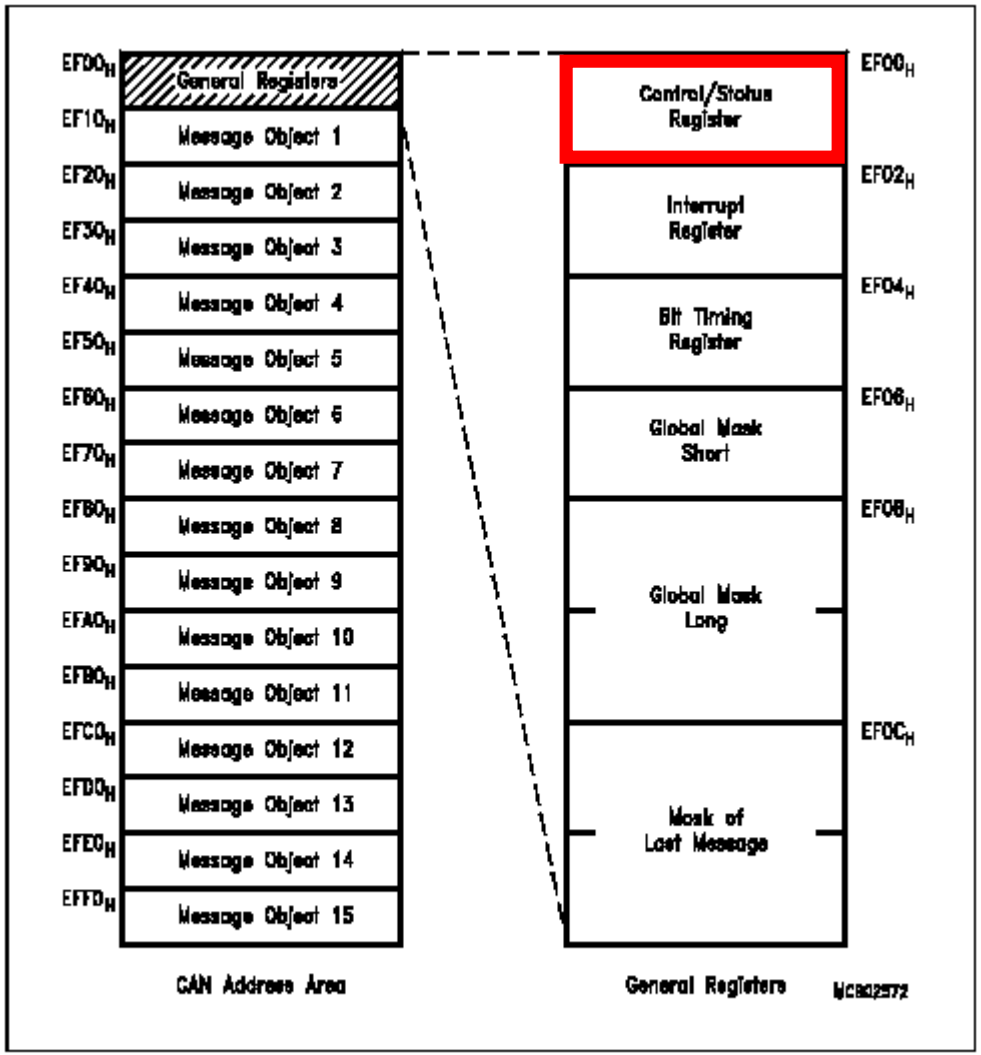


Figure 23-2
CAN Module Address Map

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	E WRN	-	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

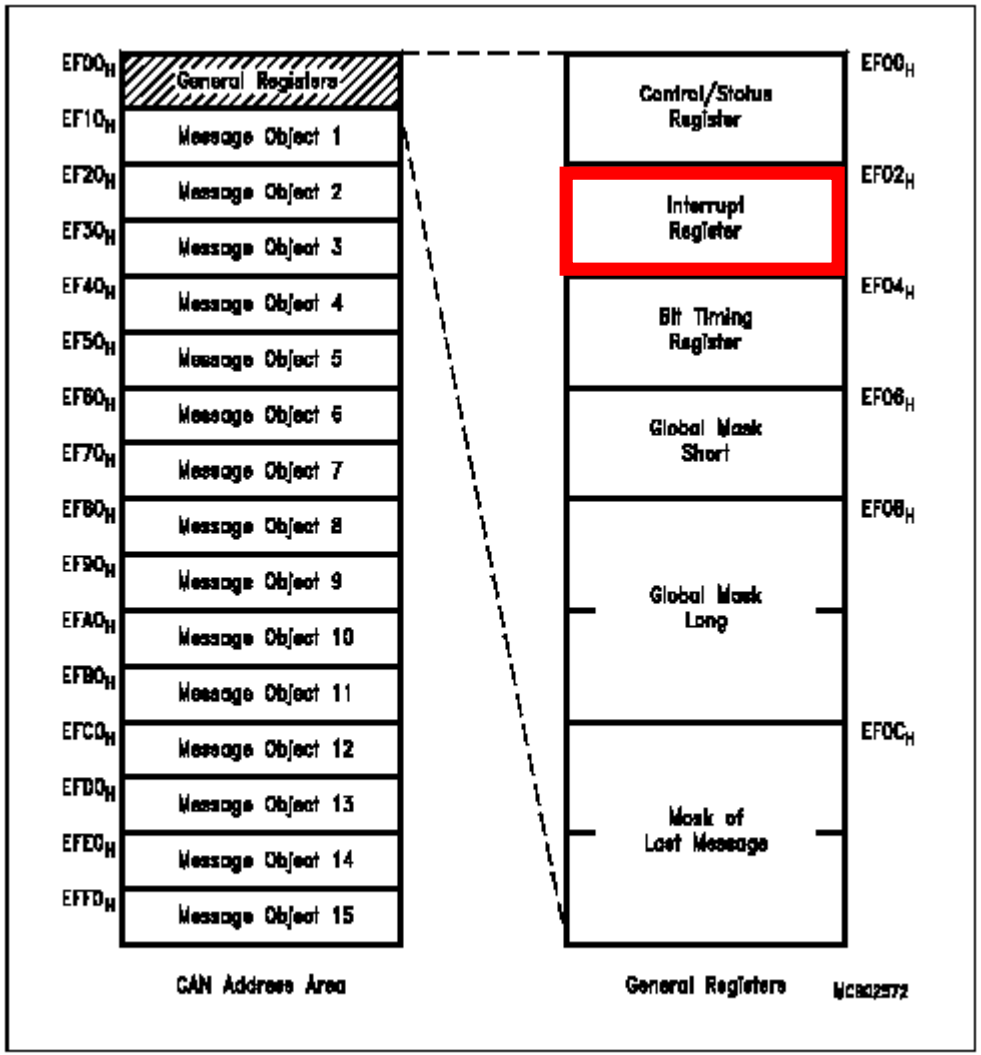
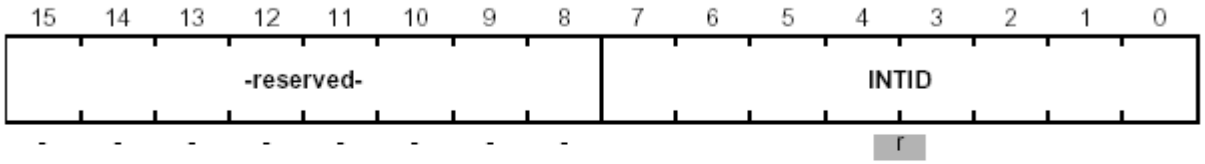


Figure 23-2
CAN Module Address Map

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

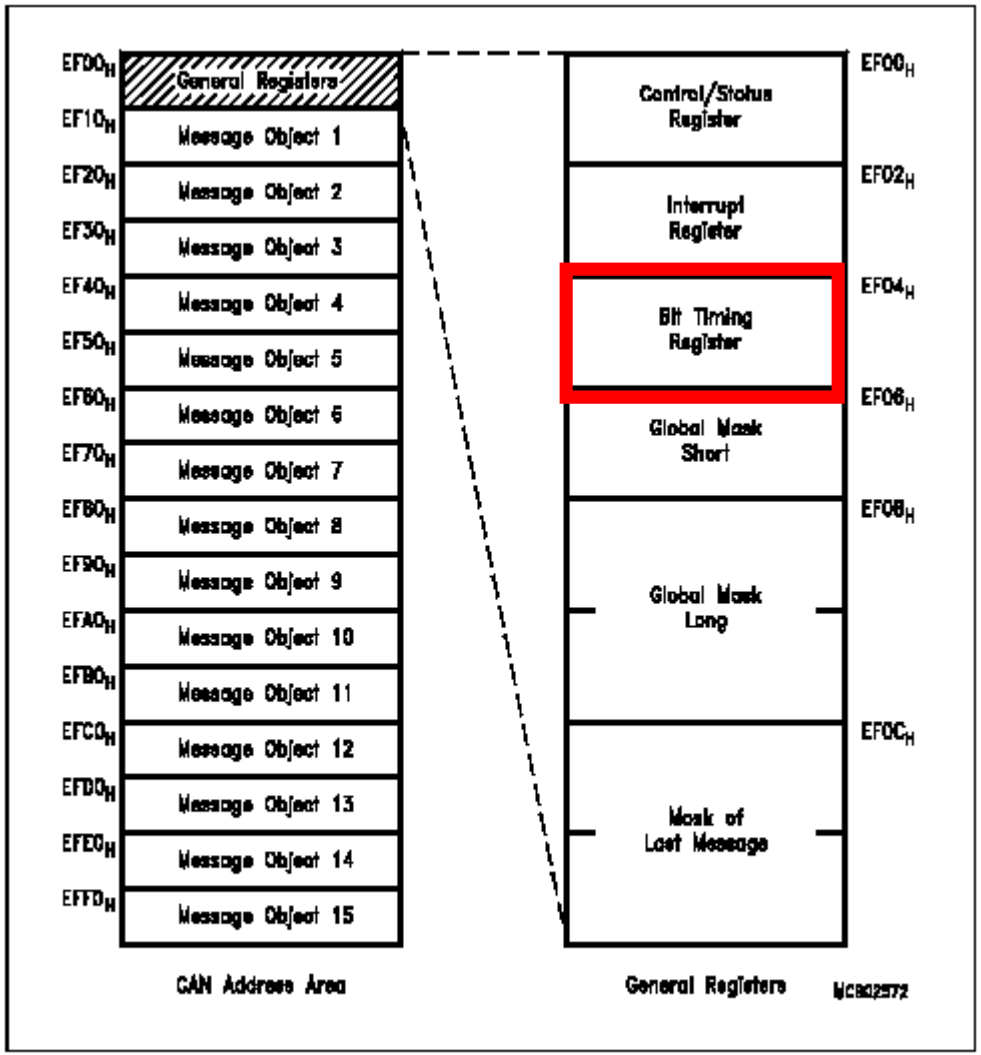


Figure 23-2
CAN Module Address Map

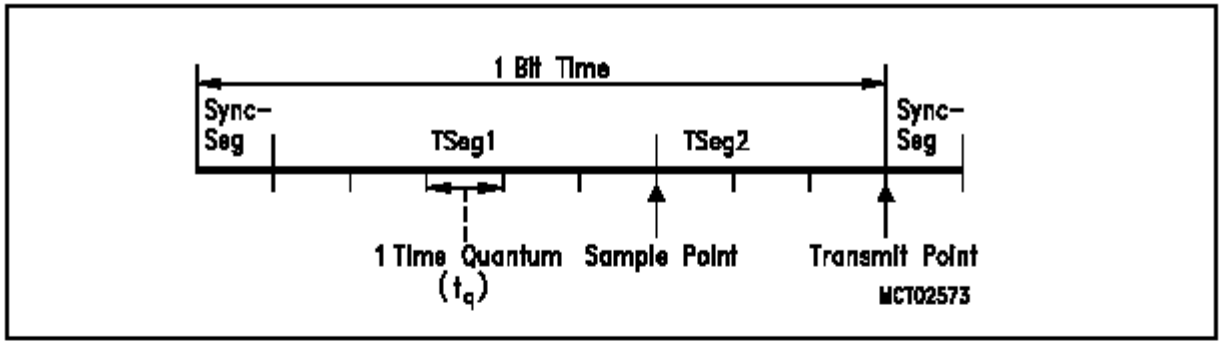


Figure 23-3
Bit Timing Definition

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

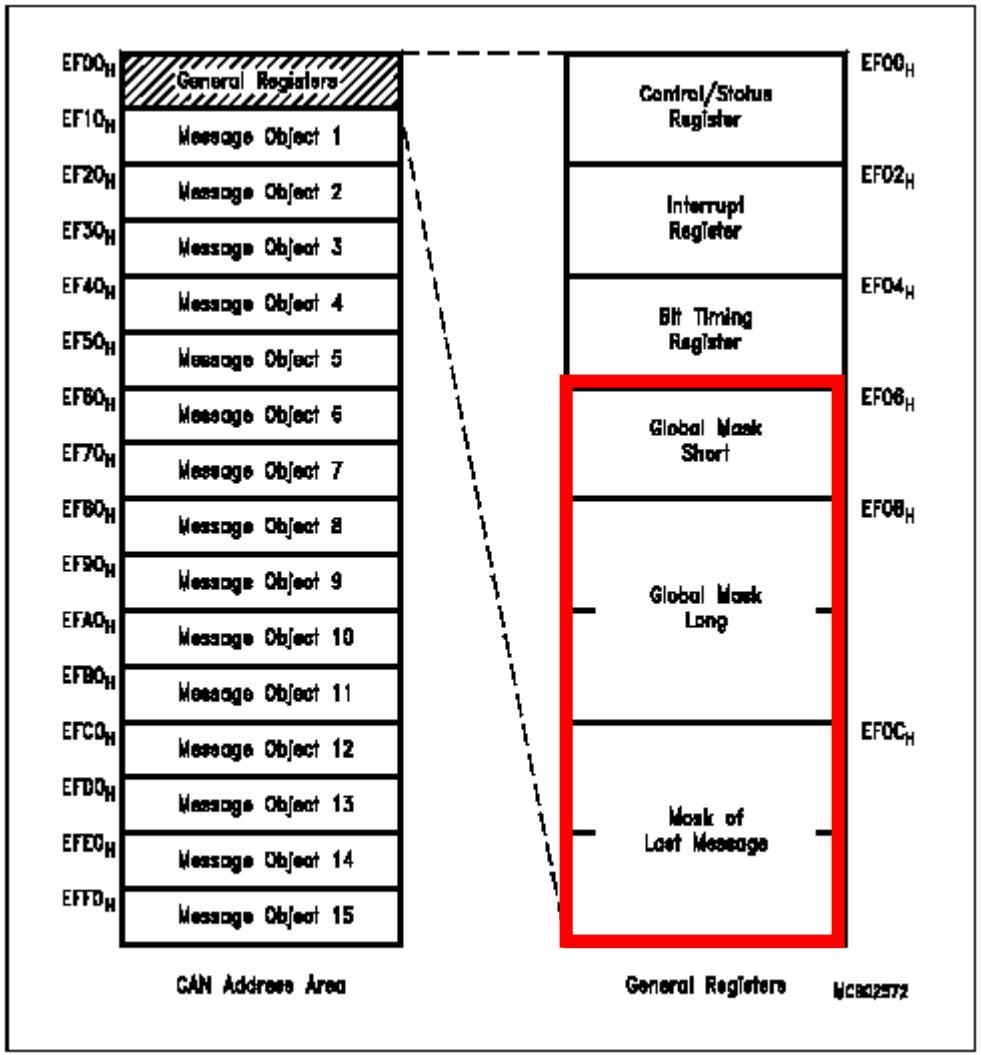
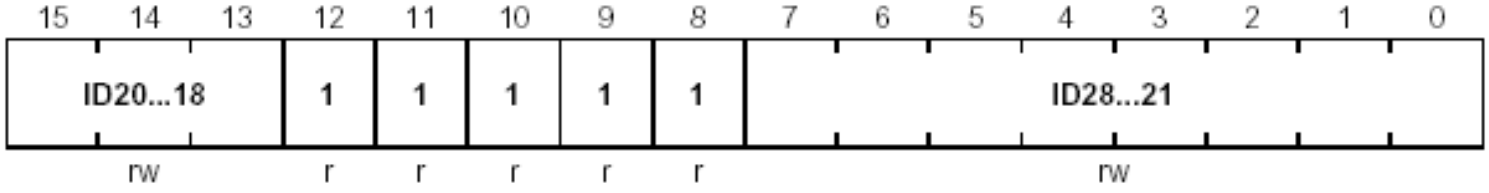


Figure 23-2
CAN Module Address Map

Mask Registers

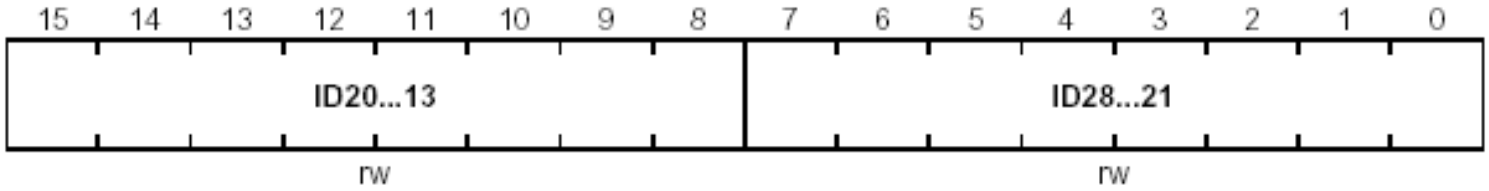
- Messages can use standard or extended identifiers. Incoming frames are **masked (filtered)** with their appropriate global masks.
- Bit IDE of the incoming message determines, if the standard 11-bit mask in Global Mask Short is to be used, or the 29-bit extended mask in Global Mask Long.
- Bits holding 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: $UFUU_H$**

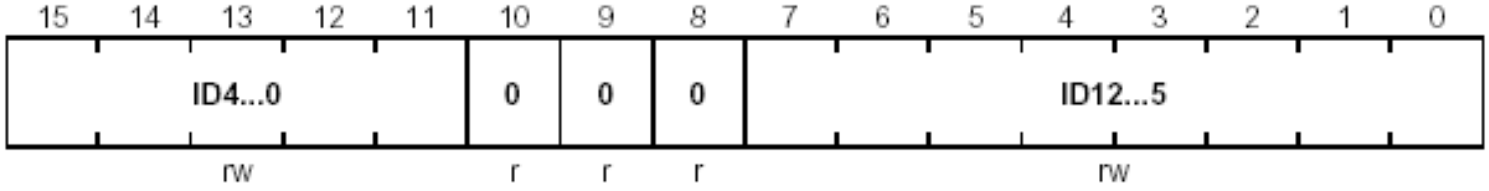


Bit	Function
ID28...18	**Identifier (11-bit)** Mask to filter incoming messages with standard identifier.

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

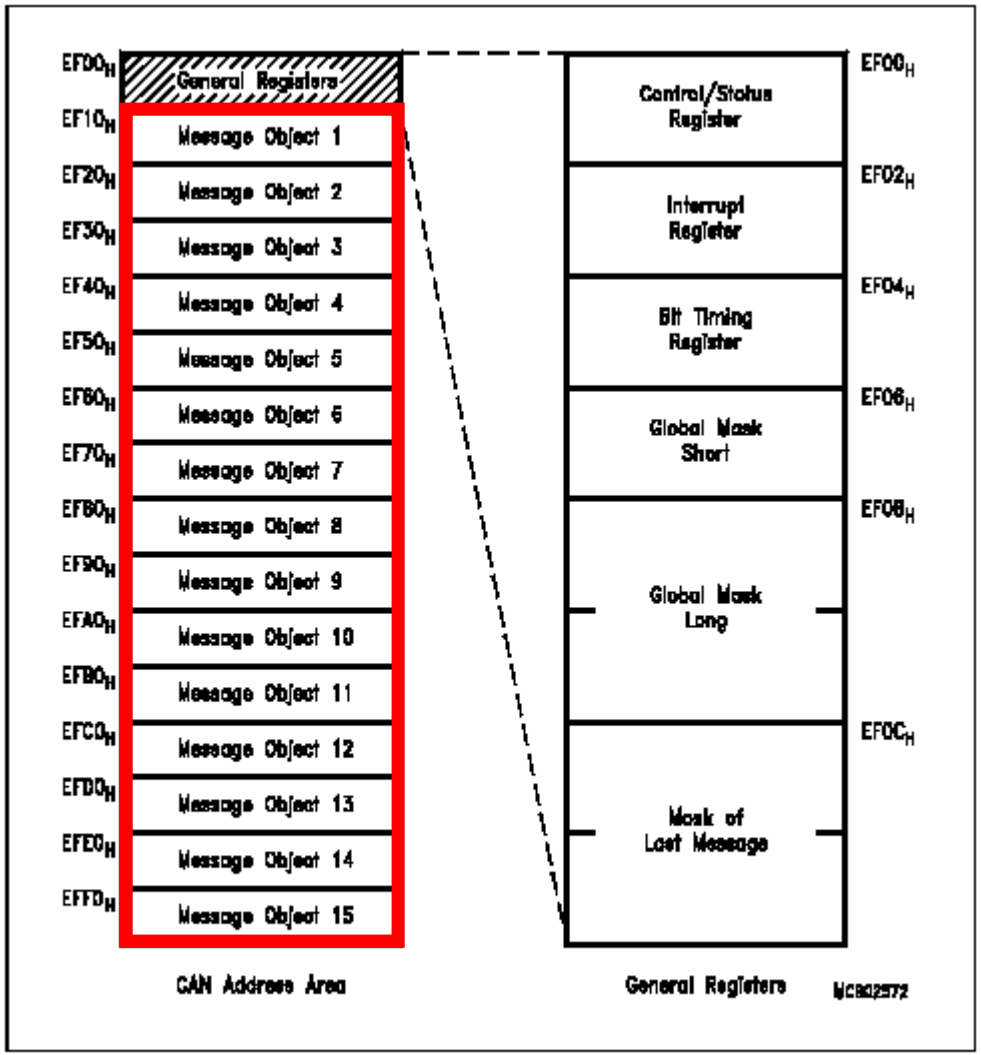


Figure 23-2
CAN Module Address Map

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.

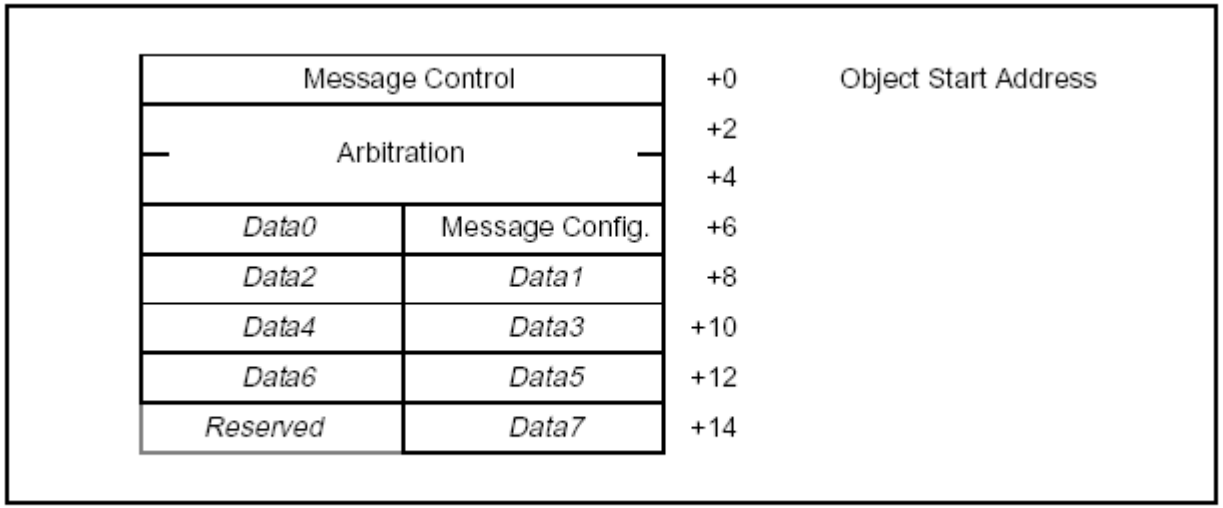


Figure 23-4
Message Object Address Map

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

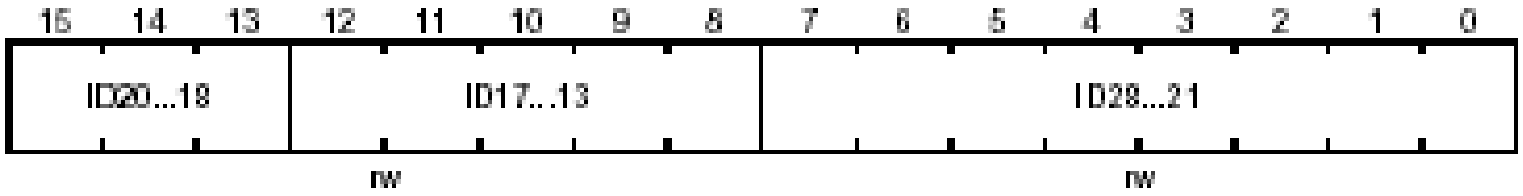
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RMTPND		TXRQ		MSGLST CPUUPD		NEWDAT		MSGVAL		TXIE		RXIE		INTPND	
rw		rw		rw		rw		rw		rw		rw		rw	

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.

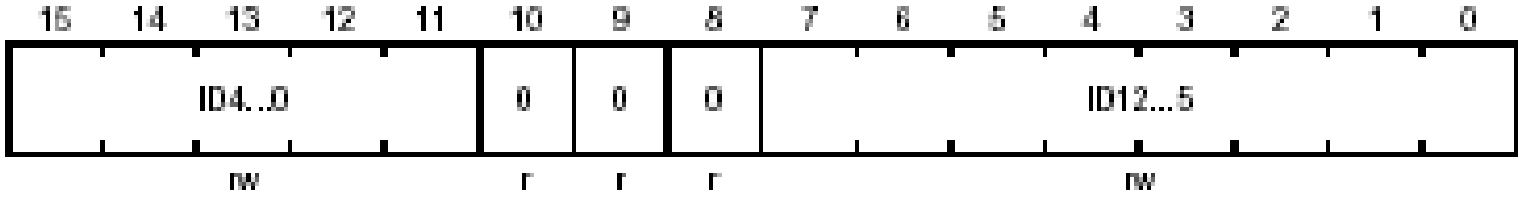
Arbitration Registers

- 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”.
- If a received message (data frame or remote frame) matches with more than one valid message object, it is stored into the buffer with the lowest number.

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



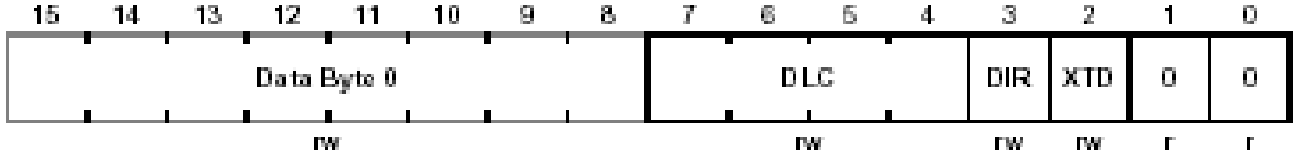
Lower Arbitration Register ($EFn4_H$) XReg Reset Value: $UUUU_H$



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

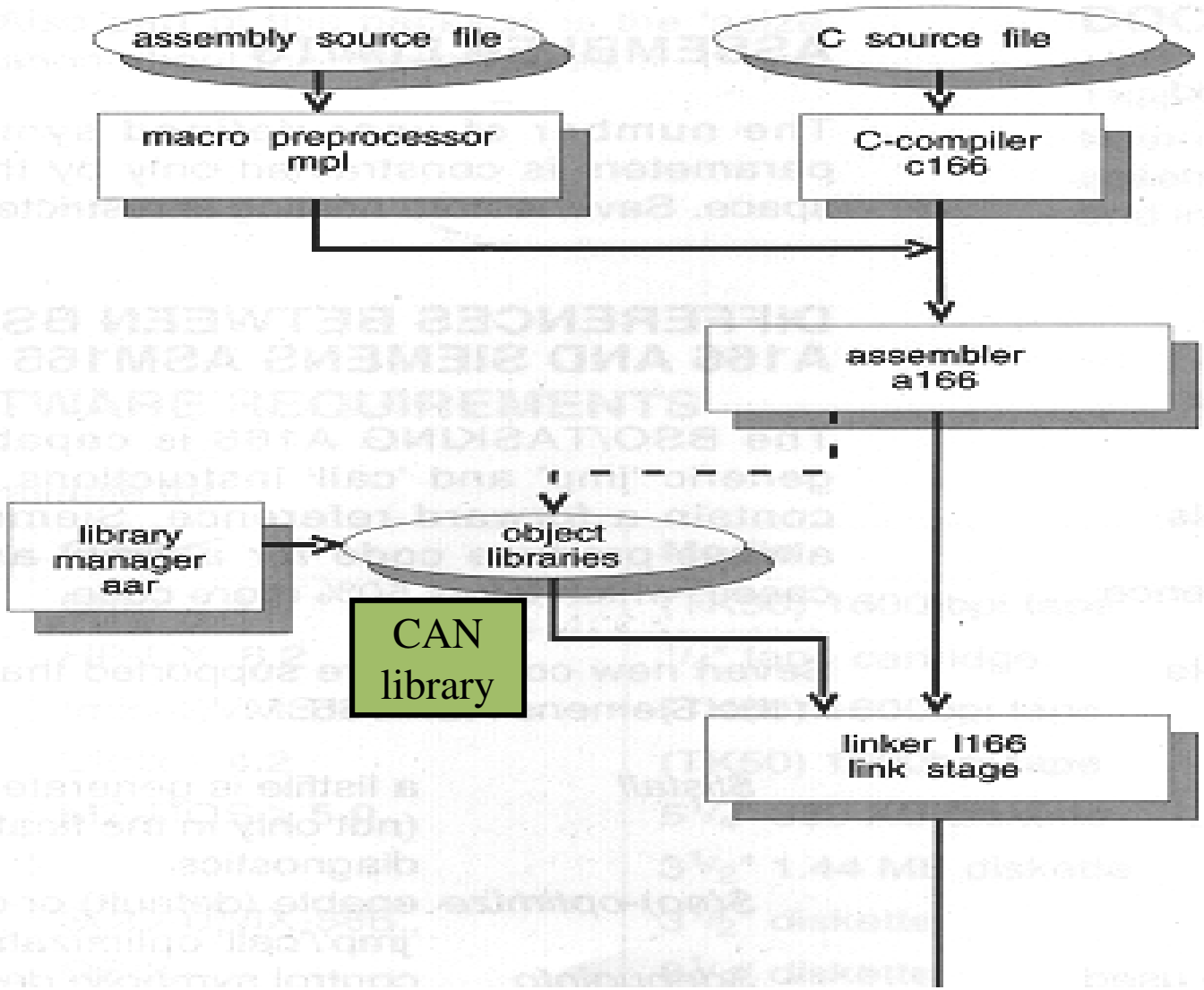
To initialize the CAN Controller, the following actions are required:

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

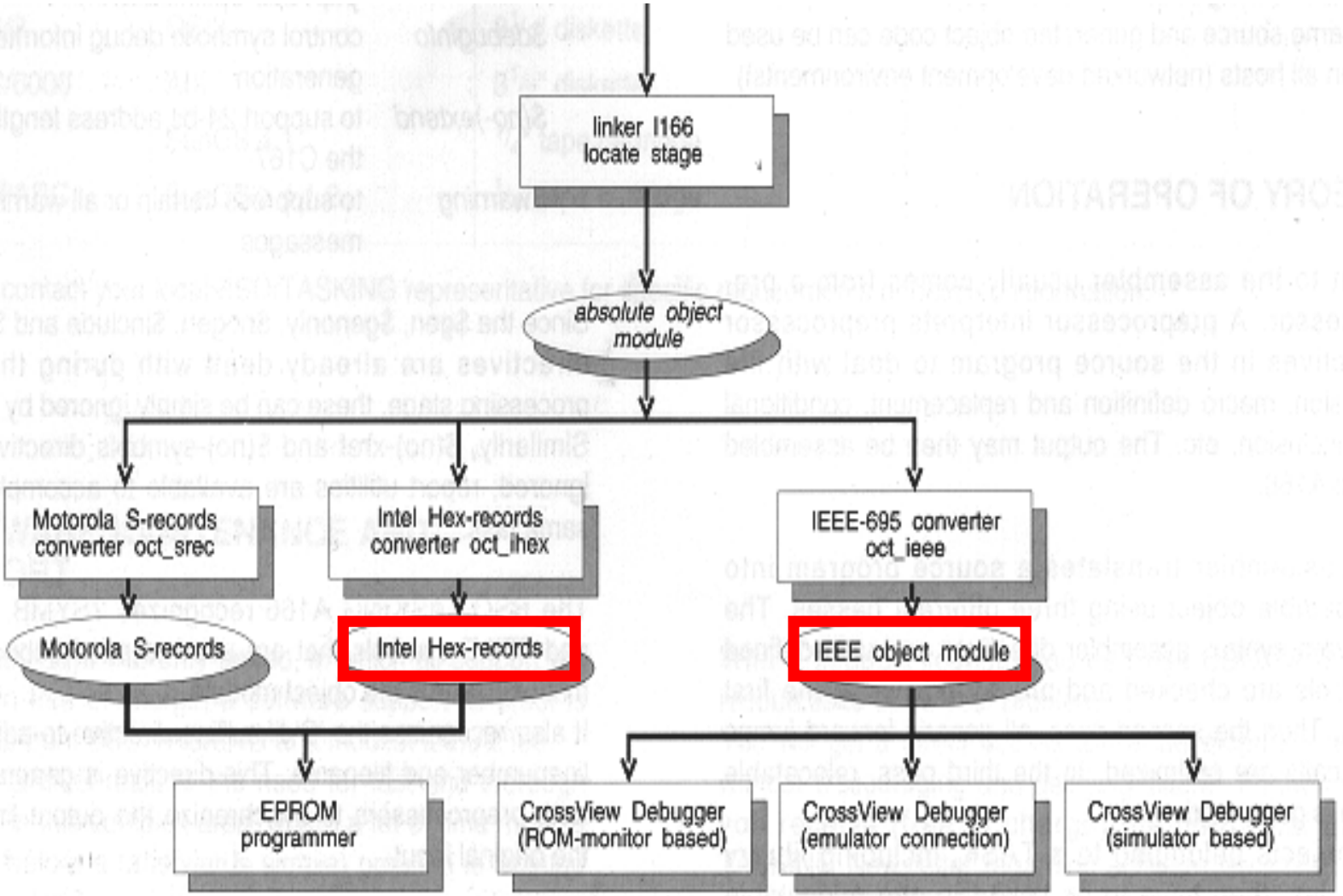
CAN Library:

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

CANKingdom

- CANKingdom is a high-level meta-protocol.
- CANKingdom provides a set of basic elements that can be used to construct complex high-level protocols based on CAN.

CANKingdom Vocabulary

- The system is a kingdom.
- Each node is a city.
- The CAN Network is the postal service.
- Every system has a system designer who is called the king.
- Every city has a mayor who runs the city.

King's and Mayor's Pages

king's pages

Page 0 Start/Stop, Modes

Page 1 Base Number, Response Request

Page 2 Assign Envelopes

Page 3 Assign Groups

Page 4 Remove Groups

Page 5 Action Page - Reaction Page

Page 8 Bit timing registers setting.

Page 9 New City Address

Page 10 Time Elapse.

Page 11 Circular Time Base Setup Page.

Page 12 Repetition Rate and Open Window Setup Page.

Page 16 Place Documents into Folders.

Page 17 Create Documents, Forms or Lines from Lists.

Page 18 Create Compressed Letters.

Page 20 Create Filters for a Postmaster.

~~Block Transfer.~~

Time Herald.

mayor's pages

Page 1 EAN/UPC Code

Page 2 Serial Number

- Red is mandatory
- Black is optional

KVASER

King's Page 0 (KP0)

- Terminates the setup phase.
- Orders a Mayor to set his City into a specific working mode:
  - Action Modes: Keep Current, Run, Freeze, or Reset,
  - Communication Modes: Keep Current, Silent, Listen Only, or Communicate.

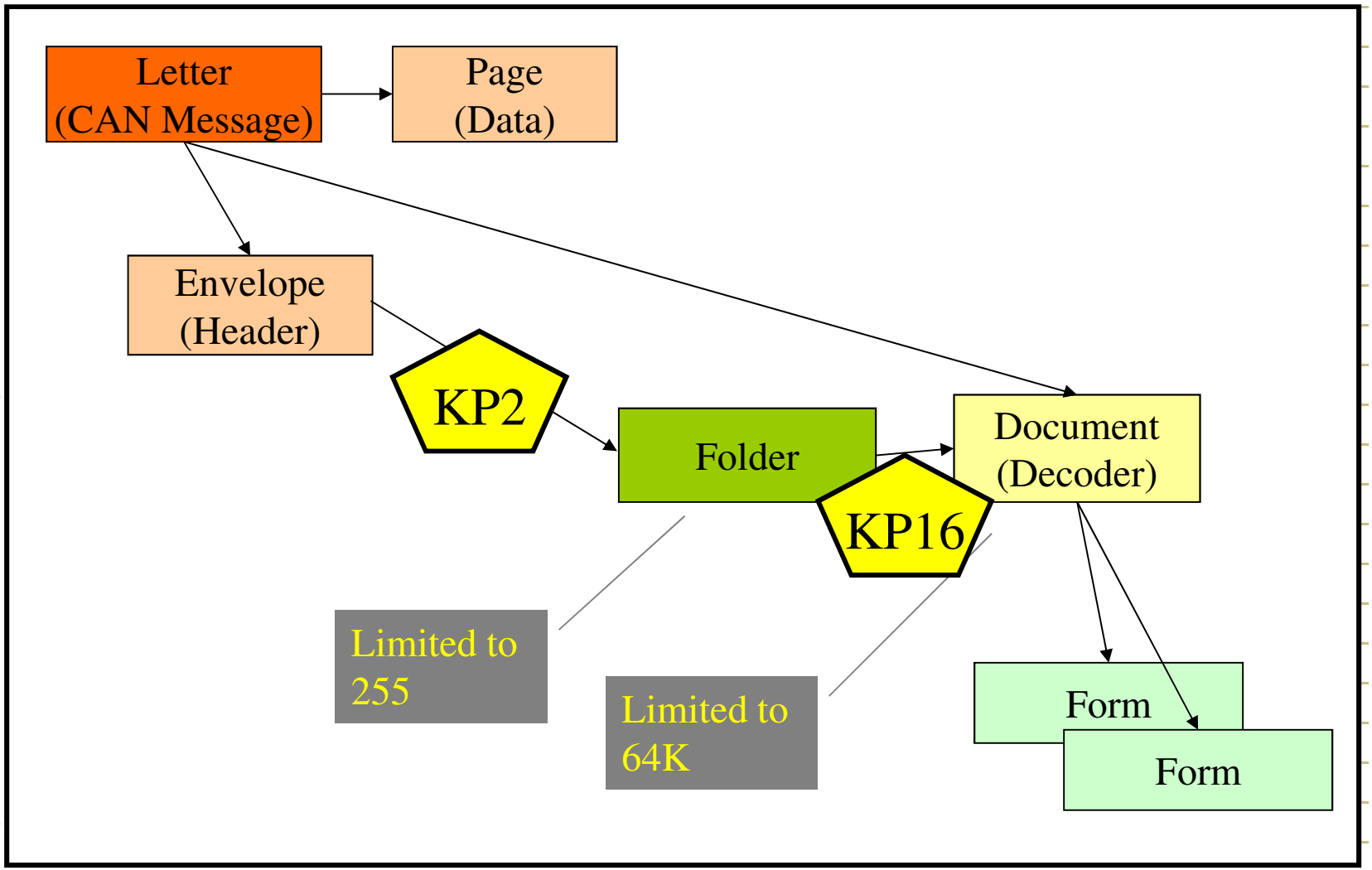
King's Page 1 (KP1)

- Initiating Page.
- Provides the Base Number and asks for a Mayor's response.
- The Mayor of City n assigns an identifier (Base Number + n) to the envelope used to transmit the Mayor's response.
- This identifier is globally unique.

King's Page 2 (KP2) and 16 (KP16)

- **KP16**: Assign a document to a folder and enable or disable a folder. Specify folder properties including Data Length Code (DLC).
- **KP2**: Assign an envelope to a folder or change an existing assignment.

King's Page 2 (KP2) and 16 (KP16)



King's Page 3 (KP3) and 4 (KP4)

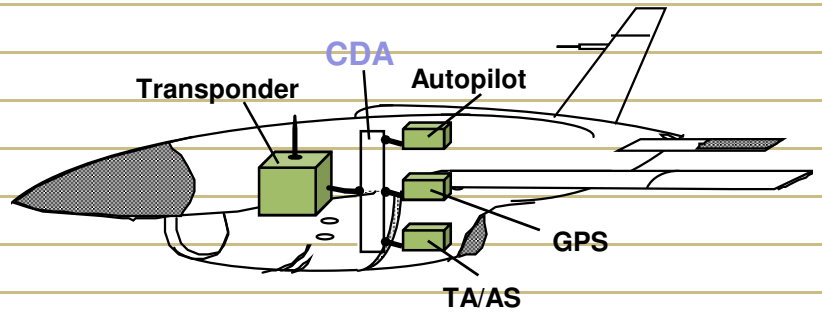
- **KP3**: Assign cities to groups.
- **KP4**: Remove cities from groups.
- KP3 and KP4 can be used to multicast messages to a subset of controllers.

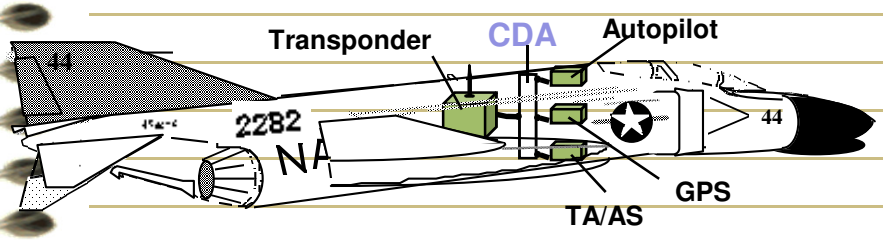
King's Page 5 (KP5)

- Specify an **action/reaction pair**.
- When a particular message is received or when a particular event occurs, a particular message can be sent or a particular event can be generated in response.

Common Digital Architecture

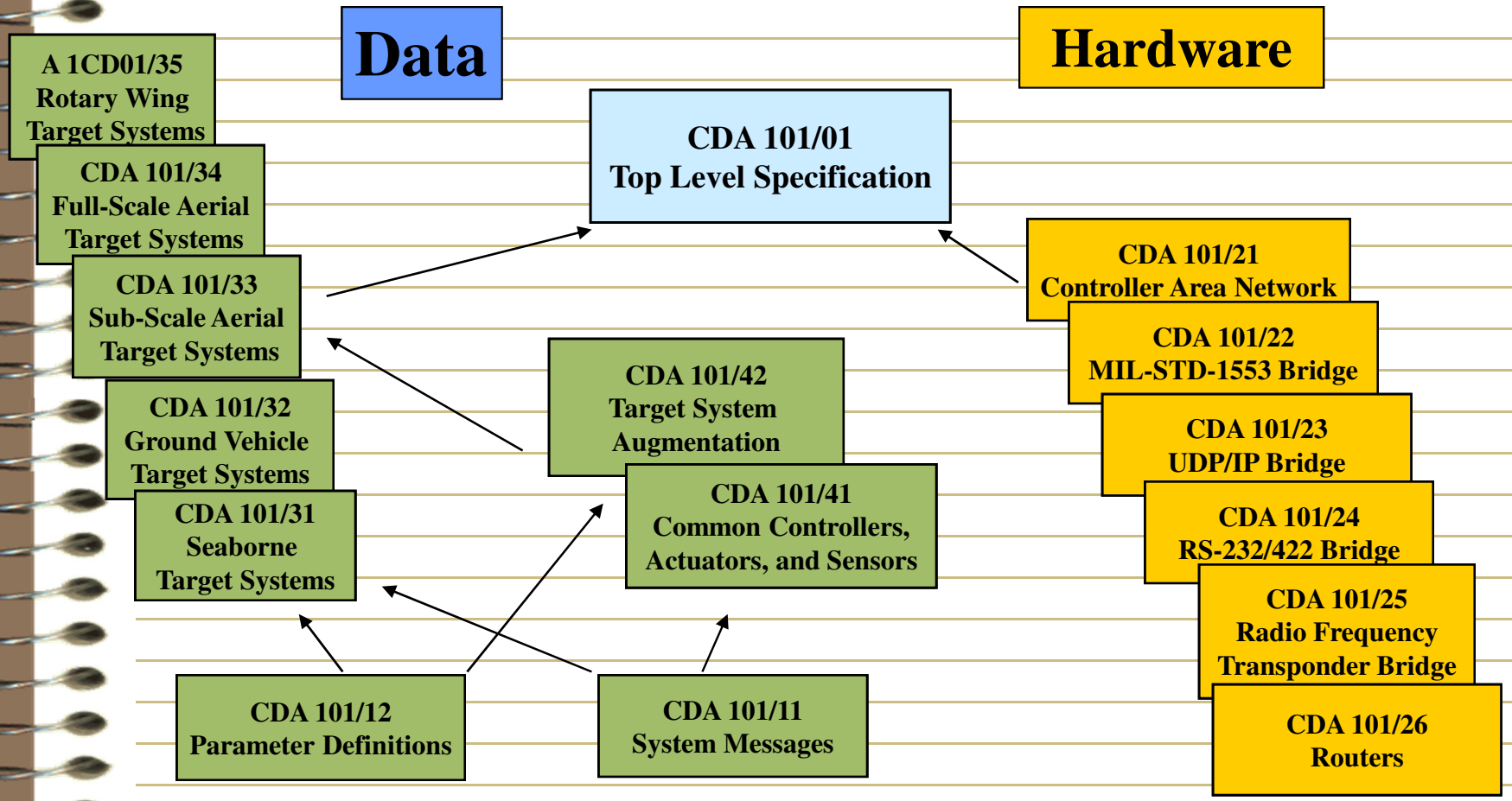
- Develop a standard for interconnecting target vehicle electronics





- Interface standard(s)
  - Software
  - Hardware

Common Digital Architecture (CDA 101)



CDA 101/11 = CANKingdom

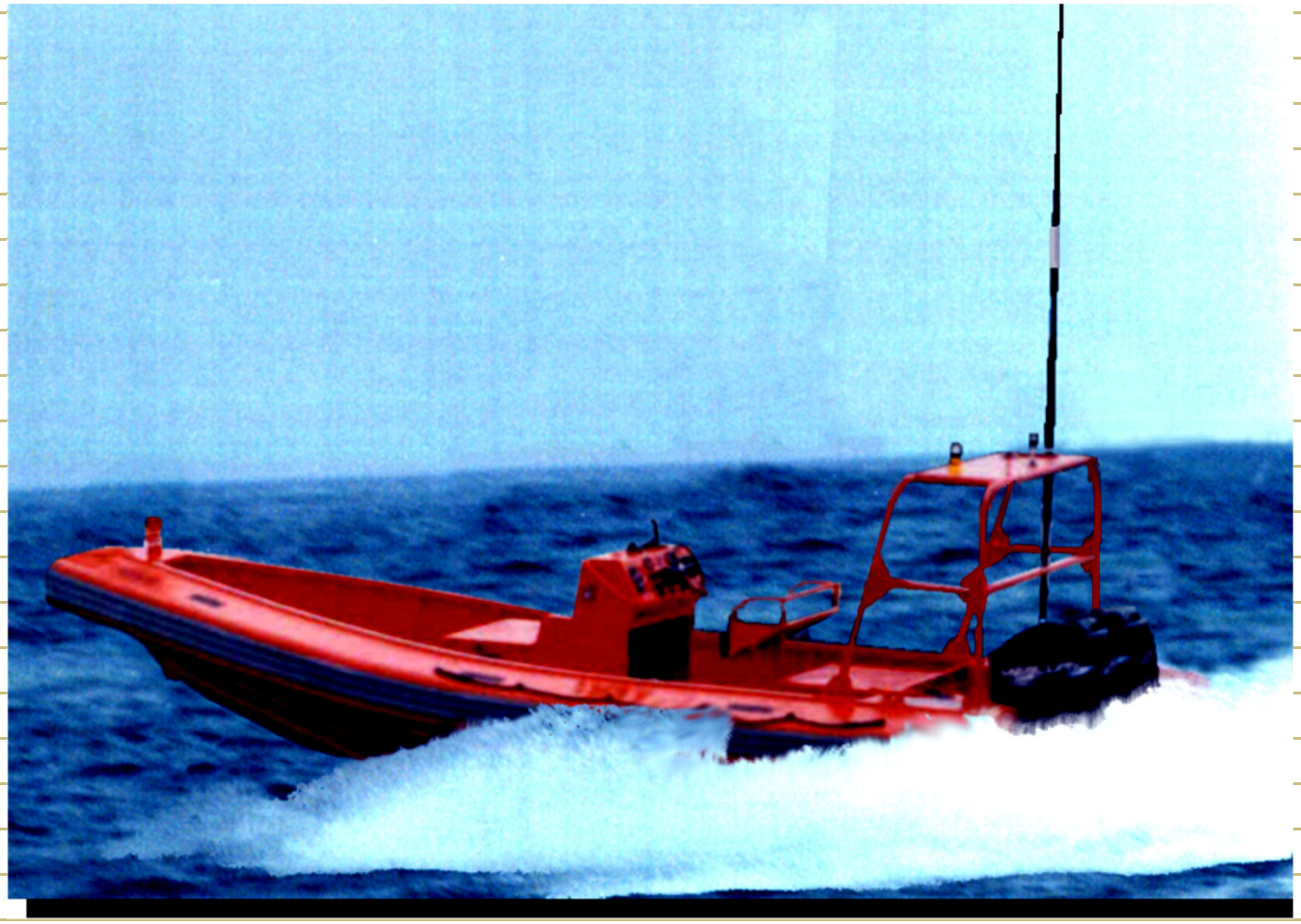
- **KINGS PAGES**
- **Page 0 Start/Stop, Modes**
- **Page 1 Base Number, Response request Page**
- **Page 2 Assign Envelopes**
- Page 3 Assign Groups
- Page 4 Remove Groups
- Page 5 Action Page - Reaction Page.
- Page 8 Bit timing registers setting.
- Page 9 New City physical address.
- Page 10 Time Elapse.
- Page 11 Circular Time Base Setup Page.
- Page 12 Repetition Rate and Open Window Setup Page.
- Page 16 Place Documents into Folders.
- Page 17 Create Documents, Forms or Lines from Lists.
- Page 18 Create Compressed Letters.
- Page 20 Create Filters for a Postmaster.
- Block Transfer.
- **Page 19 First Identifier Masks.**
- **Time Herald**
- **Linear Time**

- **Mayors Pages**
- **Page 1 EAN/UPC Code**
- **Page 2 Serial Number**

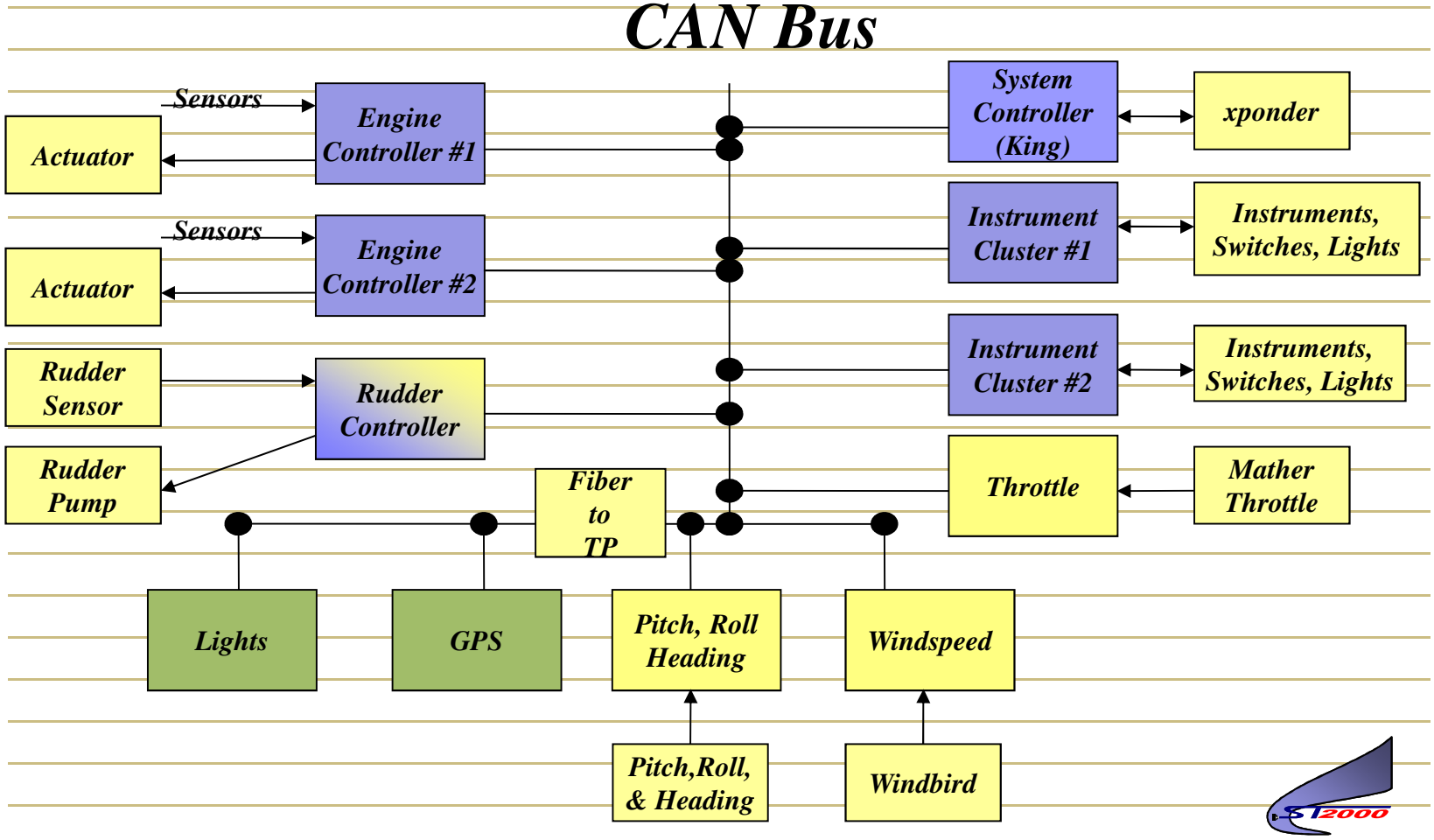
KVASER

ST2000

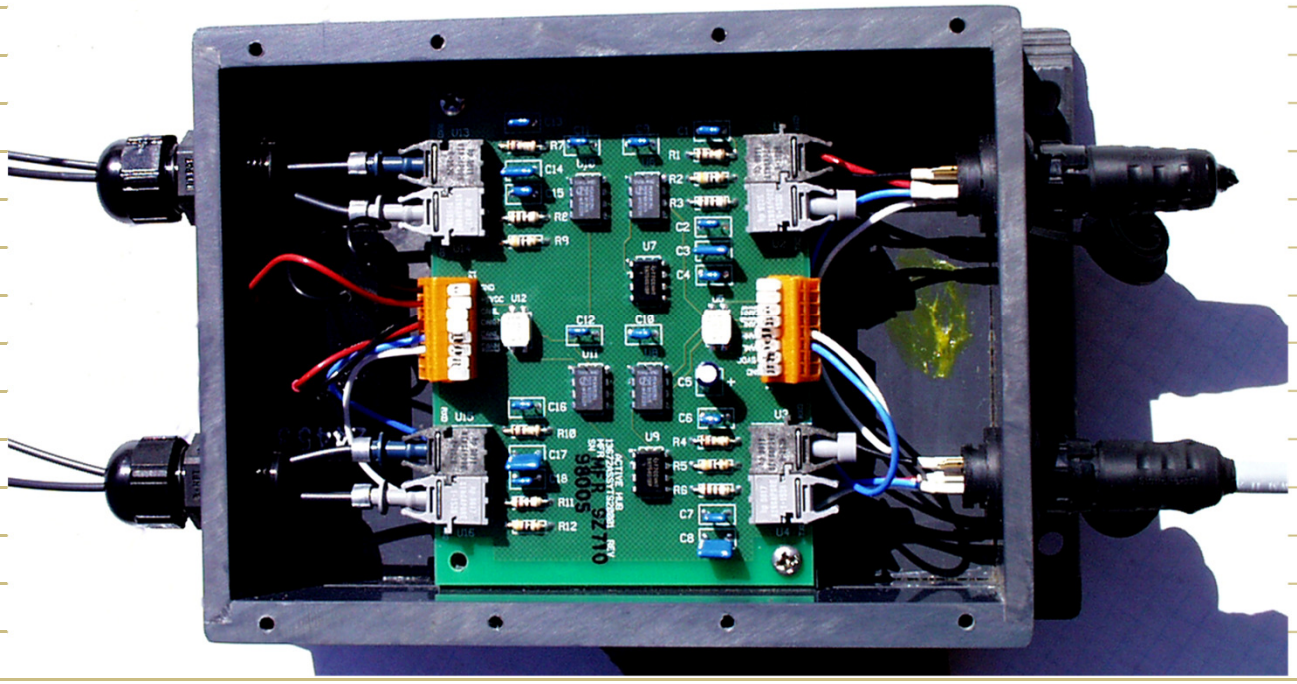
Seaborne Target (ST2000)



CDA 101/31 (ST2000)



Active Fiber Hub



Example: City Lights

- Based on the Common Digital Architecture (CDA101)
- See programs\CityLight21 – hex version or programs\CityLight21r – ROM monitor version
- Example City Application directory includes two files:
  - City.c - application specific code
  - CityDoc.c - interface between application code and CANKingdom

Summary

- Read Ch. 11
- Program 2 – due 4/2/2002