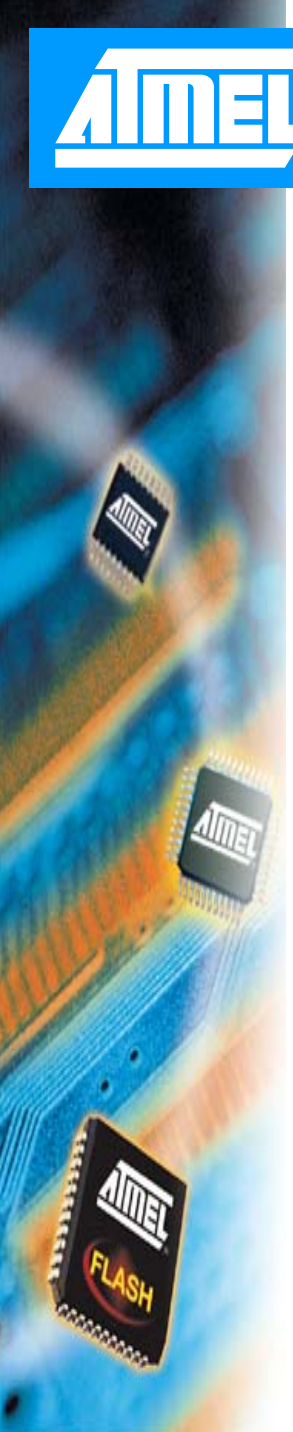
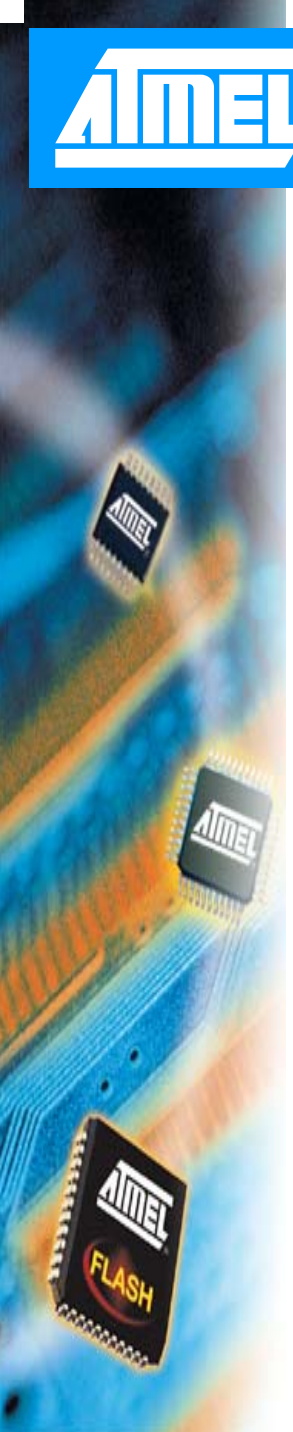
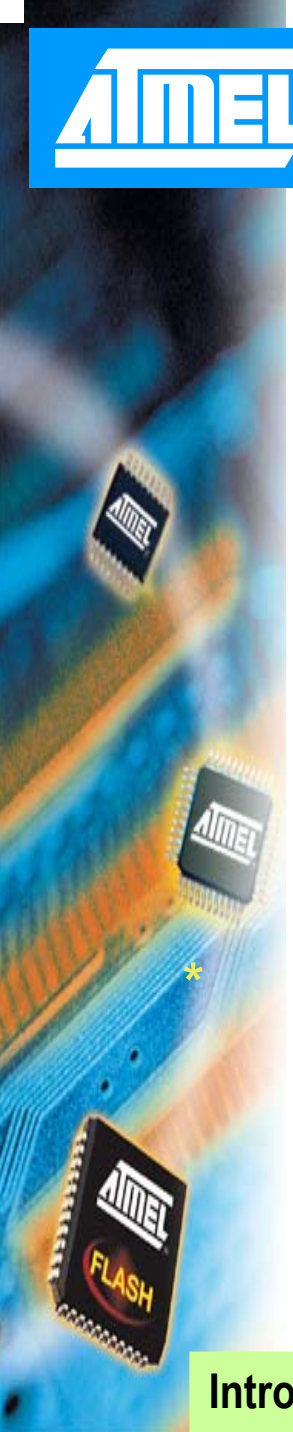


Atmel Microcontrollers
CAN Tutorial

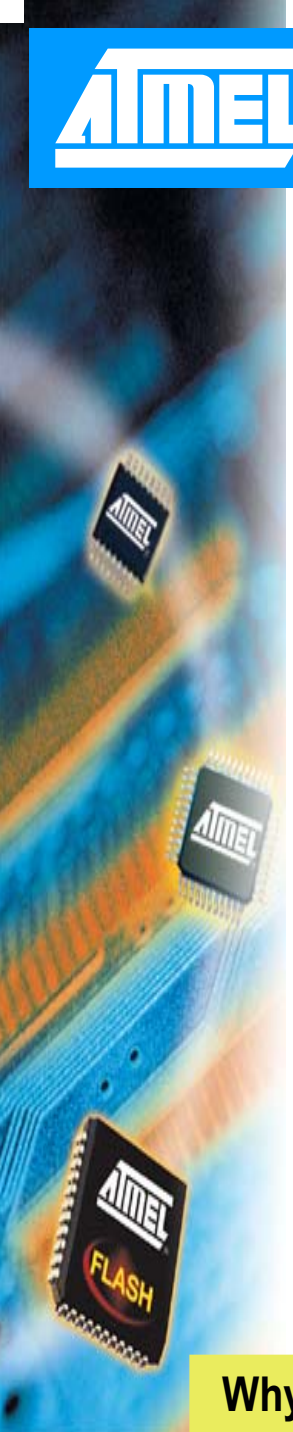
CAN Tutorial
Agenda



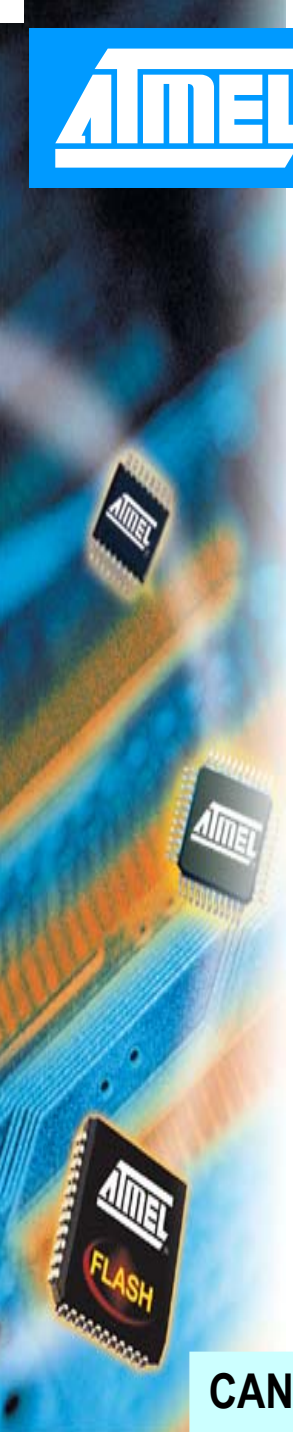
- Introduction or: What is CAN?
- Why CAN?
- CAN Protocol
- CAN higher Layer Protocols
- CAN Applications
- Atmel CAN Microcontrollers Family
- CAN Registers Details
- Conclusion



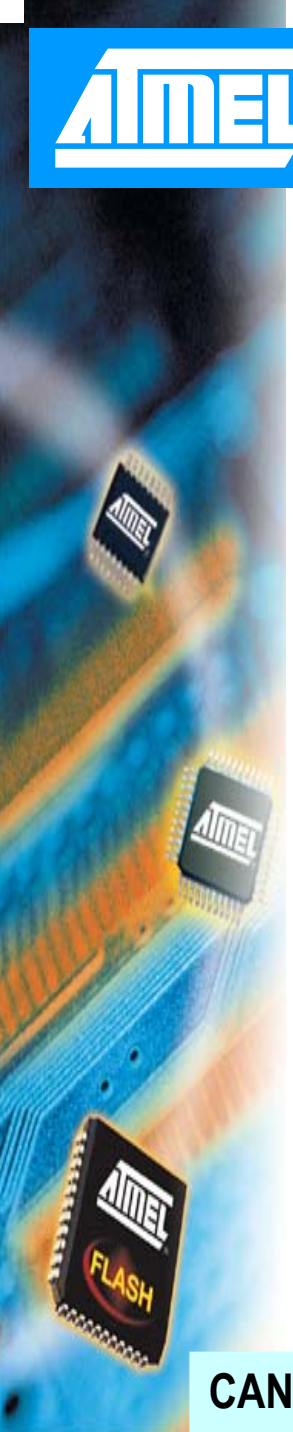
- The CAN is an ISO standard (ISO 11898) for serial communication
- The protocol was developed 1980 by BOSCH for automotive applications
- Today CAN has gained widespread use:
  - Industrial Automation
  - Automotive, …etc.
- The CAN standard includes:
  - Physical layer
  - Data-link layer
    - ✓ Some message types
    - ✓ Arbitration rules for bus access
    - ✓ Methods for fault detection and fault confinement



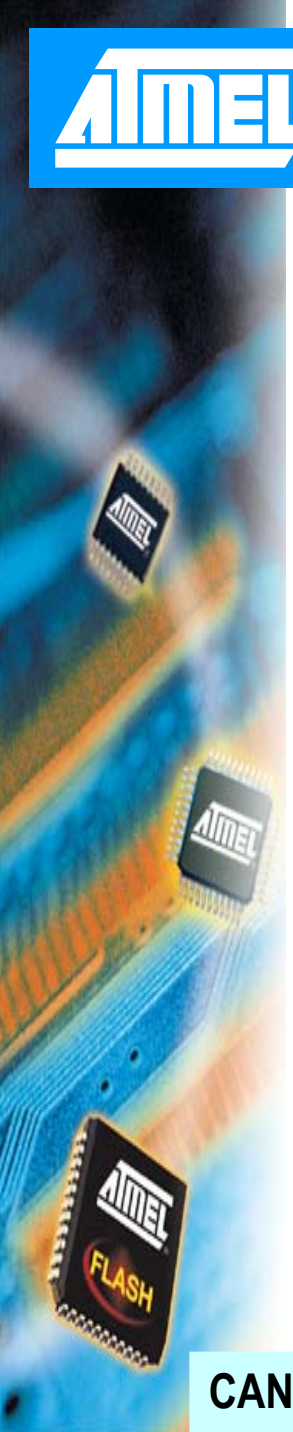
- **Mature Standard**
  - CAN protocol more than 14 years
  - Numerous CAN products and tools on the market
- **Hardware implementation of the protocol**
  - Combination of error handling and fault confinement with high transmission speed
- **Simple Transmission Medium**
  - Twisted pair of wires is the standard, but also just one wire will work
  - Other links works, too: Opto - or radio links
- **Excellent Error Handling**
  - CRC error detection mechanism
- **Fault Confinement**
  - Built-in feature to prevent faulty node to block system
- **Most used protocol in industrial and automotive world**
- **Best Performance / Price ratio**



- What is CAN?
- ISO-OSI Reference Model
- CAN Bus Logic
- Typical CAN Node
- CAN Bus Access and Arbitration
- CAN Bit Coding & Bit Stuffing
- CAN Bus Synchronization
- CAN Bit Construction
- Relation between Baud Rate and Bus Length
- Frame Formats (1)
- Frame Formats (2)
- Frame Formats (3)
- Frame Formats (4)
- Fault Confinement (1)
- Fault Confinement (2)
- Undetected Errors



- Controller Area Network
  - Invented by Robert Bosch GmbH
  - Asynchronous Serial Bus
  - Absence of node addressing
    - ✓ Message identifier specifies contents and priority
    - ✓ Lowest message identifier has highest priority
  - Non-destructive arbitration system by CSMA with collision detection
  - Multi-master / Broadcasting concept
  - Sophisticated error detection & handling system
  - Industrial and Automotive Applications



7. Application Layer

6. Presentation Layer

5. Session Layer

4. Transport Layer

3. Network Layer

2. Data Link Layer

1. Physical Layer

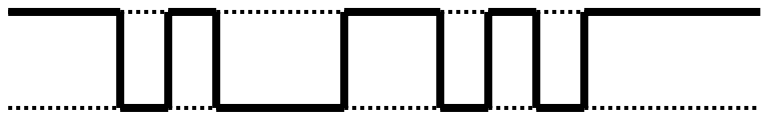
HLPs: CANopen, DeviceNet, OSEK/V**

Partially implemented by Higher Layer Protocols (HLP)

CAN Protocol

***) OSI -** Open System Interconnection

Two logic states on the CAN bus



"1" = recessive

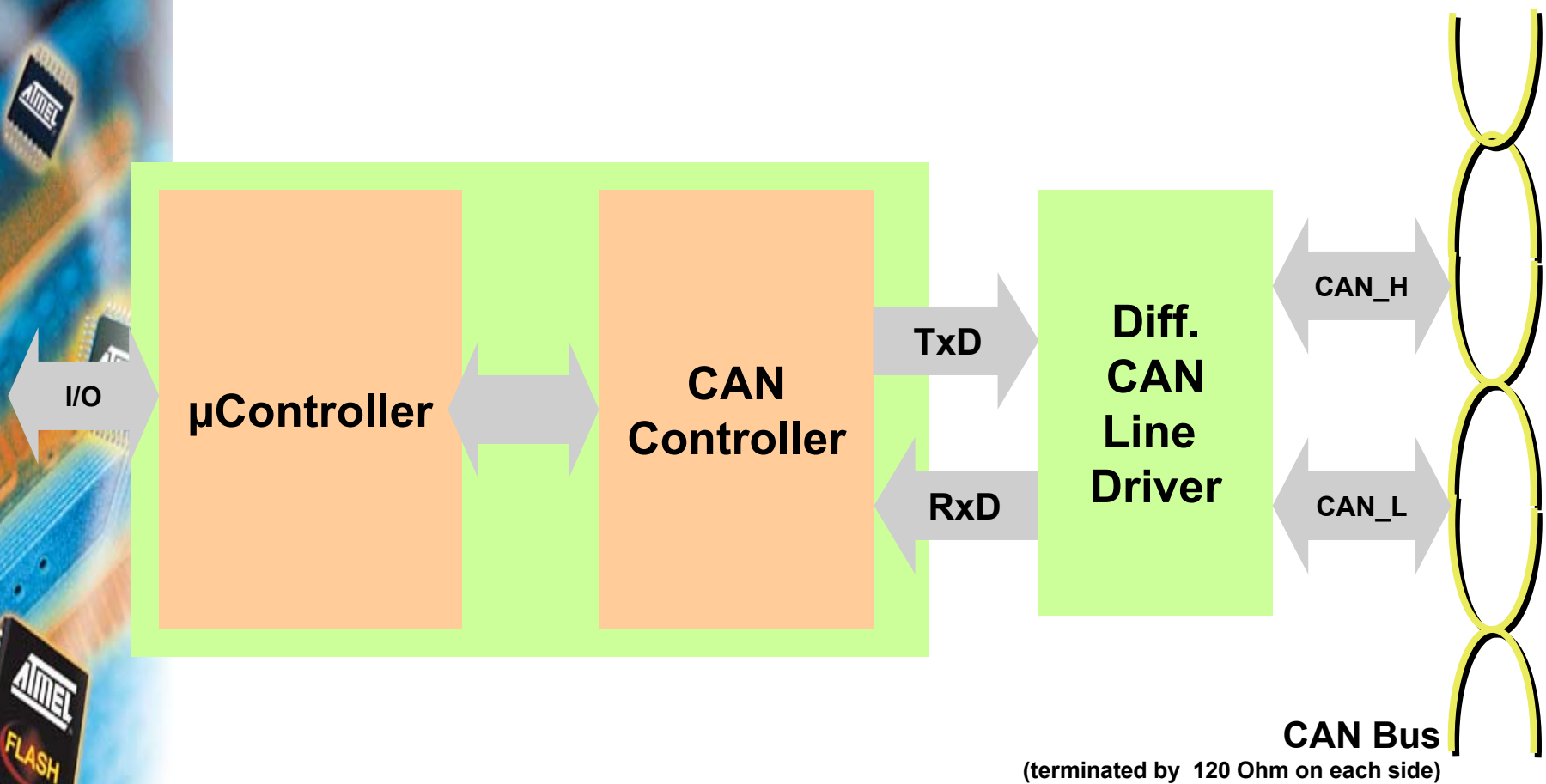
"0" = dominant

Node A	Node B	Node C	BUS
D	D	D	D
D	D	R	D
D	R	D	D
D	R	R	D
R	D	D	D
R	D	R	D
R	R	D	D
R	R	R	R

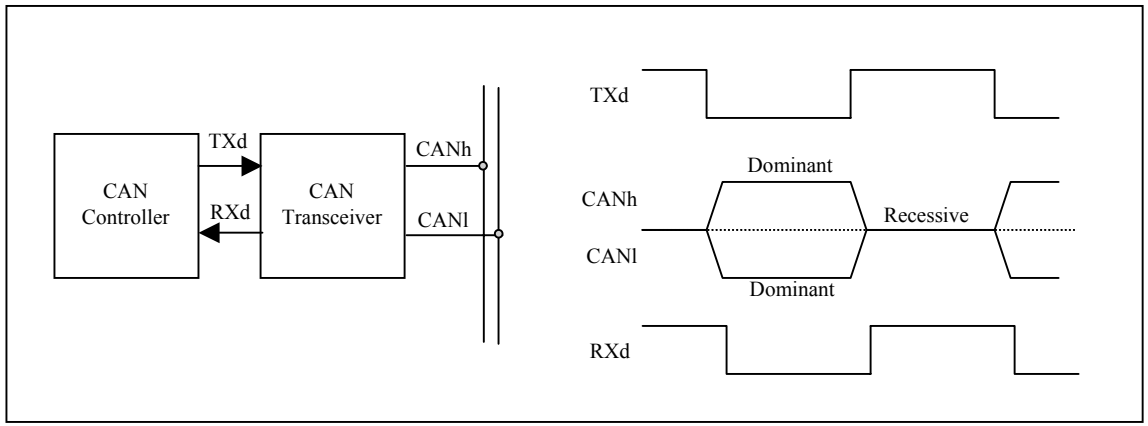
"Wired-AND" function:
as soon as one node transmit a dominant bit (zero)
the bus is in the dominant state

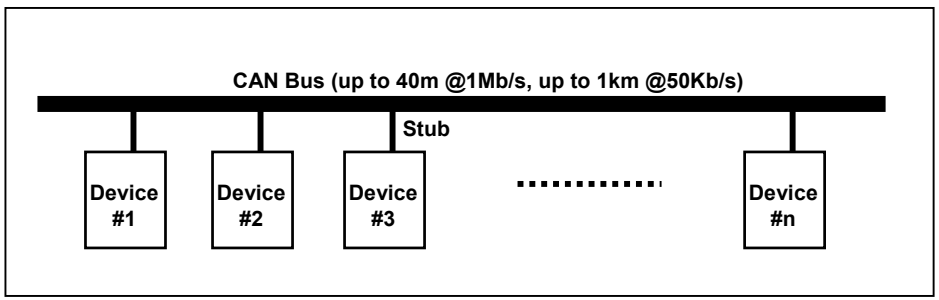
Only if all nodes transmit recessive bits (ones)
the Bus is in the recessive state

CAN Tutorial
Typical CAN Node



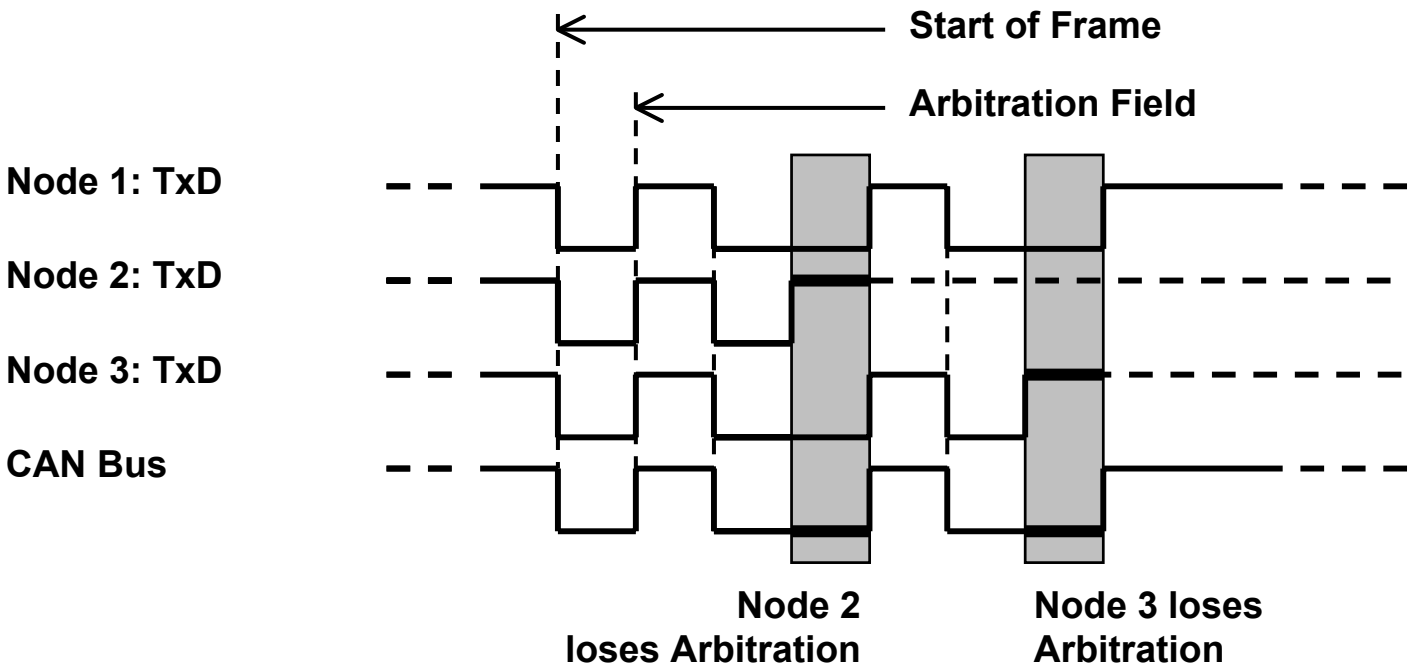
CAN Protocol





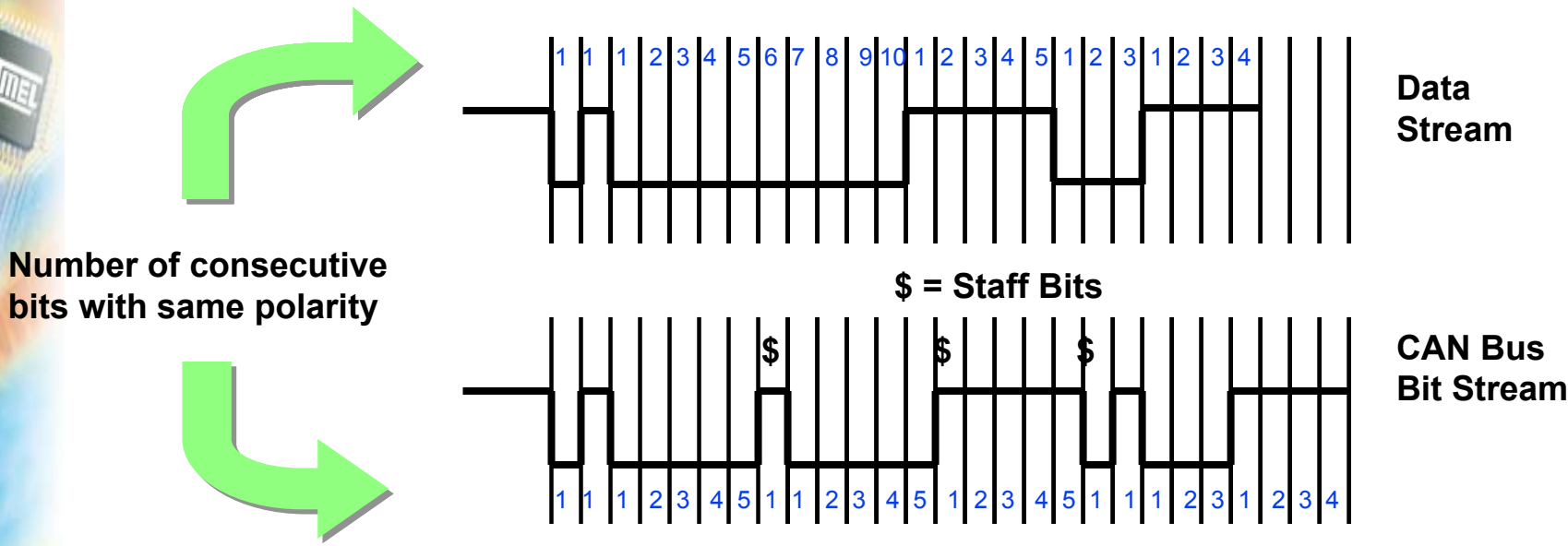
CAN Protocol

CAN Bus Access and Arbitration: CSMA/CD and AMP *)

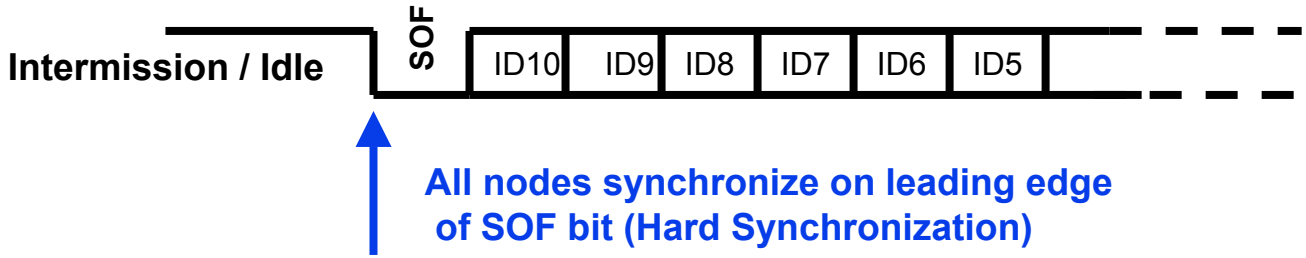


Carrier Sense Multiple Access/Collision Detection and Arbitration by Message Priority

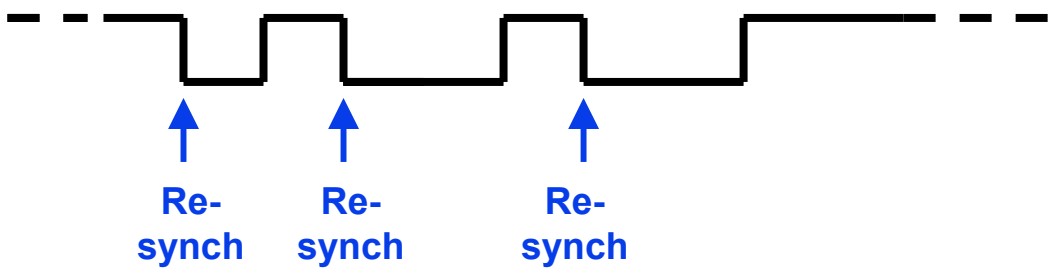
- Bit Coding : NRZ (Non-Return-To-Zero code) does not ensure enough edges for synchronization
- Stuff Bits are inserted after 5 consecutive bits of the same level
- Stuff Bits have the inverse level of the previous bit.
- No deterministic encoding, frame length depends on transmitted data



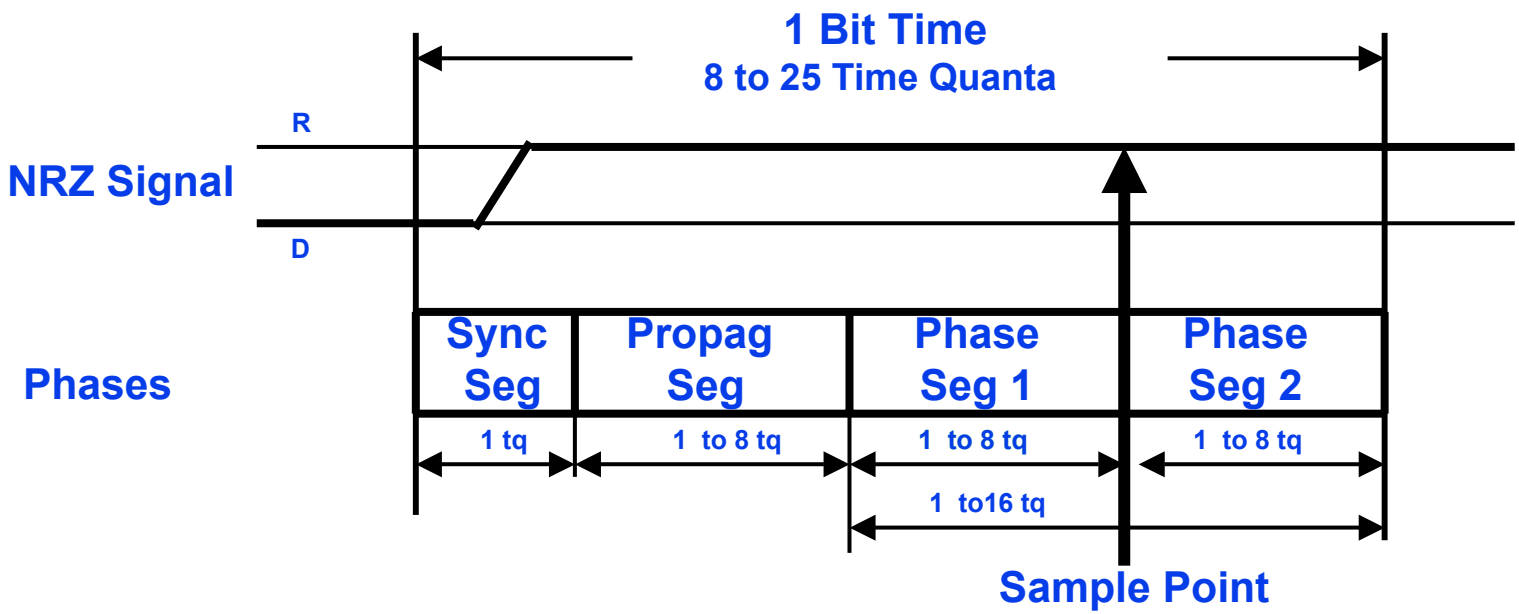
- **Hard synchronization at Start Of Frame bit**



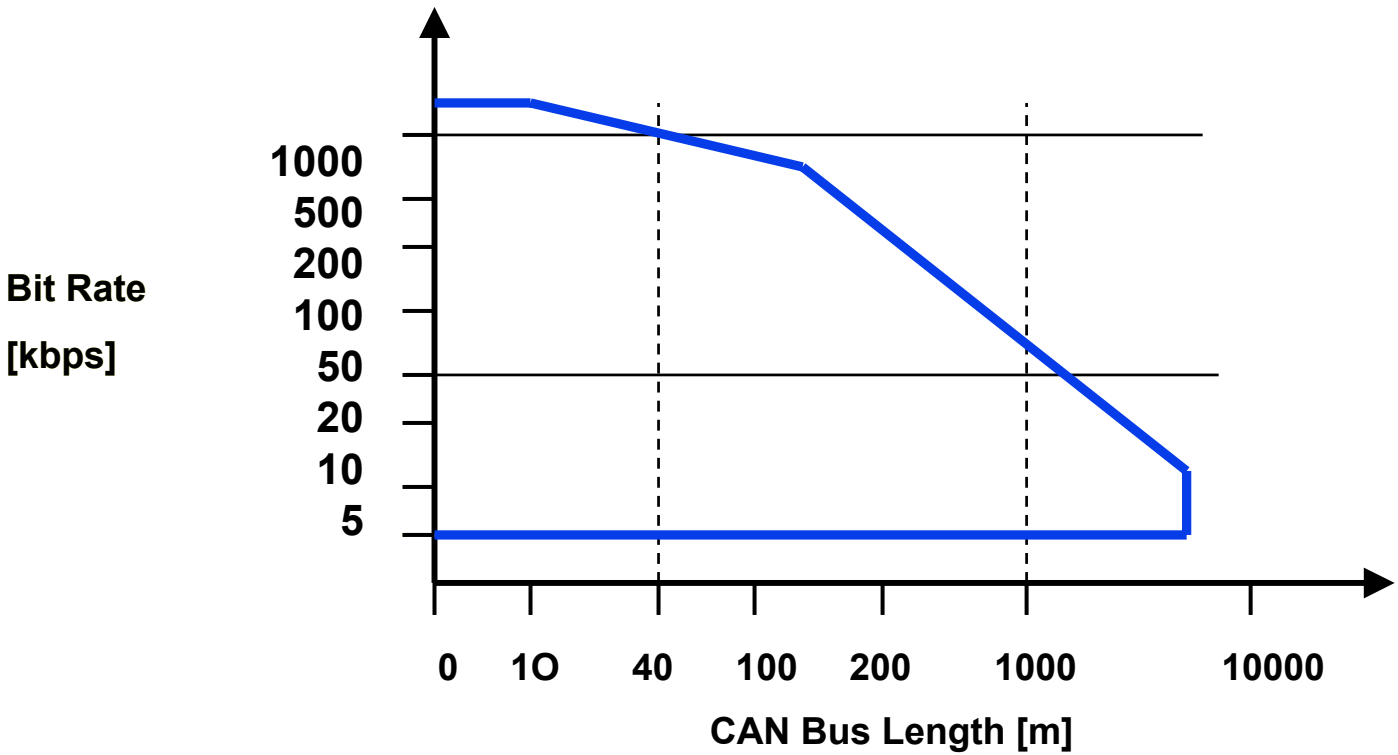
- **Re-Synchronization on each Recessive to Dominant bit**



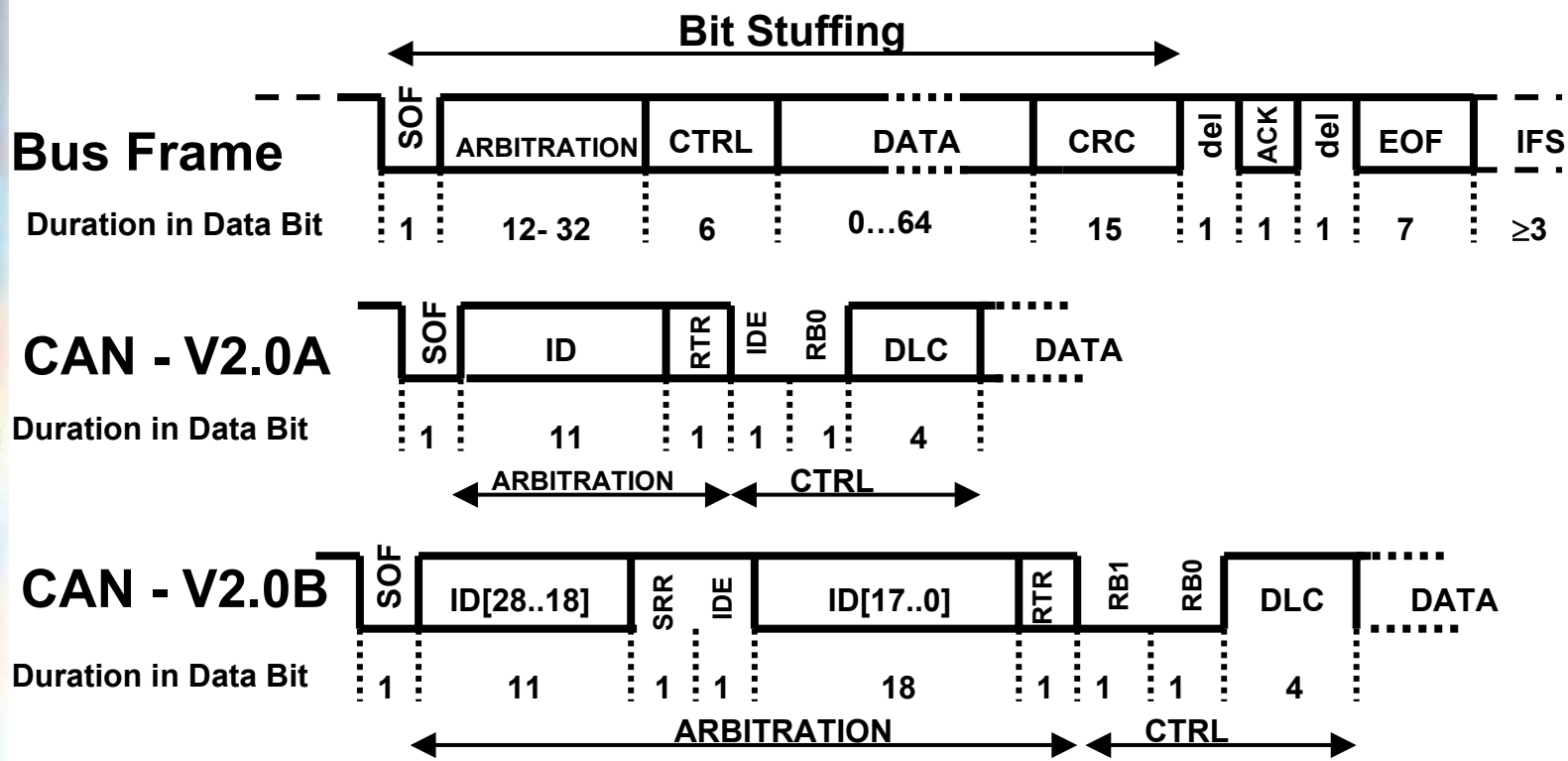
- Length of one time quanta can be set to multiple of µController clock
- 1 Time quantum = 1 period of CAN Controller base clock
- Number of time quanta in Propag and Phase segments is programmable



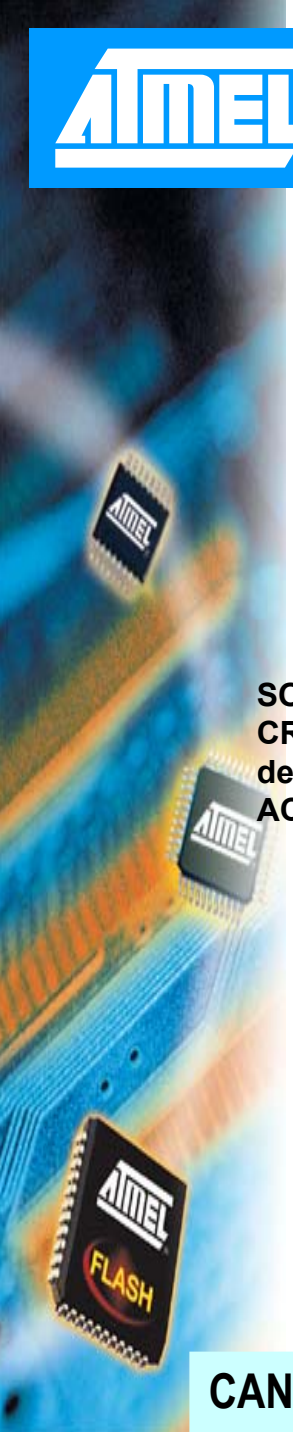
Relation between Baud Rate and Bus Length



Example based on CAN Bus Lines by twisted pair

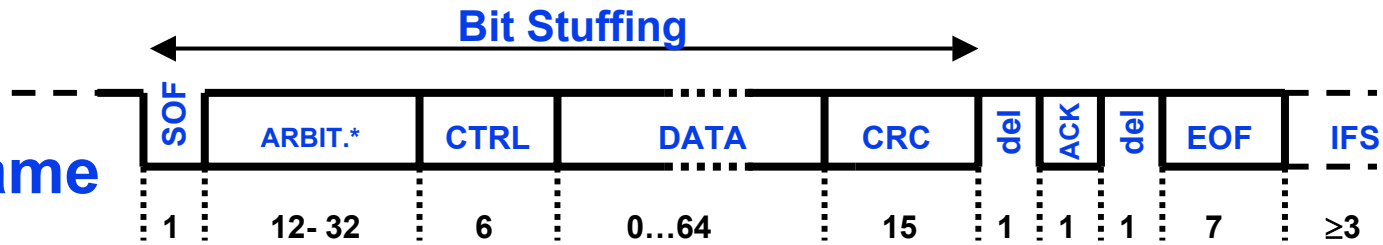


SOF	Start of Frame	EOF	End of Frame	RTR	Remote Transmission Request
CRC	Cyclic Redundancy Code	IFS	Inter Frame Spacing	SRR	Substitute Remote Request
del	Delimiter	ID	Identifier	RB0/1	Reserved bits
ACK	Acknowledge	IDE	Identifier Extension	DLC	Data Length Code



SOF	Start of Frame	EOF	End of Frame	RTR	Remote Transmission Request
CRC	Cyclic Redundancy Code	IFS	Inter Frame Spacing	SRR	Substitute Remote Request
del	Delimiter	ID	Identifier	RB0/1	Reserved bits
ACK	Acknowledge	IDE	Identifier Extension	DLC	Data Length Code

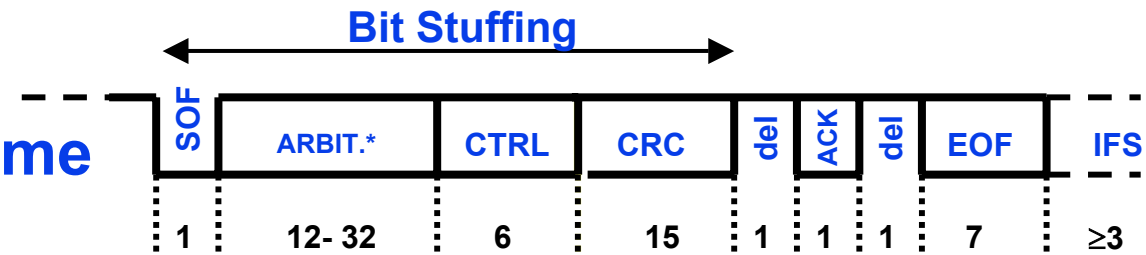
Data Frame



Duration in Data Bit

(*) RTR = dominant

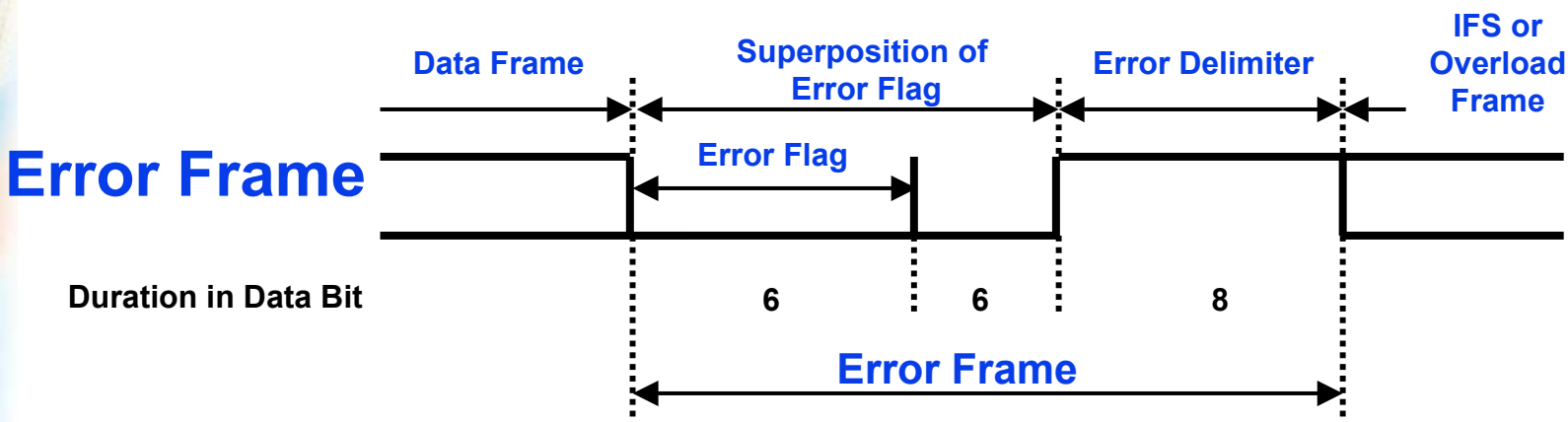
Remote Frame



Duration in Data Bit

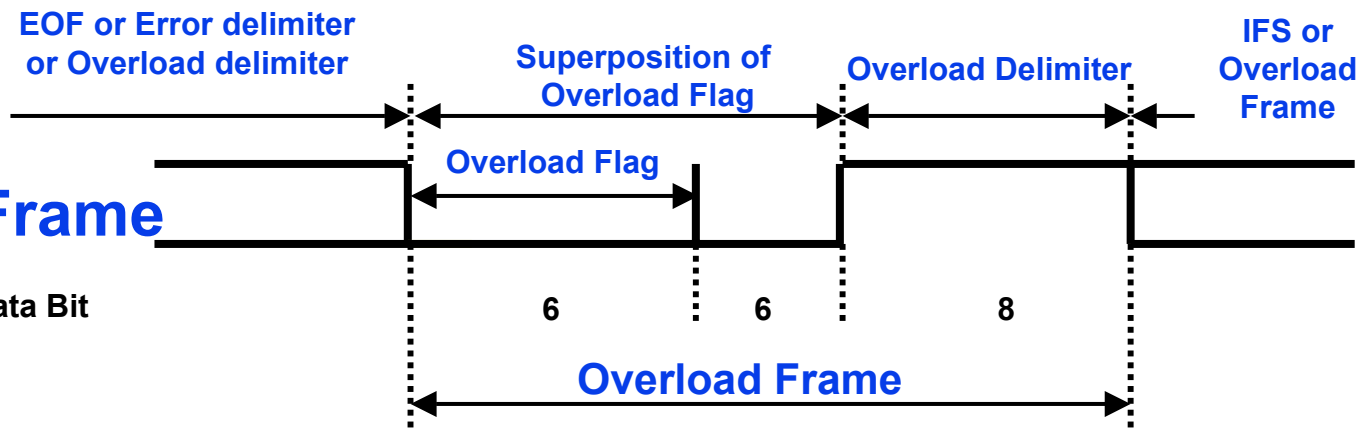
(*) RTR = recessive

- If any of the CAN nodes detects a violation of the frame format
- or a stuff error, it immediately sends an Error Frame

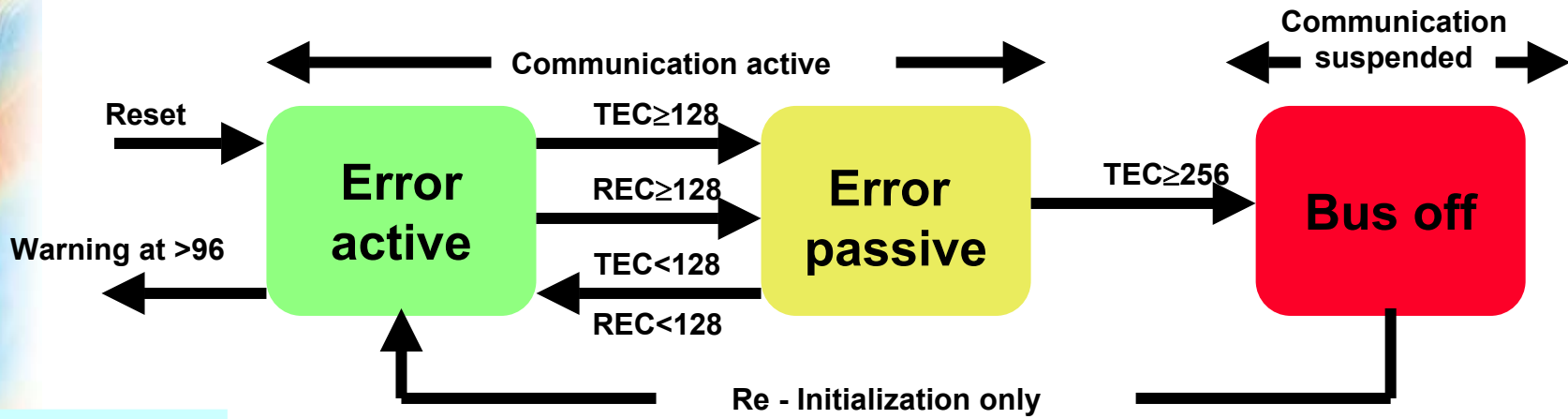


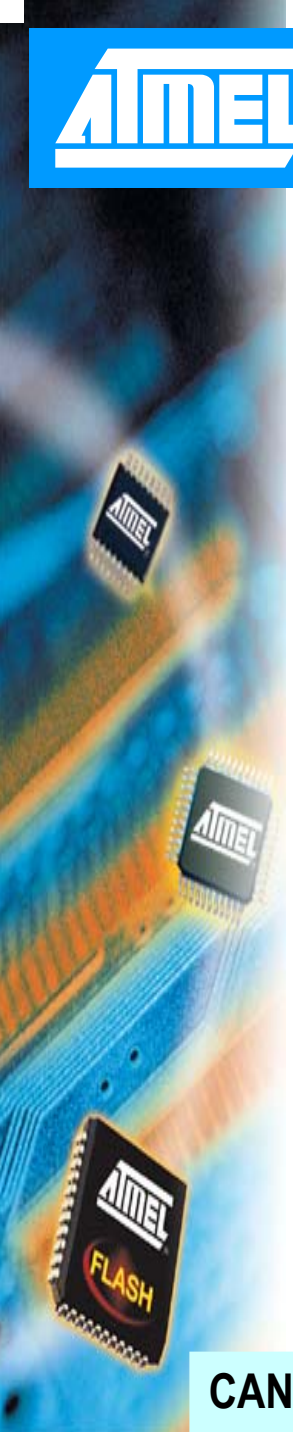
- If any of the CAN nodes suffers from a “data over flow”, it might send
- up two consecutive Overload Frames to delay the network

Overload Frame

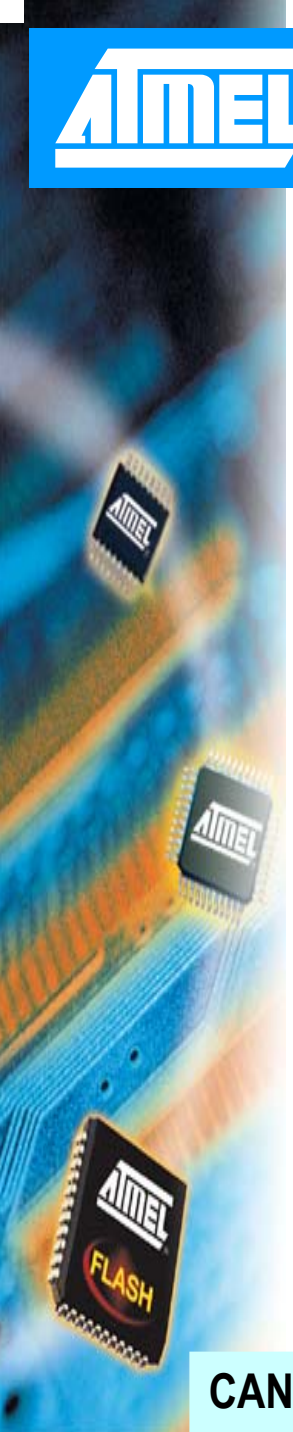


- **Three fundamental states define each node's error signaling**
  - Error active: Normal state, node can send all frames incl.error frames
  - Error passive: Node can send all frames excluding error frames
  - Bus off: Node is isolated from bus
- **Internal error counts determine the state**
  - Transmit error counter (TEC) An error increases the counter by 8
  - Receive error counter (REC) A successful operation decreases by 1
- **Aims to prevent from bus dead-locks by faulty nodes**





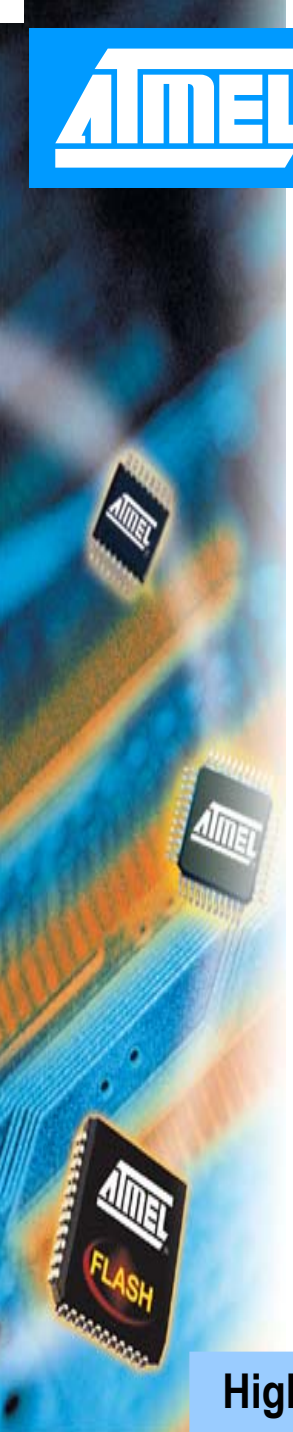
- Cyclic Redundancy Check (CRC)
- The CRC is calculated over the non-stuffed bit stream starting with the SOF and ending with the Data field by the transmitting node
- The CRC is calculated again of the destuffed bit stream by the receiving node
- A comparison of the received CRC and the calculated CRC is made by the receiver
- In case of mismatch the erroneous data frame is discarded . Instead of sending an acknowledge signal an error frame is sent.



- Error statistics depend on the entire environment
- Total number of nodes
- Physical layout
- EMI disturbance
- Automotive example
- 2000 h/y
- 500 kbps
- 25% bus load

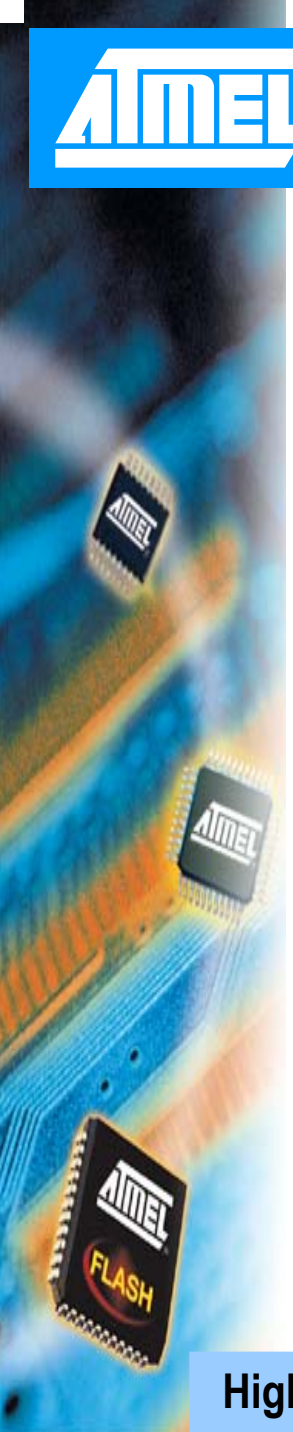
→ **One undetected error every 1000 years**

CAN Higher Layer Protocols (HLPs)

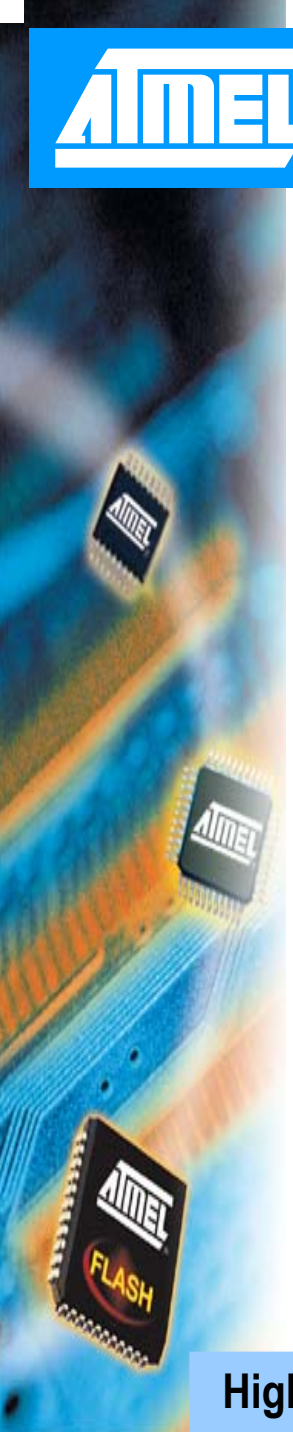


- Why HLPs
- CANOpen
- DeviceNet
- CAN Kingdom
- OSEK/VDX
- SDS
- J1939

Higher level protocols

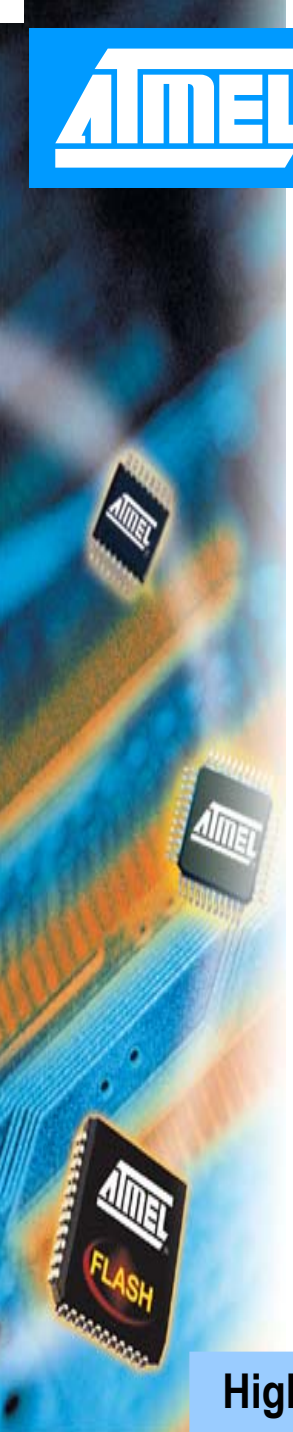


- The CAN protocol defines only the ‘physical’ and a low ‘data link layer’!
- The HLP defines:
  - Start-up behavior
  - Definition of message identifiers for the different nodes
  - Flow control
  - transportation of messages > 8bytes
  - Definition of contents of Data Frames
  - Status reporting in the system



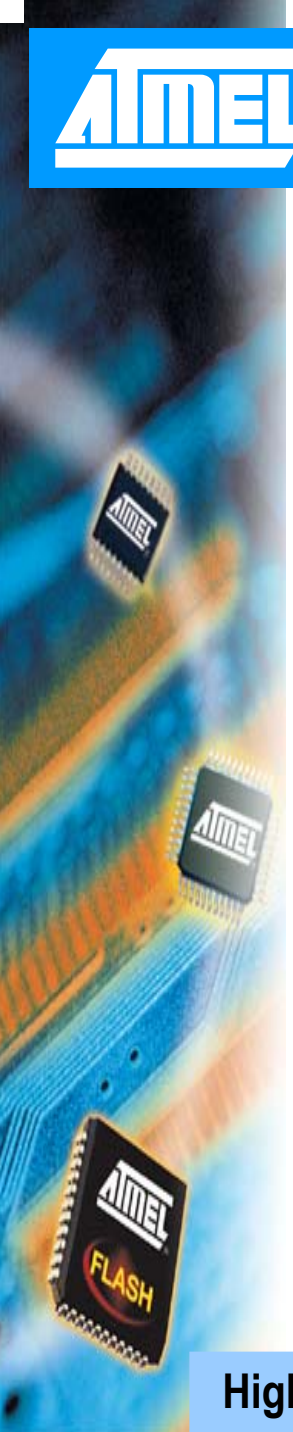
- **Features**
  - CANopen a subset from CAL (CAN Application Layer) developed by CiA!
  - Auto configuration the network
  - Easy access to all device parameters
  - Device synchronization
  - Cyclic and event-driven data transfer
  - Synchronous reading or setting of inputs, outputs or parameters
- **Applications**
  - Machine automatisation
- **Advantages**
  - Accommodating the integration of very small sensors and actuators
  - Open and vendor independent
  - Support s inter-operability of different devices
  - High speed real-time capability

Higher level protocols



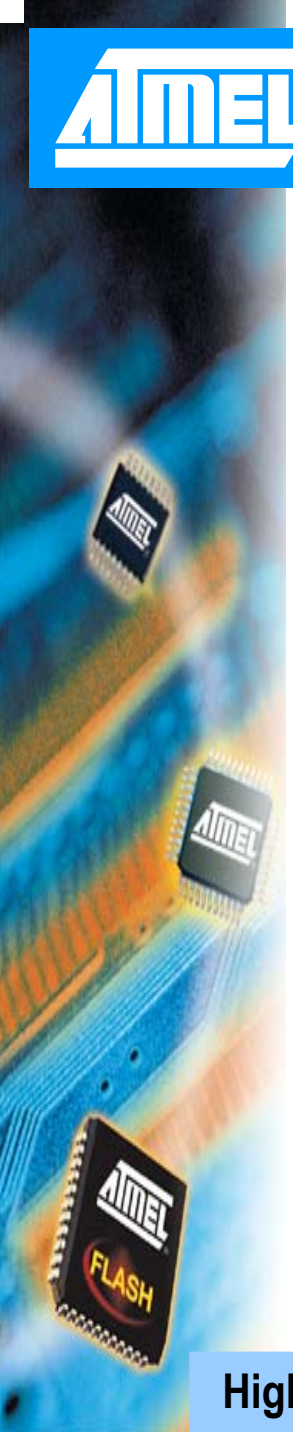
- **Features**
  - Created by Allen-Bradley (Rockwell Automatisation nowadays), now presented by the users group ODVA (Open DeviceNet Vendor Association)
  - Power and signal on the same network cable
  - Bus addressing by: Peer-to-Peer with multi-cast & Multi-Master & Master-Slave
  - Supports only standard CAN
- **Applications**
  - Communications link for industrial automatisation: devices like limit switches, photo-electric sensors, valve manifolds, motor starters, process sensors, bar code readers,variable frequency drives, panels...
- **Advantages**
  - Low cost communication link and vendor independent
  - Removal and replacement of devices from the network under power

Higher level protocols



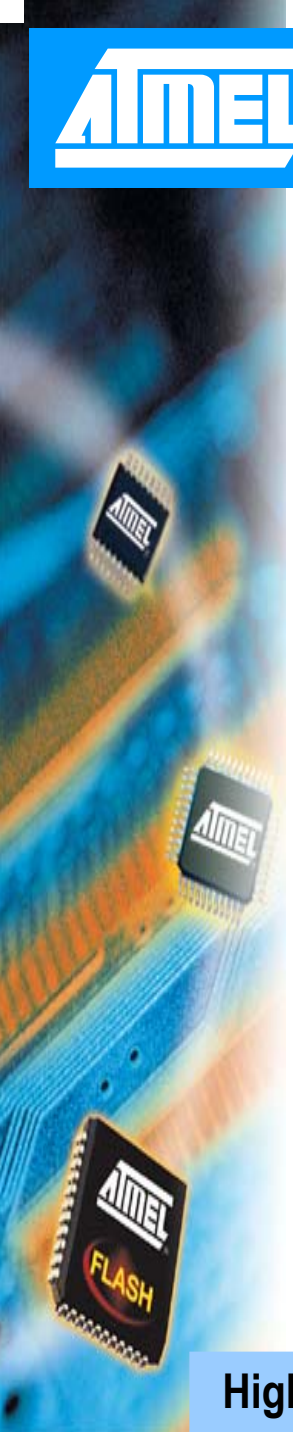
- CAN Kingdom is more then a HLP: A Meta protocol
  - Introduced by KVASER, Sweden
  - A ‘King’ (system designer) takes the full responsibility of the system
  - The King is represented by the Capital (supervising node)
  - World wide product identification standard EAN/UPC is used for
- Applications
  - Machine control, e.g. industrial robots, weaving machines, mobile hydraulics, power switchgears, wide range of military applications
- Advantages
  - Designed for safety critical applications
  - Real time performance
  - Scalability
  - Integration of DeviceNet & SDC modules in CAN Kingdom possible

Higher level protocols



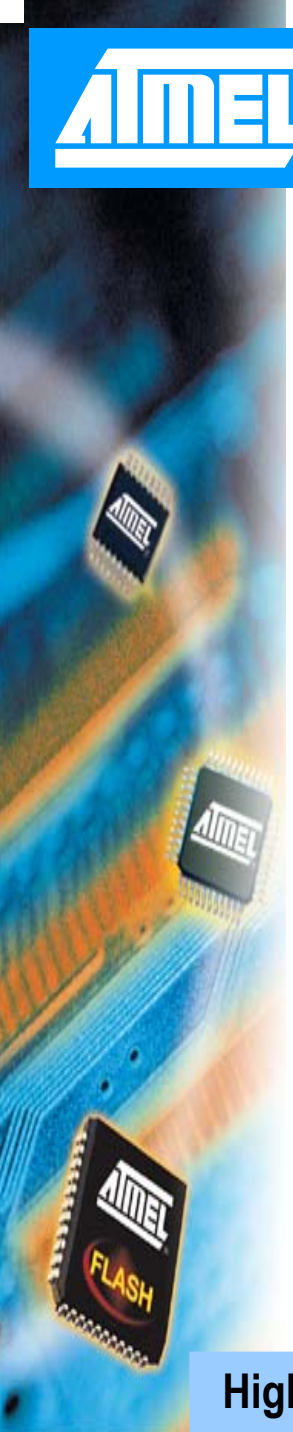
- **Initialized by:**
  - BMW, Bosch, DaimlerChrysler, Opel, Siemens,VW & IIIT of the University of Karlsruhe / PSA and Renault
- **OSEK/VDX includes:**
  - Communication (Data exchange within and between Control Units)
  - Network Management (Configuration determination and monitoring)
  - Operating System (Real-time executive for ECU software)
- **Motivation:**
  - High, recurring expenses in the development and variant management of non-application related aspects of control unit software
  - Compatibility of control units made by different manufactures due to different interfaces
- **Goal:** Portability and re-usability of the application software
- **Advantages:** Clear saving in costs and development time

Higher level protocols



- **Features**
  - Developed by Society of Automotive Engineers heavy trucks and bus division (SAE)
  - Use of the 29 identifiers
  - Support of real-time close loop control
- **Applications**
  - Light to heavy trucks
  - Agriculture equipment e.g. tractors, harvester etc…
  - Engines for public work

Higher level protocols

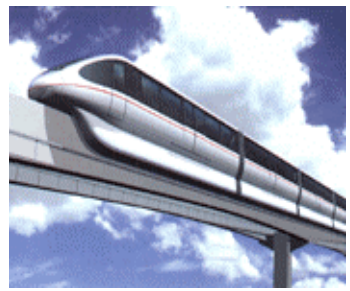


- **Features**
  - **Created by Honeywell**
  - **Close to DeviveNet, CAL & CANopen**

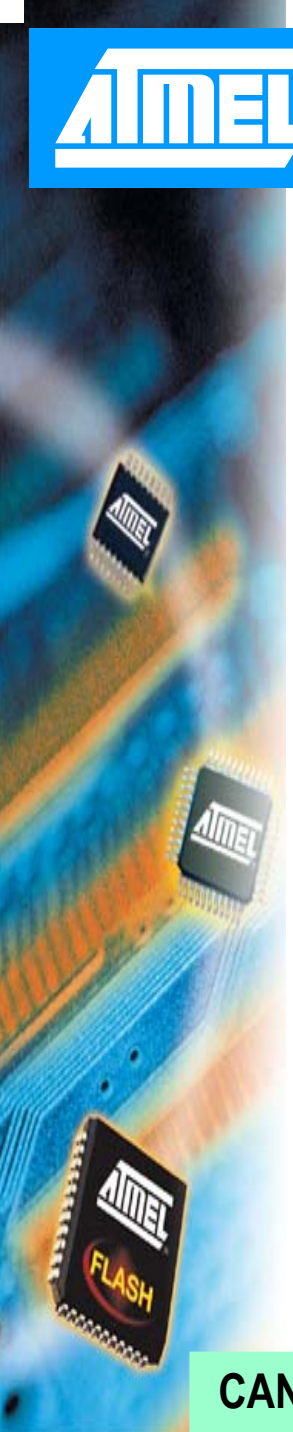
Higher level protocols

CAN: a Large Field of Applications

- Building Automatisation
- Domestic & Food distribution appliances
- Automotive & Transportation
- Robotic
- Production Automatisation
- Medical
- Agriculture

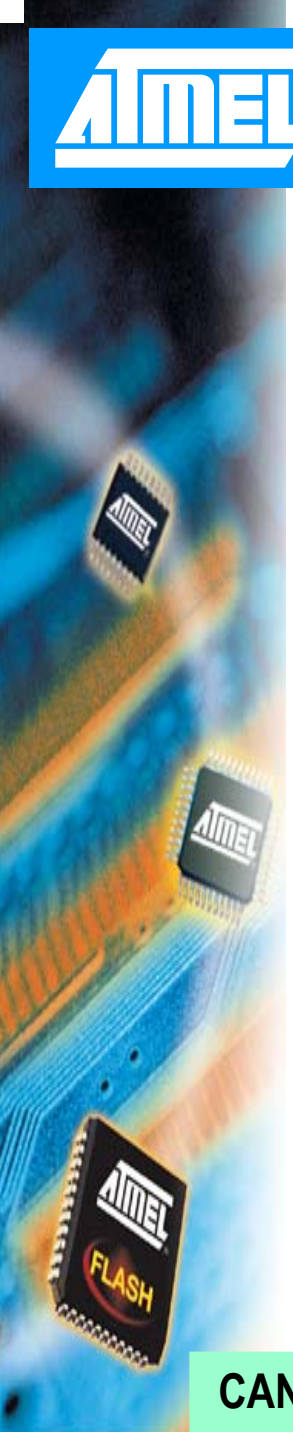


CAN applications

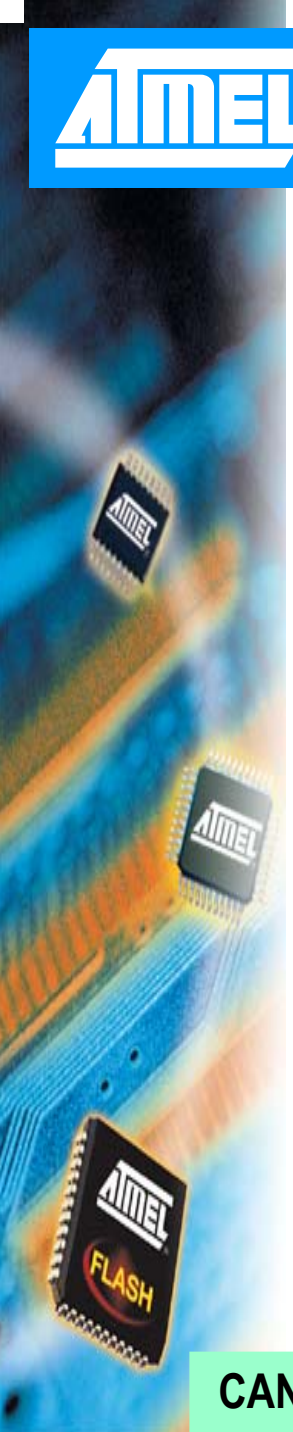


- Heating Control
- Air Conditioning (AC)
- Security (fire, burglar…)
- Access Control
- Light Control

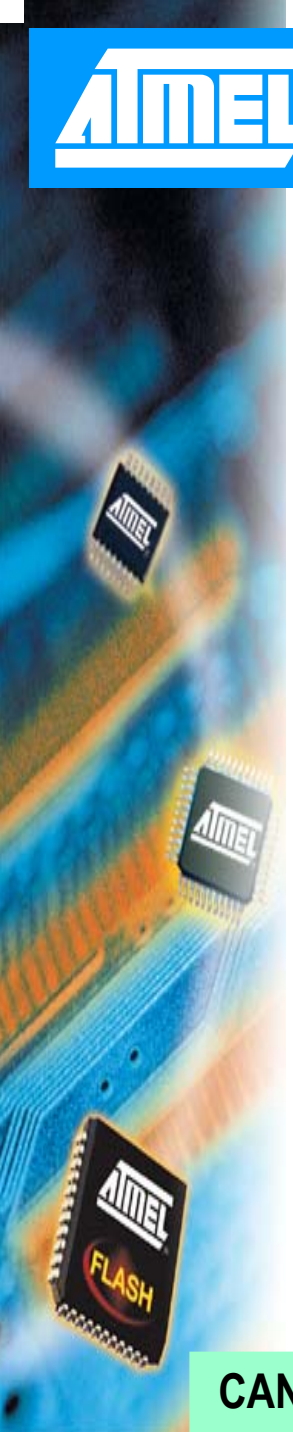
Domestic & Food distribution appliances



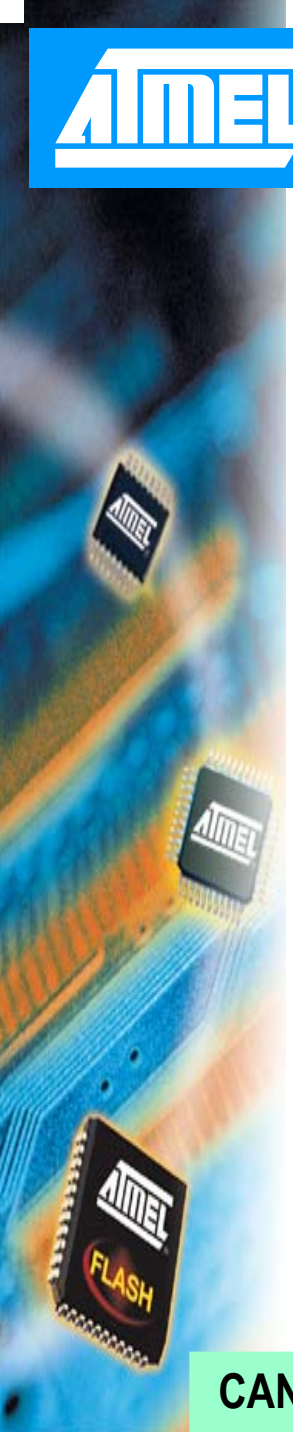
- Washing machines
- Dishes cleaner
- Self-service bottle distributors connected to internet



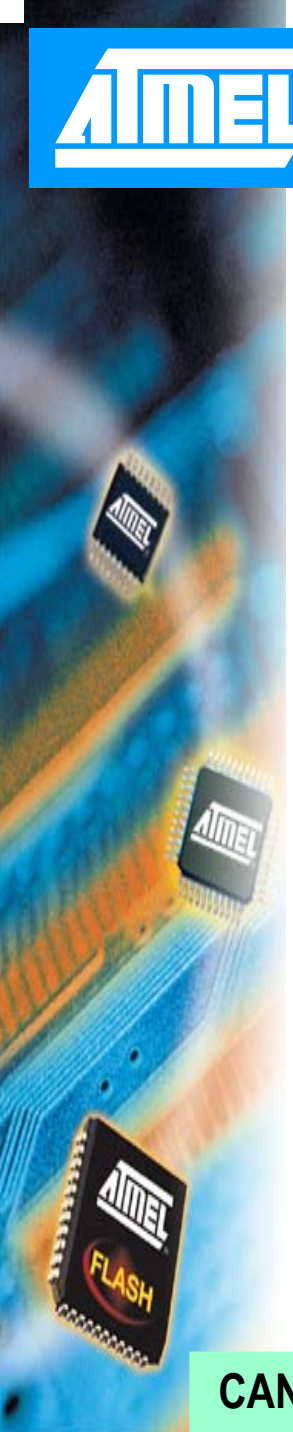
- Automotive
  - Dash board electronic
  - Comfort electronic
- Ship equipment
  - Train equipment
  - Lifts
  - Busses
  - Trucks
  - Storage transportation systems
  - Equipment for handicapped people
- Service & Analysis systems



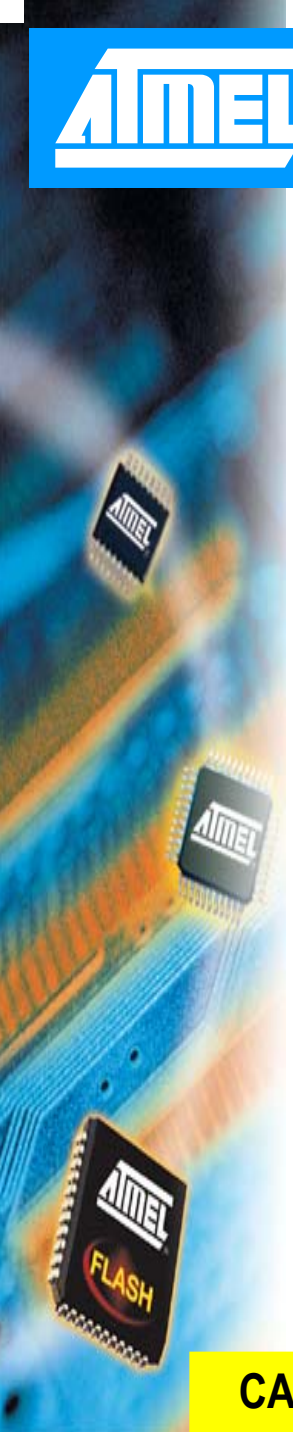
- Tool machines
- Transport systems
- Assembly machines
- Packaging machines
- Knitting machines
- Plastic injection machines
- etc...



- Control and link of production machines
- Production control
- Tool machines
- Transport systems
- Assembly machines
- Packaging machines
- Knitting machines
- Plastic injection machines
- etc...



- Harvester machines
- Seeding/Sowing machines
- Tractor control
- Control of live-stock breeding equipment

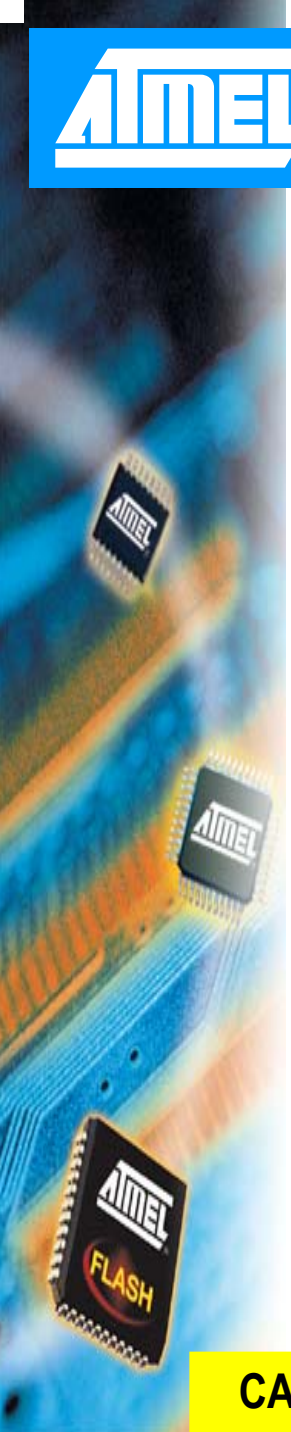


- Atmel CAN Bus Controller
  - Main Features
  - Mailbox concept (1)
  - Mailbox concept (2)
  - Channel Data Buffer (1)
  - Channel Data Buffer (2)
  - Autobaud & Listening Mode
  - Auto Reply Mode
  - Time Triggered Mode
  - Error Analysis Functions
  - CAN Self Test
  - Atmel CAN Controller
  - Conclusion
- CAN processor cores
  - Advanced C51 5 MIPS Core
  - Advanced AVR 16MIPS Core
- T89C51CC01
  - Block Diagram
  - Features (1) Features (2)
  - Advantages
- T89C51CC02
  - Block Diagram
  - Features (1) Features (2)
  - Advantages
- AT89C51CC03
- AT90CAN128
  - Block Diagram
  - Competitive advantages
- CAN family summary

CAN family

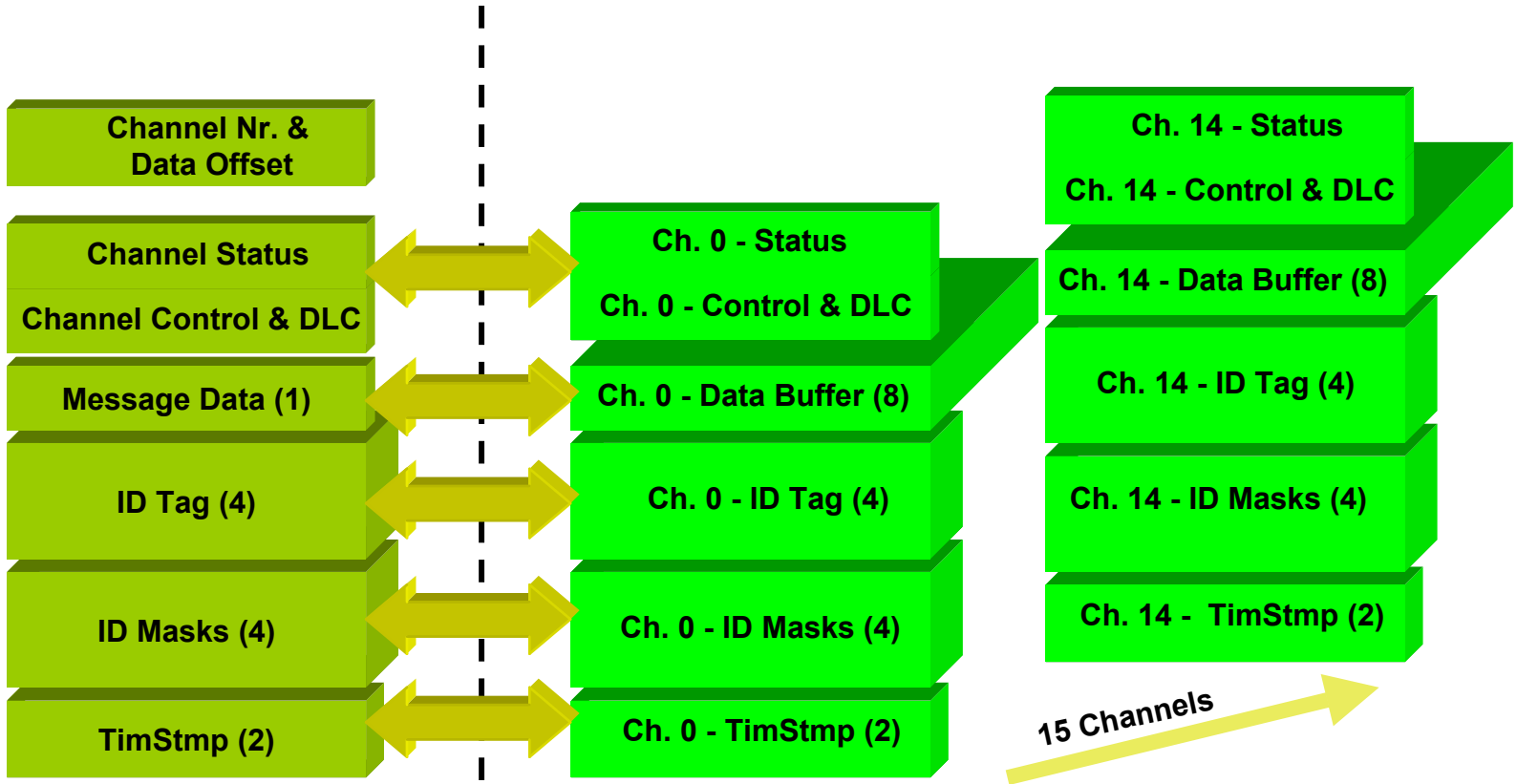
CAN Controller: Main Features

- Full validation by iVS/C&S Wolfenbüttel/Germany
- CAN 2.0A and 2.0B programmable / Channel
- 1 MHz CAN Bus Data Rate at 8 MHz Crystal
- 15 Channel with 20 Bytes of Control & Data / Channel
- 120 Bytes Reception Buffer
- Support of Time Triggered Communication (TTC)
- Auto Baud, Listening & Automatic Reply mode
- Mail Box addressing via indirect addressing
- All Channel features programmable on-the-fly
- Interrupt accelerator (available on AVR based controller)



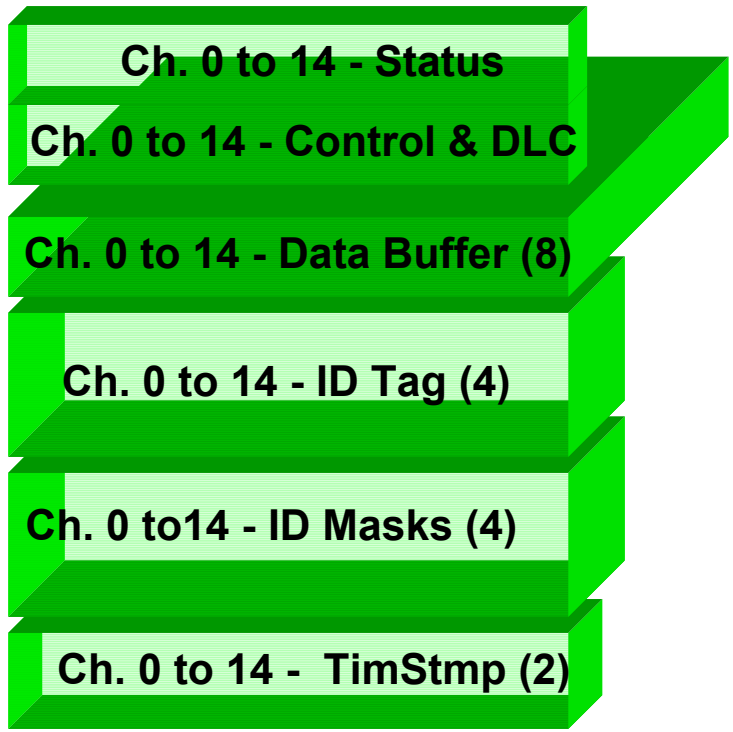
CAN family

CAN Controller: Mailbox concept (1)



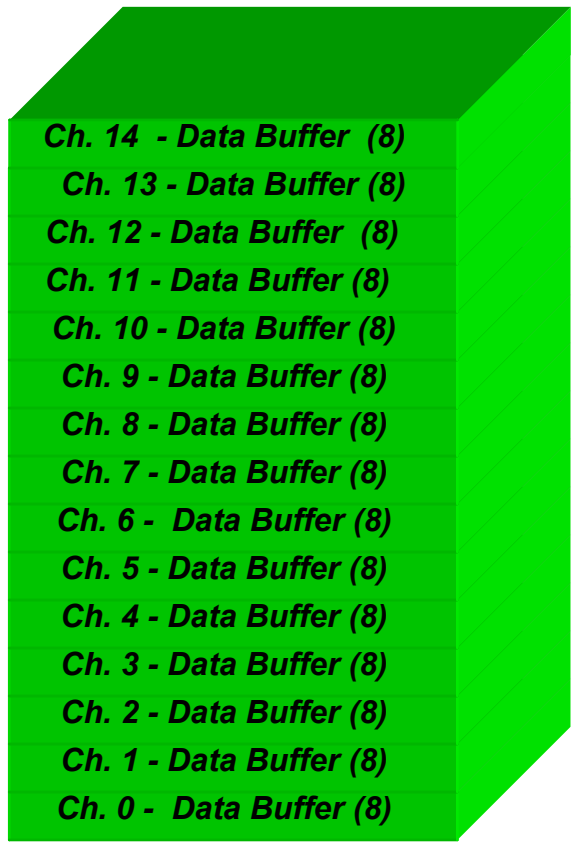
CAN Controller: Mailbox concept (2)

- Channel features
  - 32 bit of ID Mask Register
  - 32 bit of ID Tag Register
  - 64 bit of cyclic Data Buffer Register
  - 16 bit of Status, Control & DLC
  - 16 bit of Time Stamp Register



CAN Controller: Channel Data Buffer (1)

- **Main Features**
  - 15 Channels of 8 Byte (120 Bytes) Data Buffer
  - All Channels programmable as:
    - ✓ Receiver
    - ✓ Transmitter
    - ✓ Receiver Buffer
  - Highest Priority for lowest Channel Nr.
  - Interrupts at:
    - ✓ Correct Reception of Message
    - ✓ Correct Transmission
    - ✓ Reception Buffer full



CAN Controller: Channel Data Buffer (2)

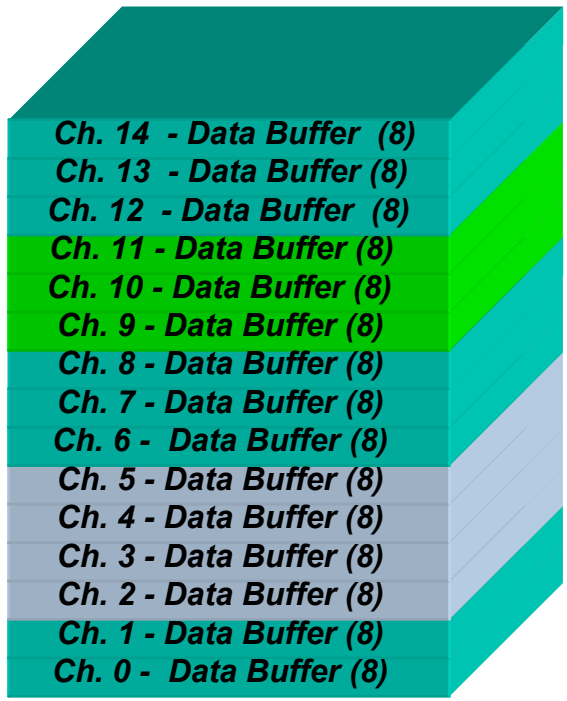
- **Reception Buffer Features:**
  - **Several Channels with same ID Mask (no important message will be missed)**
  - **Lowest Channel Number served first**
  - **Each Channel can participate (no consecutive sequence needed)**

Example:

Reception Buffer

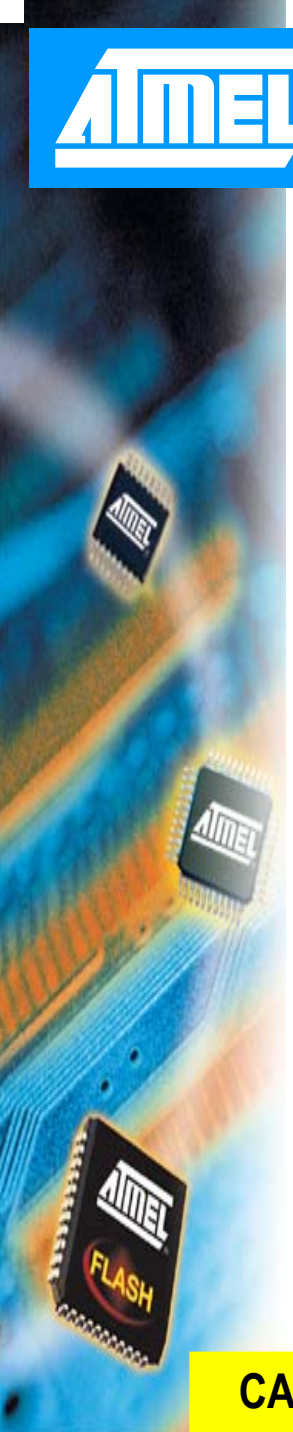
Receiver

Transmitter



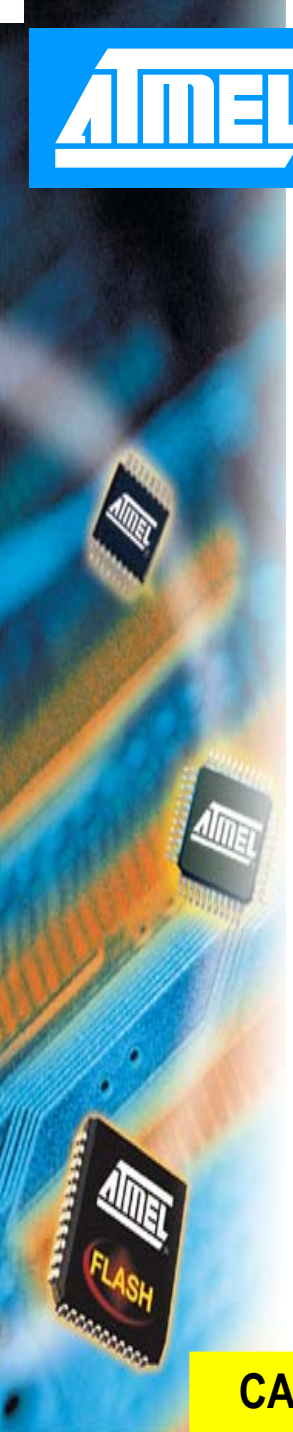
CAN Controller: Autoband & listening mode

- CAN monitoring without influence to the bus lines
  - No acknowledge by error frames
  - Error counters are frozen
  - Only reception possible
  - No transmission possible
  - Full error detection possible
- Bit-rate adaption support
  - Hot-plugging of bus nodes to running networks with unknown bit-rate



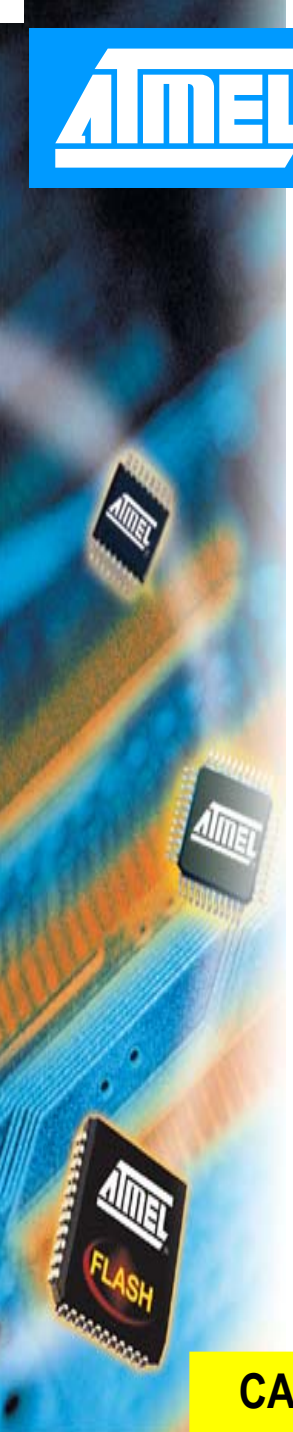
CAN Controller: Automatic Reply Mode

- Automatic Message Transfer
  - Automatic message transfer after
  - reception of Remote Frame
  - Deferred message transfer after
  - reception of Remote Frame
  - Automatic Retransmission of Data Frames under
  - Software control



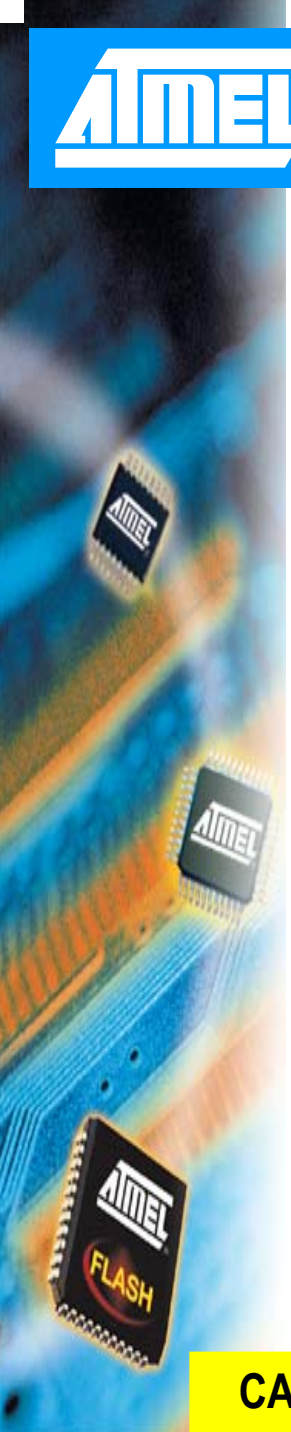
CAN Controller: Time Triggered Communication (TTC)

- **Support of Real Time Applications**
  - **Single shot transmission**
  - **16 bit CAN timer with IT at overflow**
  - **16 bit Time Stamp Register / Channel**
  - **Trigger for Time Stamp Register at**
  - **End of Frame (EOF) or Start of Frame (SOF)**



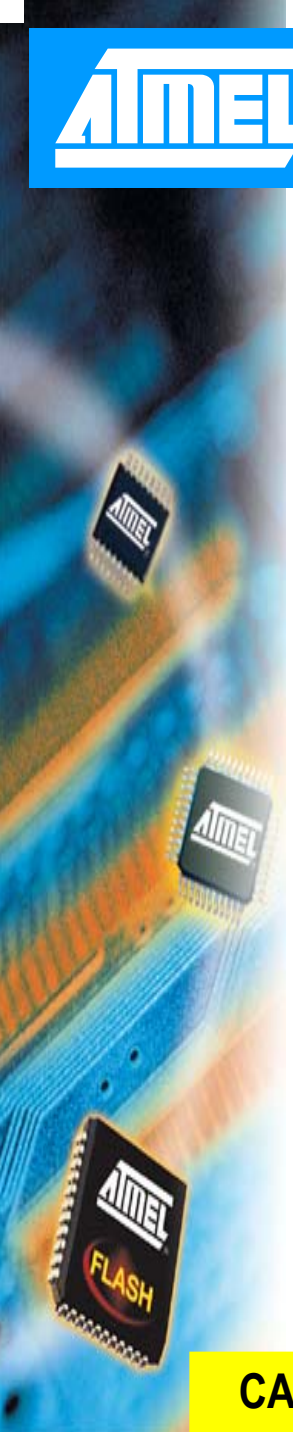
CAN Controller: Error Analysis Functions

- Channel Status Register (Error Capture Register)
  - Associated to each Channel
  - Type of CAN bus errors: DLC warning, Transmit OK, Receive OK, Bit error (on in transmit), Stuff error, CRC error, Form error, Acknowledgement error
- Error Interrupts
  - Bus errors, Error passive and Error warning
- Readable Error Counters



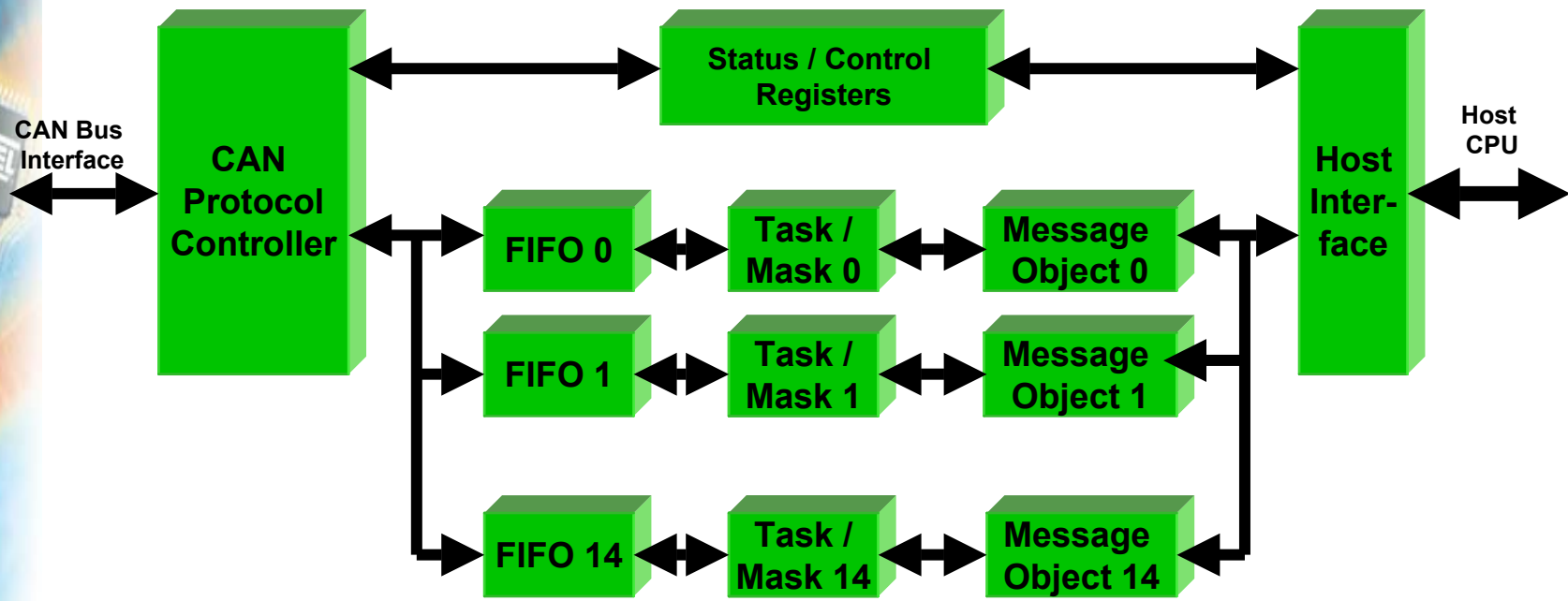
CAN Controller: CAN Self Test

- Analysis of own transmitted Message
- Support of local self test
- Support of global self test
- Software comparison of Tx & Rx buffer
- Monitoring of CAN bus traffic



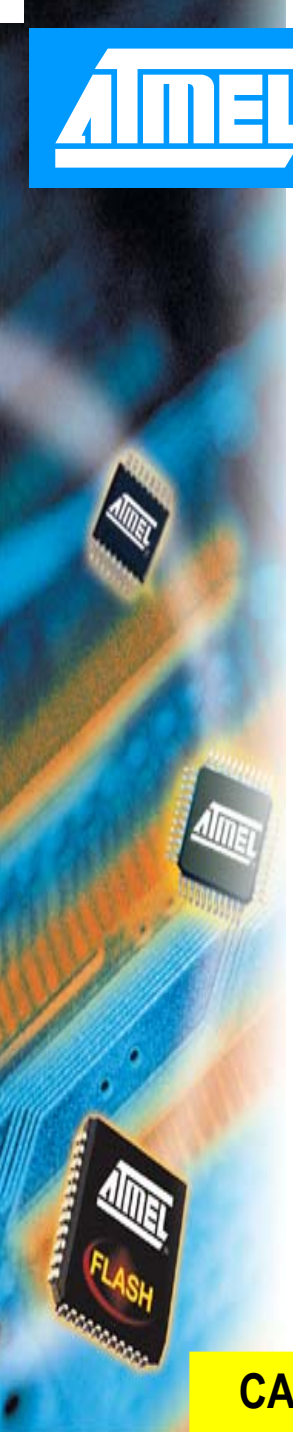
CAN Controller: The Atmel CAN controller

- 15 Message objects (Channels), each with filtering, masking and FIFO buffer
- All Channel features programmable on-the-fly



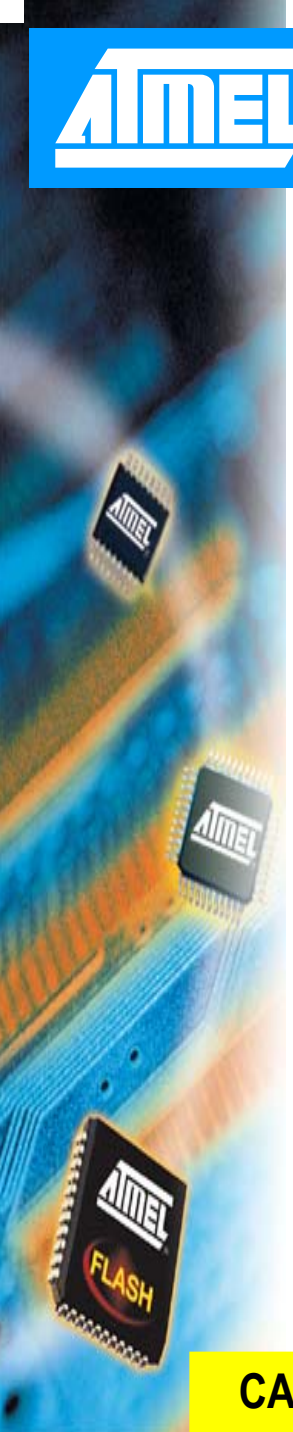
CAN family

CAN Controller: Conclusion



- 1 MHz/sec CAN Bus Data Rate at 8 MHz Crystal Freq.
- CAN 2.0A and 2.0B programmable / Channel
- 15 Channel with 20 Bytes of Control & Data / Channel
- 120 Bytes Reception Buffer
- Support of Time Triggered Communication (TTC)
- Auto Baud, Listening & Automatic Reply mode
- Mail Box addressing via SFRs
- All Channel features programmable on-the-fly
- Interrupt process accelerator with AVR based controller

CAN Microcontrollers: Advantages (1)

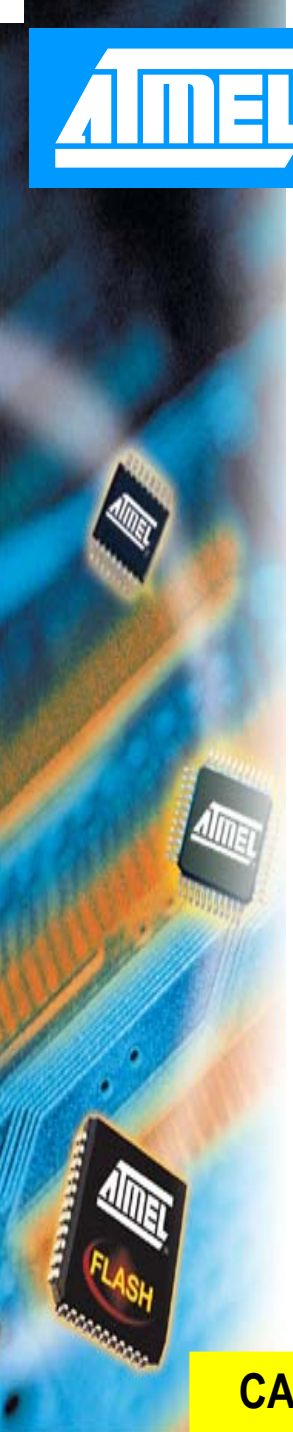


Advanced C51 Core

- 5MIPS (30MHz X2, 60MHz X1)
- Fully static operation
- Asynchronous port reset
- Second data pointer
- Inhibit ALE
- X2 CORE
- 4 level priority interrupt system
- Enhanced UART
- Programmable Timer 2 clock out
- Power Consumption reduction
- Wake up with external interrupts from Power Down

CAN family

CAN Microcontrollers: Advantages (2)

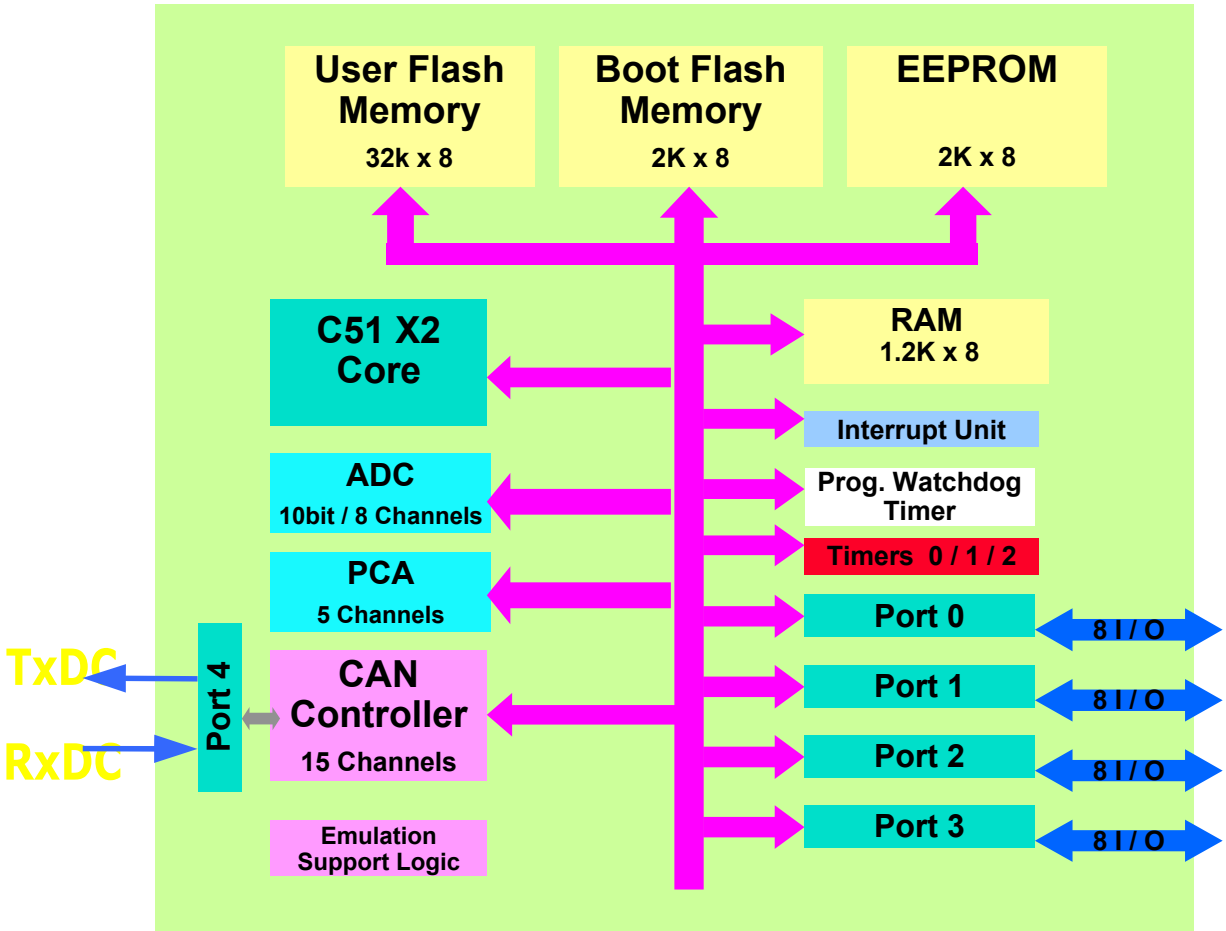


Advanced AVR Core

- 16MIPS core at 16MHz
- Hardware Multiplier
- IEEE 1149.1 Compliant JTAG Interface
- Instruction set optimized for C programming
- Self-Programming Memory
  - Remote Programming or Field Upgrade
  - Read While Write
  - Lock Bit/Brownout Protection
  - Variable Boot Block Size: 1 to 8KB
- Highest Code Density in C and Assembly
- Highest System Level Integration
- Complete Set of Development Tools

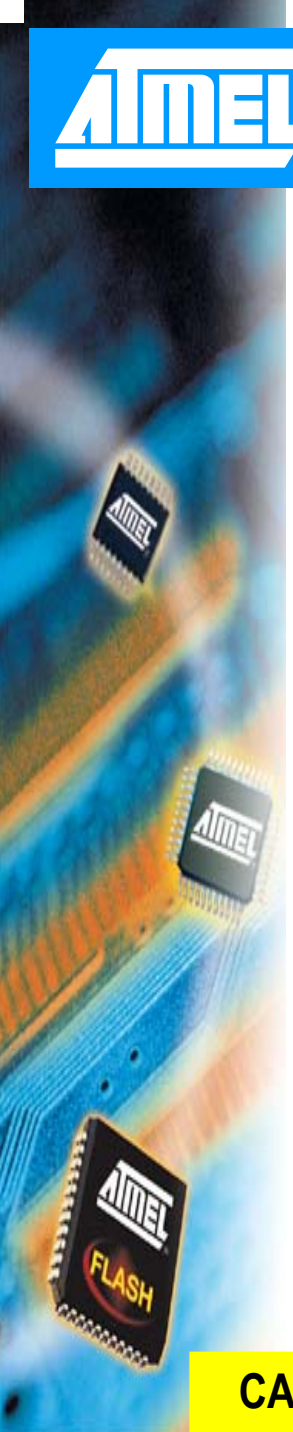
CAN family

CAN Microcontrollers: T89C51CC01 Block Diagram



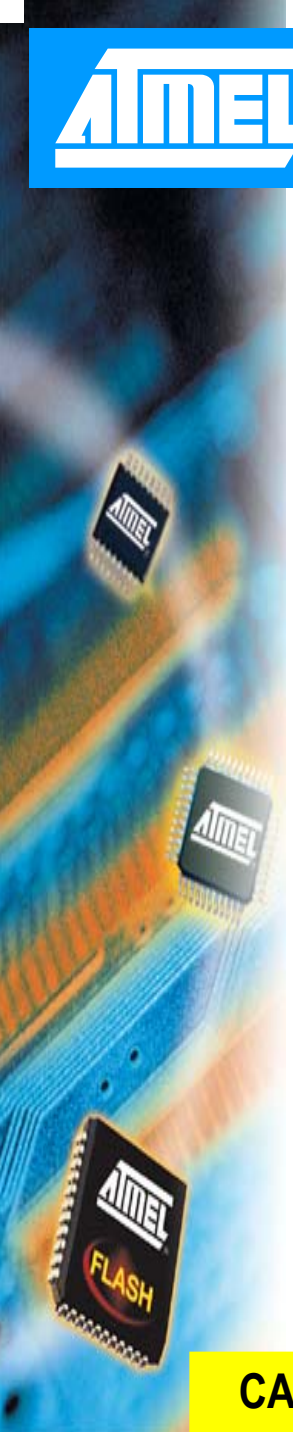
Packages: PLCC44, TQFP44, CA-BGA64

CAN Microcontrollers: T89C51CC01 Features



- C52 Core compatible
- Up to 60 MHz operation (X2 mode)
- X2 Core
- Double Data Pointer
- 32 Kb FLASH ISP, 2 Kb FLASH Boot Loader
- 2Kb EEPROM
- 1.25 k RAM (256b scratchpad RAM + 1kb XRAM)
- 3-16 bit Timers (T0,T1,T2)
- Enhanced UART
- CAN Controller with 15 channels (2.0A and 2.0B)

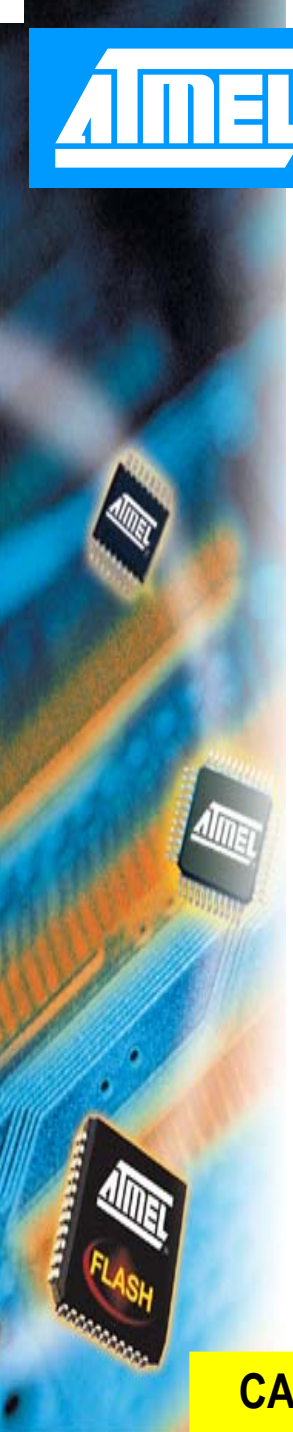
CANary Microcontrollers: T89C51CC01 Features



- 10 bits A/D with 8 Channels
- 5 I/O Ports
- Programmable Counter Array
  - 5 channels, 5 Modes:
    - ✓ PWM, Capture, Timer, Counter, Watchdog(Channel 4 only)
- 1Mbit /sec CAN at 8MHz Crystal Frequency (X2mode)
- Temperature: -40 to 85°C
- Voltage: 3 to 5 Volt +/-10%
- Packages: PLCC44, TQFP44, CA-BGA64

CAN family

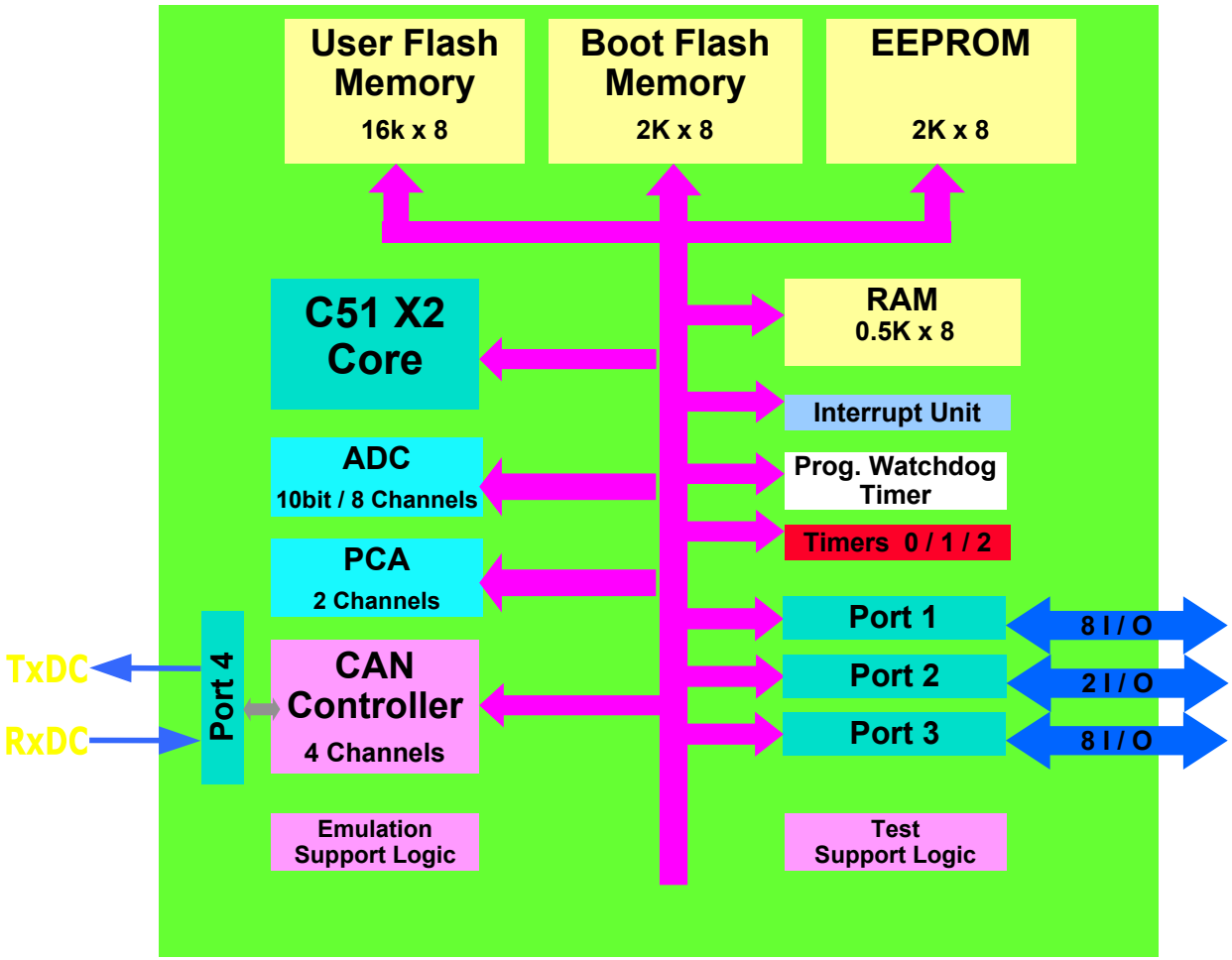
CAN Microcontrollers: T89C51CC01 Advantages



- T89C51CC01 is the first CAN Controller of a new generation for smart embedded applications which offers Flash and ISP Technology for Customer Code & Application Parameter up-date in a 44-pin package .
- For security reasons the 2kB Boot Memory is physically separated from 32kB Customer memory.
- Further for security reasons the Boot memory can be written only in Parallel Mode outside the application.
- 10b ADC & 5 channel PCA allow T89C51CC01 single-chip applications in most cases
- Included in the delivery is a wide range of Application Programming interfaces (API) concerning ISP, EEPROM, Security, Customer & Boot Flash

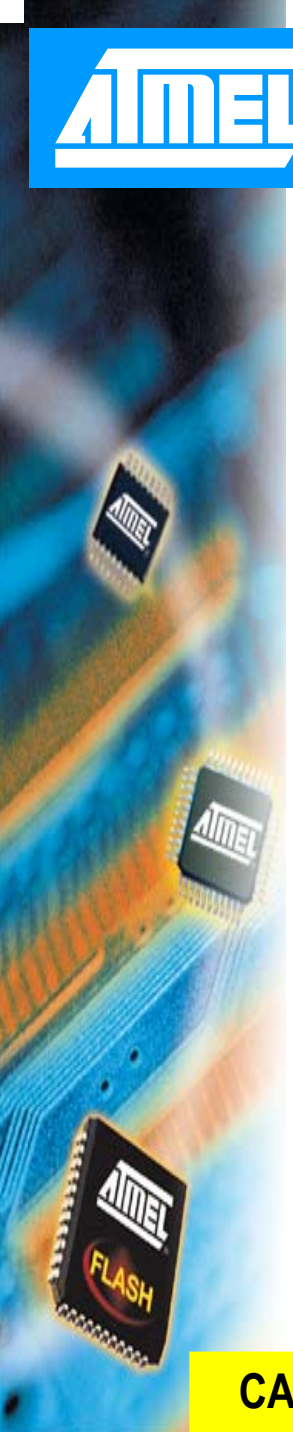
CAN family

CAN Microcontrollers: T89C51CC02 Block Diagram



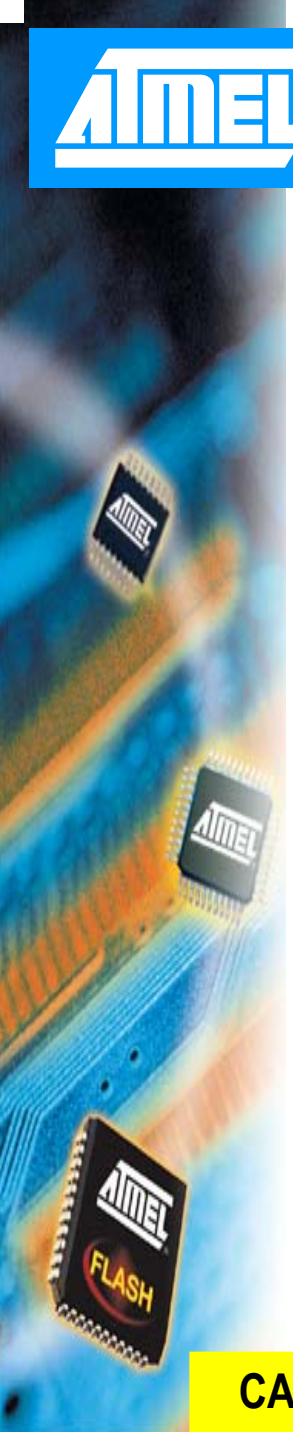
Packages: PLCC28, SOIC28, QPF32

CAN Microcontrollers: T89C51CC02 Features



- C52 Core and T89C51CC01 compatible
- Up to 60 MHz operation (X2 mode)
- X2 Core
- Double Data Pointer
- 16 kb FLASH ISP
- 2 Kb FLASH Boot Loader
- 2 kb EEPROM
- 512b RAM (256b scratchpad RAM + 256b XRAM)
- 3-16 bit Timers (T0,T1,T2)
- Enhanced UART
- CAN Controller with 4 channels (2.0A and 2.0B)

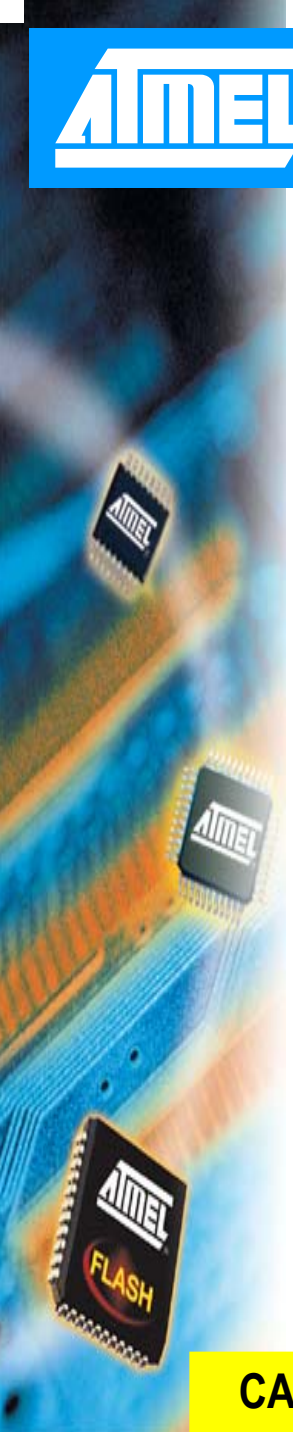
CANary Microcontrollers: T89C51CC02 Features



- 10 bit ADC with 8 Channels
- 3 I/O Ports
- Programmable Counter Array (PCA)
  - 2 channels, 5 Modes
    - PWM, Capture, Timer, Counter
- 1MBit/sec CAN at 8MHz Crystal Frequency (X2 mode)
- Temperature: -40 to 85°C
- Voltage: 3 to 5 Volt +/10%
- Package: SOIC28, Plcc28, TQFP32, TSSOP28

CAN family

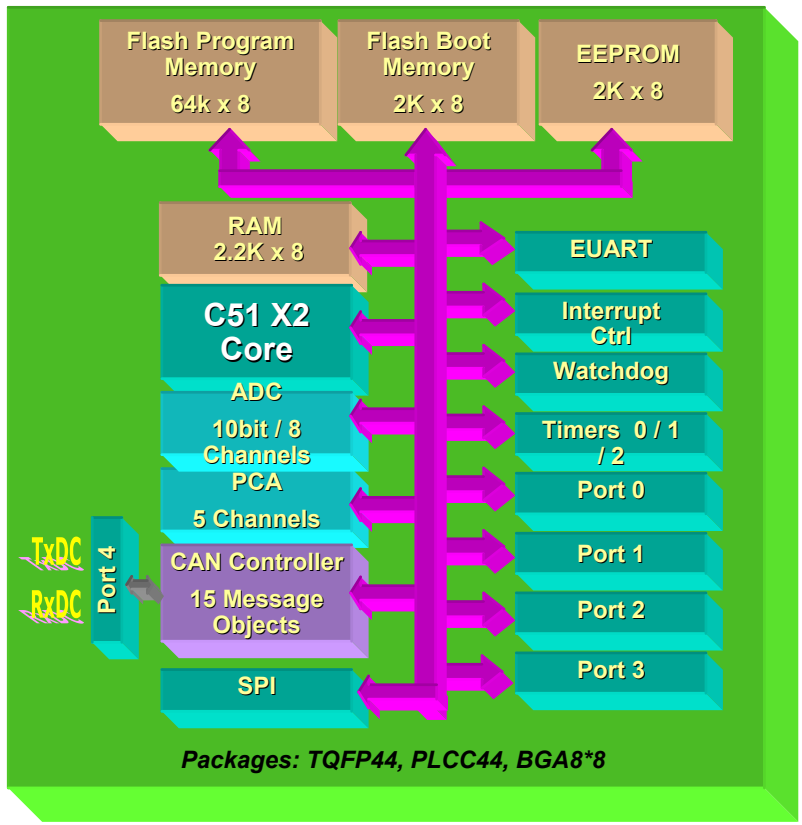
CAN Microcontrollers: T89C51CC02 Advantages



- T89C51CC02 is designed for embedded low-end, high volume applications
- Same functions like included in T89C51CC01
- Reduced costs in a 24 pin package.
- Main difference to T89C51CC01:
  - No access possible to external RAM/ROM via Ports 0 & 2
  - Customer Flash Memory 16kBytes
  - On-chip RAM: 512Bytes
  - 4 channel CAN Controller
  - 2 channel PCA
- All other functions will remain identical in the sense that the T89C51CC01 development tools can be used for T89C51CC02.

- Extend the CAN family to 64KB Flash and 2KB RAM
- Up to 60 MHz operation (X2 mode)
- Integrated Power Fail Detect (replace external BOD)
- Protection against false flash write
- Sport a fast SPI with Master and Slave mode
- Fully compatible with T89C51CC01 (32KB Flash)
- 3 volts to 5.5 volts
- UART bootloader and CAN bootloader
- PLCC44, VQFP44, BGA64, PLCC52(*) VQFP64(*) packages

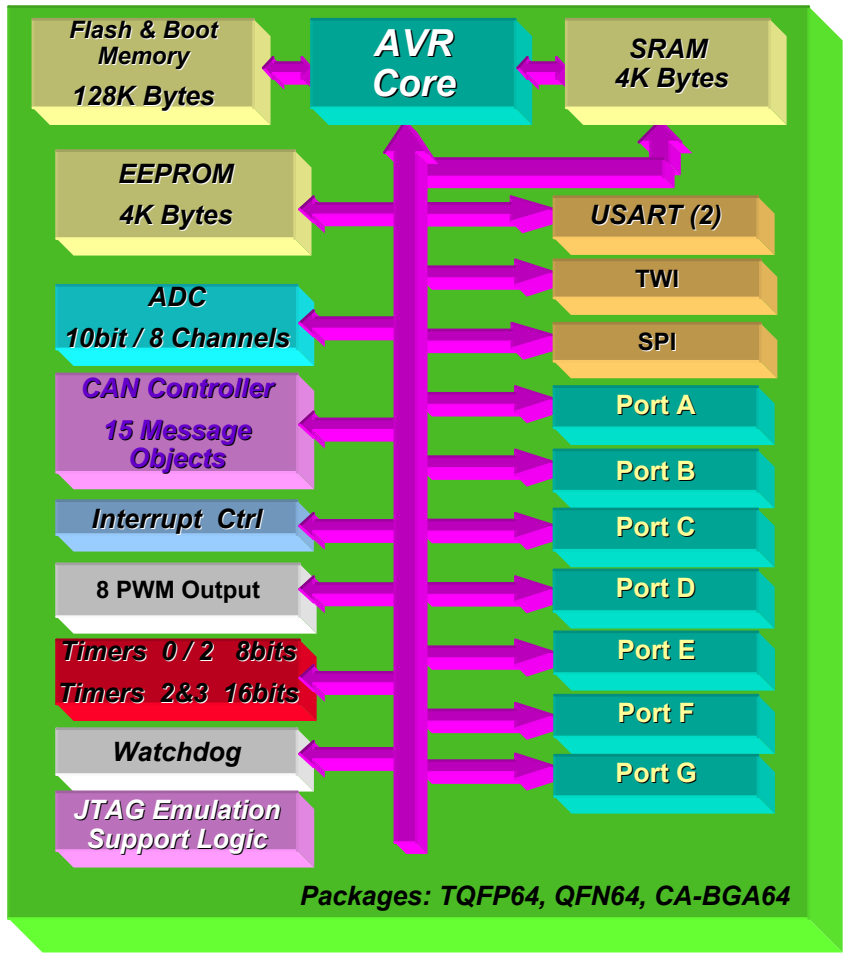
(*) with SPI interface



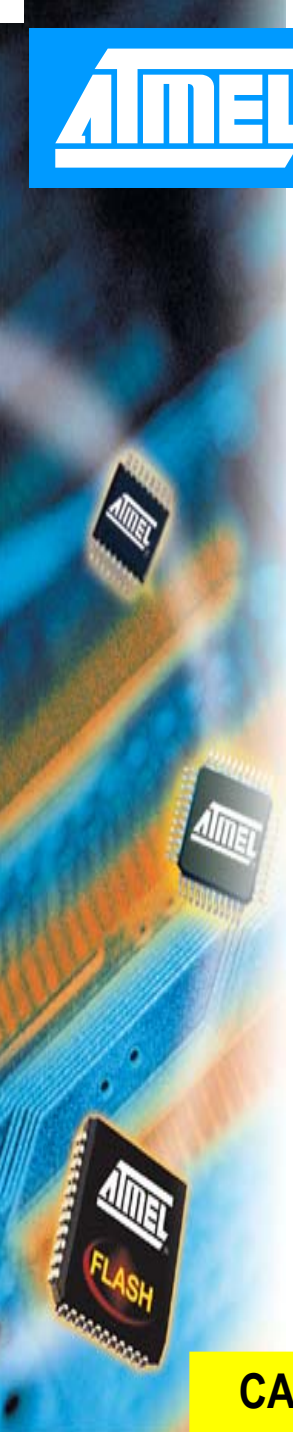
CAN family

Main Characteristics

- **8-Bit AVR Core**/1 MIPS per MHz (16MHz at 4.5V)
- **128KB Flash**
- **4KB RAM**
- **4KB EEPROM (100K cycles)**
- **CAN Controller CAN 2.0A/B with 15 MOB**
- **8-channel 10-bit ADC**
- **2 x 8-bit Timer/Counter0 and Timer/Counter2**
- **16-bit Timer/Counter 1/3**
- **Dual Programmable USART:** LIN capable
- **Two Wire Interface**
- **Programmable SPI:** master/slave
- **Programmable Brown-out detector**
- **TQFP64 + QFN64 + CA-BGA64 packages**



AT90CAN128 Competitive Advantage



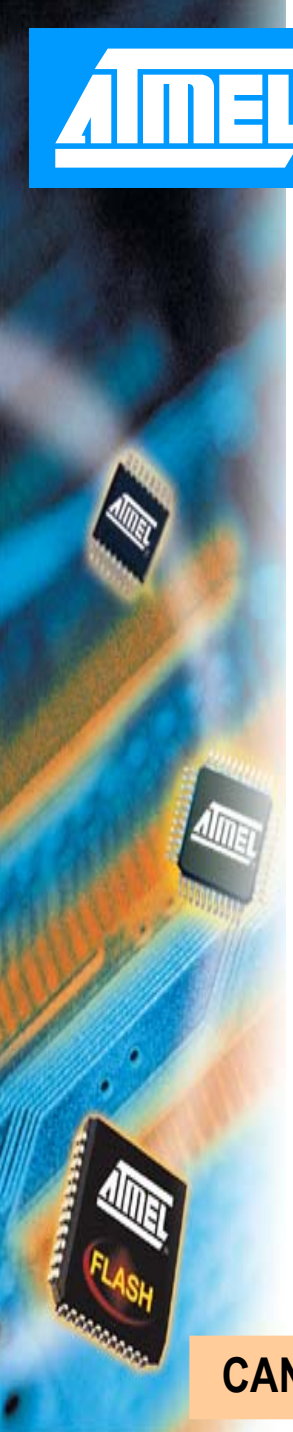
- **Performance:**
  - **Processing Speed: 16MIPS AVR RISC Core**
  - **128KB Flash Program Memory; 4KB EEPROM; 4KB RAM**
  - **V2.0A/B CAN Controller: Mail Box Message management up to 15 dynamic messages at the same time**
- **Flexibility:**
  - **Self-Programming Memory**
    - ✓ **Remote Programming or Field Upgrade**
    - ✓ **Read While Write**
    - ✓ **Lock Bit/Brownout Protection**
    - ✓ **Variable Boot Block Size: 1 to 8KB**
- **Higher Layer Protocol Software: CANopen and DeviceNet™ from Tool Vendors Partners**
- **Hardware Multiplier**
- **IEEE 1149.1 Compliant JTAG Interface**
- **Atmel Commitment to CAN Networking: a Complete CAN Microcontroller Family**

CAN family

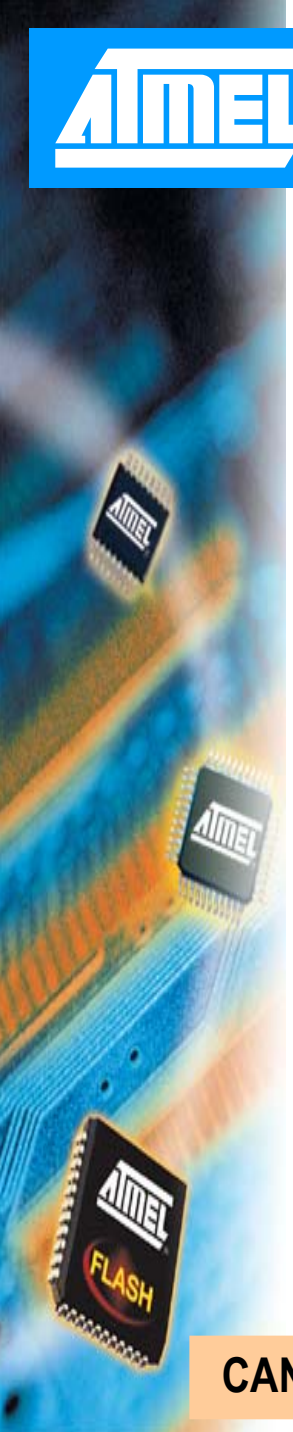
CAN Tutorial
AT90CAN128 in CAN Family

NEW

	8051 Architecture			AVR 8-Bit RISC
	T89C51CC01	T89C51CC02	AT89C51CC03	AT90CAN128
Introduce in	Y2000	Y2001	Y2003	Intro : March 2004
MIPS	5	5	5	16
Flash program/boot	32KB/2KB	16KB/2KB	64KB/2KB	128KB/ up to 8KB
EEPROM	2KB	2KB	2KB	4KB
RAM	1.2KB	0.5KB	2.2KB	4KB
Power Fail Detect	-	-	YES	YES
CAN Controller	15 Message Objects	4 Message Objects	15 Message Objects	15 Message Objects
SPI	-	-	YES	YES
Supply (V)	3 to 5.5	3 to 5.5	3 to 5.5	2.7 to 5.5
ADC	10 bit / 8 channels	10 bit / 8 channels	10 bit / 8 channels	10 bit / 8 channels
PCA	5 channels	2 channels	5 channels	-
Timers 8bit	-	-	-	Timers 0 / 2
Timers 16bit	Timers 0 / 1 / 2	Timers 0 / 1 / 2	Timers 0 / 1 / 2	Timer 1 / 3
UART (Hardware)	1	1	1	2
Port	Port 0 / 1 / 2 / 3	Port 1 / 2 / 3	Port 0 / 1 / 2 / 3	Port A/B/C/D/E/F/G
Bootloader	UART / CAN (DeviceNet – CANopen)	UART / CAN	Same UART / CAN as T89C51CC01	Hard : SPI, JTAG Soft : UART, CAN
Packages	TQFP44, PLCC44, CA-BGA64	SOIC24, SOIC28, TQFP32, PLCC28	TQFP44, PLCC44, BGA8*8	TQFP64, QFN64, BGA64



CAN REGISTERS
C51 base Core



CANBT1

-	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0	-

CANBT2

-	SJW1	SJW0	-	PRS2	PRS1	PRS0	-

CANBT3

-	PHS2-2	PHS2-1	PHS2-0	PHS1-2	PHS1-1	PHS1-0	SMP

BRP = prescaller. Tscl = (BRP+1)/FCAN

SJW = resynchronization Jump bit

PRS = propagation to compensate physical delay

PHS2 = phase error compensation after sampling point

PHS1 = phase error compensation before sampling point

SMP = sample type : 0 for 1 sample, 1 for 3 samples (1/2Tscl apart from center)

CANGCON

ABRQ	OVRQ	TTC	SYNCTTC	AUTOBAUD	TEST	ENA	GRES

General Control register

ABRQ : Abort request. An on going transmission or reception will be completed before the abort.

OVRQ : Overload frame request (see frame format 4 page for details)

TTC : Set to select the Time Trigger Communication mode

SYNCTTC : select Start of Frame or end of Frame for TTC synchronization

AUTOBAUD : listening mode only when set

Test : factory reserved bit

ENA : Enable CAN standby when bit=0

GRES : General reset

CANGSTA

-	OVFG	-	TBSY	RBSY	ENFG	BOFF	ERRP

General Status register

OVFG : Overload Frame Flag (set while the overload frame is sent. No IT)

TBSY : Transmitter Busy (Set while a transmission is in progress)

RBSY : Receive Busy (set while a reception is in progress)

ENFG : Enable On Chip CAN controller Flag. Cleared after completion of on going transmit or receive after ENA has been cleared in CANGCON

BOFF : Bus Off indication

ERRP : Error Passive indication

CANGIT

CANIT	-	OVRTIM	OVRBUF	SERG	CERG	FERG	AERG

General Interrupt

CANIT : Set if one at least of the 15 channels has an IT

OVRTIM : Overrun CAN TIMER (roll over FFFF to 0000) can generate an INT if ETIM bit in IE1 is set

OVRBUF : Overrun Buffer

SERG : Stuffing error detected

CERG : CRC error detected (the faulty channel will also get a CRC error in its CANSTCH)

FERG : Form error

AERG : Acknowledge error general : A transmit message has not been acknowledged (ACK bit was red at 1)

CANGIE

-	-	ENRX	ENTX	ENERCH	ENBUF	ENERG	-

General Interrupt enable register

ENRX : Enable receive interrupt

ENTX : Enable Transmit Interrupt

ENERCH : Enable message error (SERR, CERR, FERR, AERR) coming from any channel (See ENERG below)

ENBUF : Enable Buffer INT (OVRBUF)

ENERG : Enable general error (SERG, CERG, FERG, AERG) (See ENERCH above)

CANEN1

-	ENCH14	ENCH13	ENCH12	ENCH11	ENCH10	ENCH9	ENCH8

CANEN2

ENCH7	ENCH6	ENCH5	ENCH4	ENCH3	ENCH2	ENCH1	ENCH0

ENCHi = 0 for unused channel

ENCHi = 1 for message enable (ready to send or ready to receive)

ENCHi is set by rewriting the configuration in CANCONCHi

CANSIT1

-	SIT14	SIT13	SIT12	SIT11	SIT10	SIT9	SIT8

CANSIT2

SIT7	SIT6	SIT5	SIT4	SIT3	SIT2	SIT1	SIT0

Status Interrupt message object

SITi = 0 No Interrupt for channel i
SITi = 1 Interrupt request from channel i

CANIE1

-	IECH14	IECH13	IECH12	IECH11	IECH10	IECH9	IECH8

CANIE2

IECH7	IECH6	IECH5	IECH4	IECH3	IECH2	IECH1	IECH0

Enable Interrupt message object

IECHi = Interrupt disable for channel i
SITi = 1 Interrupt enable for channel i

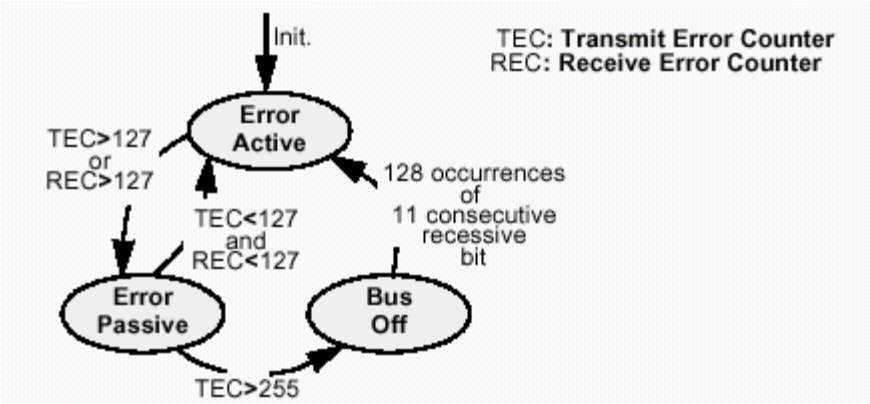
CANTEC

TEC7	TEC6	TEC5	TEC4	TEC3	TEC2	TEC1	TEC0

CANREC

REC7	REC6	REC5	REC4	REC3	REC2	REC1	REC0

CANTEC and CANREC : error counters



CAN Message Object pointer register

CANPAGE

CHNB3	CHNB2	CHNB1	CHNB0	AINC	INDX2	INDX1	INDX0

CHNB : Selection of the message object : 0 to 14

AINC : Auto-increment the index if AINC=0 . Successive access to CANMSG register will read or write the successive bytes (up to 8)

INDX : Byte location (0 to 7) in the data buffer array pointed by CANMSG.

C51 CAN Message object register (1)

CANCONCH

CONCH1	CONCH0	RPLV	IDE	DLC3	DLC2	DLC1	DLC0

CAN Message Object Control Register

CONCH : 00 Disable
01 Transmit
10 Receive
11 Receive Buffer

RPLV : Automatic reply (0 to reply not ready, 1 to reply ready)

IDE : 0 for CAN2.0A (11 bit Identifier)
1 for CAN2.0B (29 bit identifier)

DLC : Data length code (0 to 8) indicate the number of valid Bytes expected from a received message. Give number of valid byte to transmit for a transmit message.

C51 CAN Message object register (2)

CANSTCH

DLCW	TXOK	RXOK	BERR	SERR	CERR	FERR	AERR

Message Object status register

DLCW : Error number of receive byte is not equal to CANCONCH DLC
TXOK : transmit OK
RXOK : receive OK
BERR : transmit bit error.
Recessive bit detected while a dominant bit was sent.
Or Dominant ack bit during an error frame transmission
SERR : Stuffing error
CERR : CRC error
FERR : Form error
AERR : Ack error : no detection of a dominant bit in the ack slot

Message object register IDT CAN2.0A (3)

CANIDT1

IDT10	IDT9	IDT8	IDT7	IDT6	IDT5	IDT4	IDT3

CANIDT2

IDT2	IDT1	IDT0	-	-	-	-	-

CANIDT3

-	-	-	-	-	-	-	-

CANIDT4

-	-	-	-	-	RTRTAG	-	RBOTAG

IDT = transmit Identifier or expected receive Identifier (see also mask)

RTRTAG : Remote transmit request tag

RB0TAG : reserved bit

Message object register IDT CAN2.0B (4)

CANIDT1

IDT28	IDT27	IDT26	IDT25	IDT24	IDT23	IDT22	IDT21

CANIDT2

IDT20	IDT19	IDT18	IDT17	IDT16	IDT15	IDT14	IDT13

CANIDT3

IDT12	IDT11	IDT10	IDT9	IDT8	IDT7	IDT6	IDT5

CANIDT4

IDT4	IDT3	IDT2	IDT1	IDT0	RTRTAG	RB1TAG	RBOTAG

IDT = transmit Identifier or expected receive Identifier (see also mask)

RTRTAG : Remote transmit request tag

RB1TAG RB0TAG : reserved bit

Message object register IDM CAN2.0A (5)

CANIDM1

IDMSK10	IDMSK9	IDMSK8	IDMSK7	IDMSK6	IDMSK5	IDMSK4	IDMSK3

CANIDM2

IDMSK2	IDMSK1	IDMSK0	-	-	-	-	-

CANIDM3

-	-	-	-	-	-	-	-

CANIDM4

-	-	-	-	-	RTRMSK	-	IDEMSK

IDM = Receive Identifier mask(see also IDT)

RTMSK : remote transmission request mask value : 0 comparison true forced, 1 bit comparison enable

IDEMSK : Identifier extension mask value : 0 comparison true forced 1 comparison enable (detect CAN2.0b reception while CAN2.0a expected)

Message object register IDM CAN2.0B (6)

CANIDM1

IDMSK28	IDMSK27	IDMSK26	IDMSK25	IDMSK24	IDMSK23	IDMSK22	IDMSK21

CANIDM2

IDMSK20	IDMSK19	IDMSK18	IDMSK17	IDMSK16	IDMSK15	IDMSK14	IDMSK13

CANIDM3

IDMSK12	IDMSK11	IDMSK10	IDMSK9	IDMSK8	IDMSK7	IDMSK6	IDMSK5

CANIDM4

IDMSK4	IDMSK3	IDMSK2	IDMSK1	IDMSK0	RTRMSK	-	IDEMSK

IDM = Receive Identifier mask(see also IDT)

RTMSK : remote transmission request mask value : 0 comparison true forced, 1 bit comparison enable

IDEMSK : Identifier extension mask value : 0 comparison true forced 1 comparison enable (detect CAN2.0A reception while CAN2.0B expected)

C51 CAN Message object register (7)

CANMSG

MSG7	MSG6	MSG5	MSG4	MSG3	MSG2	MSG1	MSG0

Can Message . If auto increment is programmed in CANCONCH, the index will be automatically incremented after each write or read into CANMSG (count = 0 to 7)

CANTCON

TPRESC7	TPRESC6	TPRESC5	TPRESC4	TPRESC3	TPRESC2	TPRESC1	TPRESC0

Prescaller for Timer Clock control (clock for CANTIMH/L)
The prescaller clock input is Fcan/6

CANTIMH

CANGTIM15	CANGTIM14	CANGTIM13	CANGTIM12	CANGTIM11	CANGTIM10	CANGTIM9	CANGTIM8

CANTIML

CANGTIM7	CANGTIM6	CANGTIM5	CANGTIM4	CANGTIM3	CANGTIM2	CANGTIM1	CANGTIM0

CAN general timer (16 bit) receives clock from the prescaller CANTCON

Timestamp register 1 per message object (3)

CANSTAMPH

TIMSTMP15	TIMSTMP14	TIMSTMP13	TIMSTMP12	TIMSTMP11	TIMSTMP10	TIMSTMP9	TIMSTMP8

CANSTAMPL

TIMSTMP7	TIMSTMP6	TIMSTMP5	TIMSTMP4	TIMSTMP3	TIMSTMP2	TIMSTMP1	TIMSTMP0

CAN TIME STAMP (16 bit)

CANTIM value stored in CANSTAMP with TXOK or RXOK

One TimeStamp register per message object

C51 CAN TTC register General register

CANTTCH

TIMTTC15	TIMTTC14	TIMTTC13	TIMTTC12	TIMTTC11	TIMTTC10	TIMTTC9	TIMTTC8

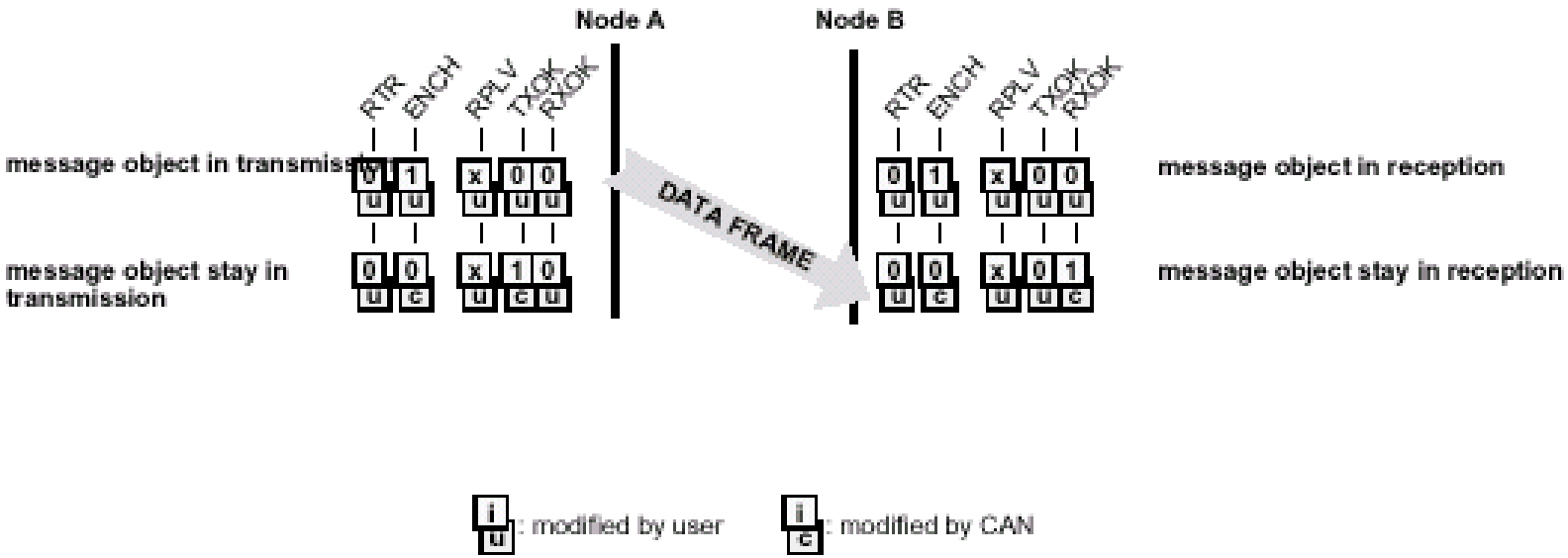
CANTTCL

TIMTTC7	TIMTTC6	TIMTTC5	TIMTTC4	TIMTTC3	TIMTTC2	TIMTTC1	TIMTTC0

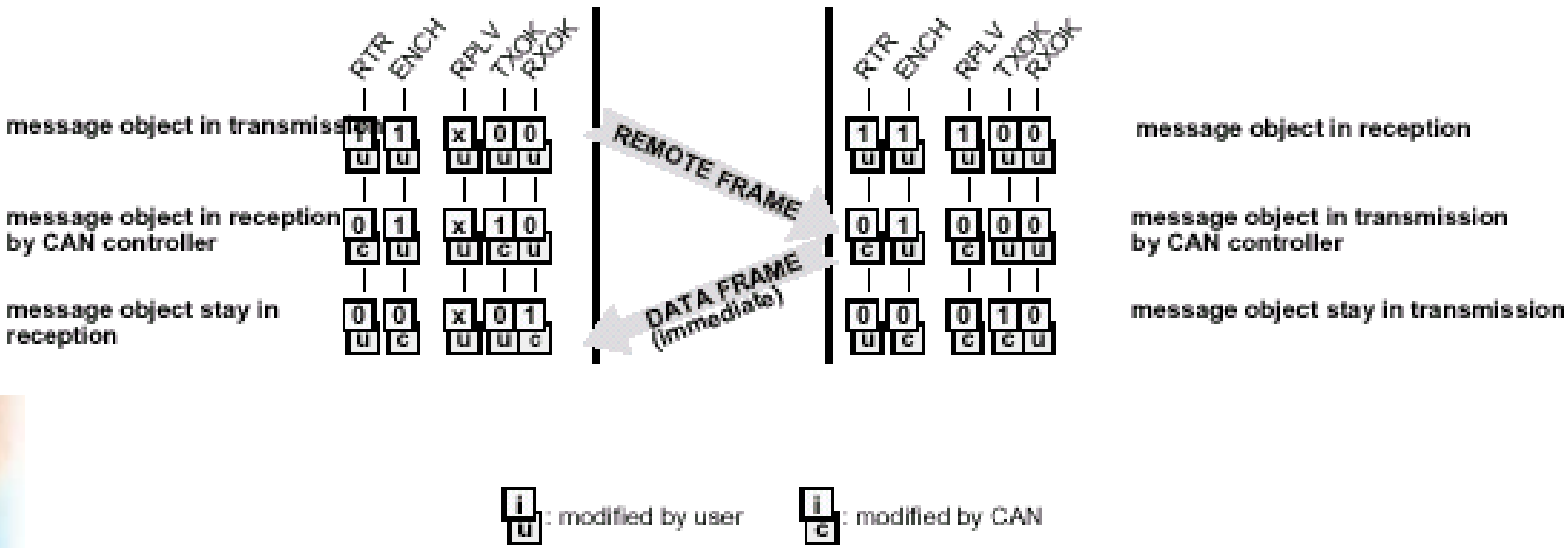
CAN Time Trigger Communication TTC (16 bit)

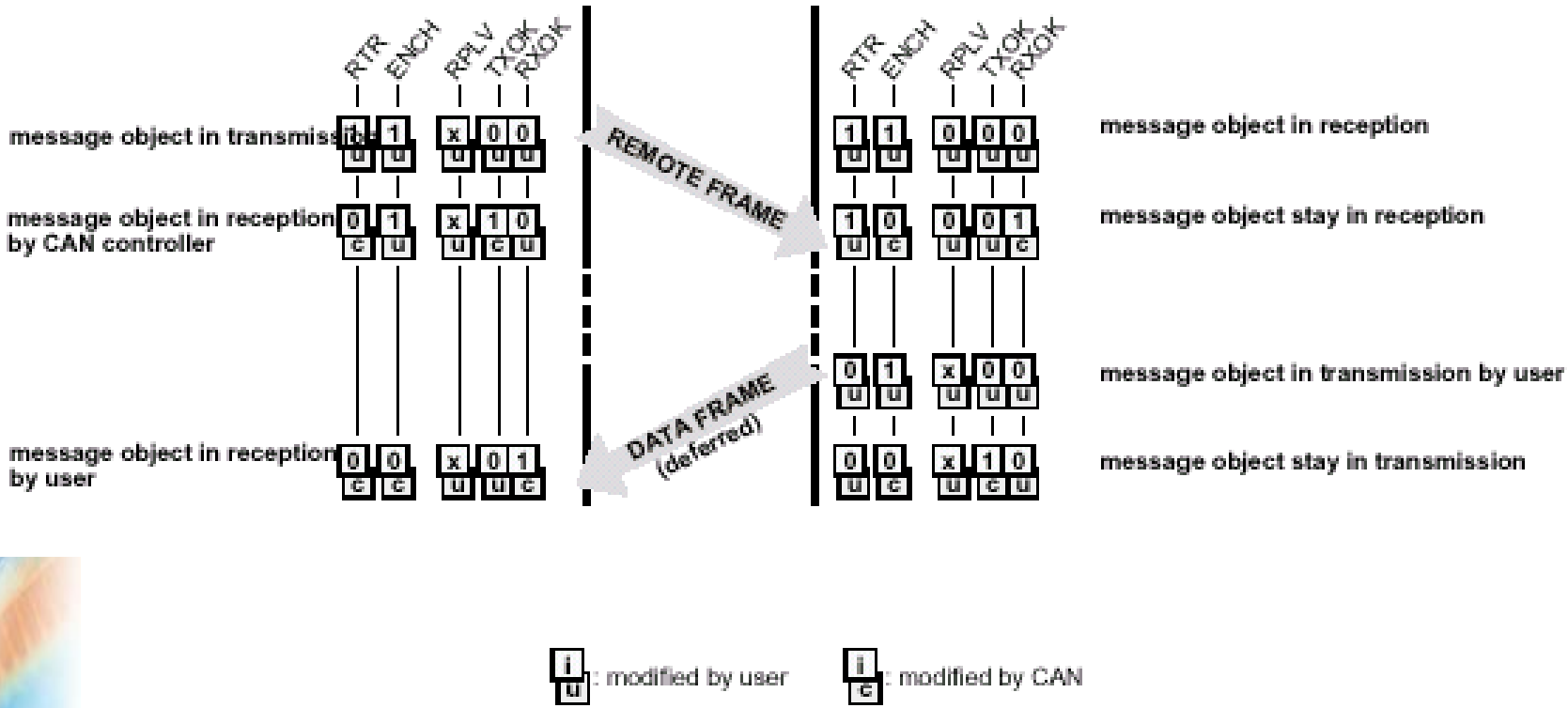
CANTIM value stored in CANTTC at start of Frame if SYNCTTC=1 or End Of Frame if SYNCTTC=0

Only one CANTTC register


Node A
Node B
RTR
IDCH
RPLV
TXOK
RXOK
message object in transmission
message object stay in transmission
DATA FRAME
message object in reception
message object stay in reception
: modified by user
: modified by CAN

CAN Tutorial
Detail remote frame with automatic reply



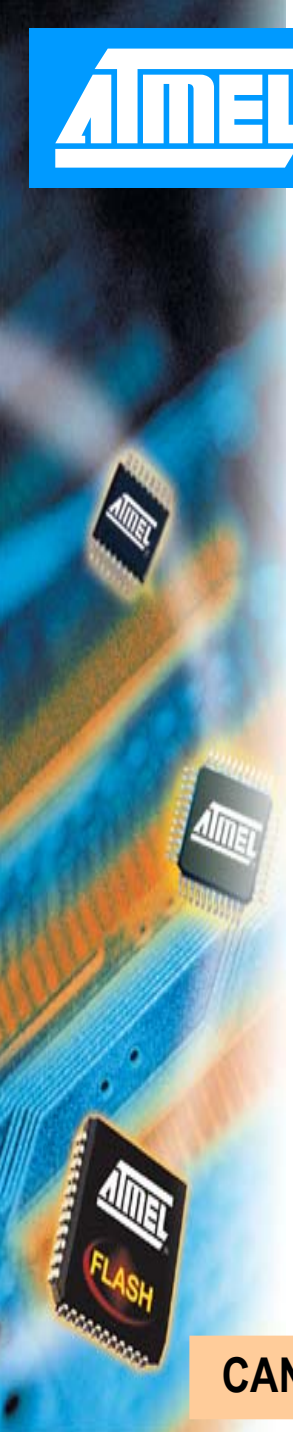

RTR
IDMSK
RPLV
TXOK
RXOK
message object in transmission by user
message object in reception by CAN controller
message object in reception by user
REMOTE FRAME
DATA FRAME (deferred)
message object in reception
message object stay in reception
message object in transmission by user
message object stay in transmission
: modified by user
: modified by CAN

C51 CAN How to start a CAN transmission

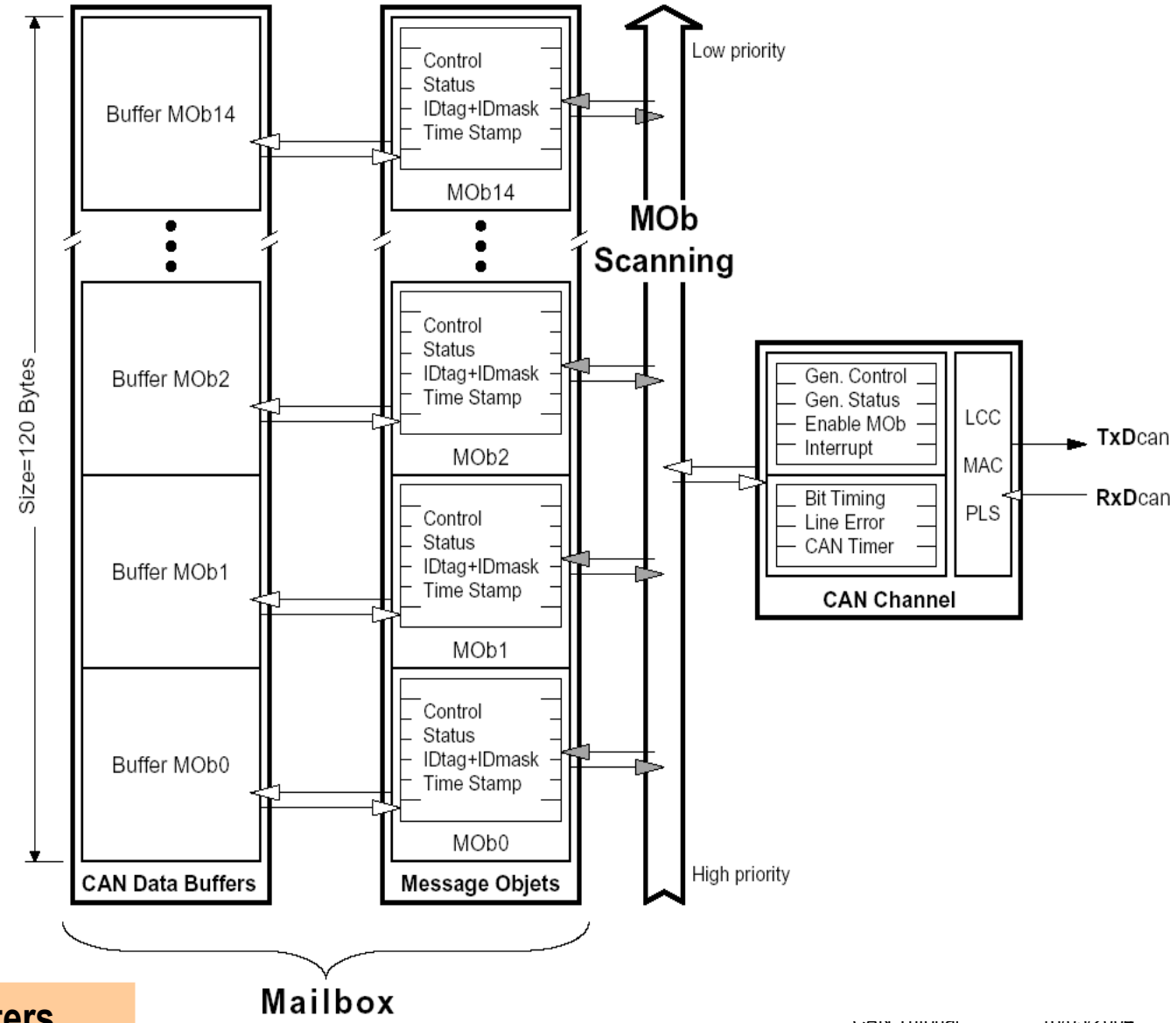
- Reset CAN Controller — CANGCON = 0x01
- Disable CAN IT — IE1.0=0
- Disable CAN Timer IT (TTC) — IE1.2=0
- Initialize all Message Object (num = 0 to 14)
  - CANPAGE = num shl(4)
  - CANCONCH = 0 , CANSTCH=0, CANIDT(1:4)=0, CANIDM(1:4)=0
  - For n=1 to 8 do CANMSG=0
- Initialize BIT timings: CANBT1, CANBT2, CANBT3
- Enable CAN controller: — CANGCON = 0x02
- Configure one Message Object for TX
  - Select CANPAGE
  - Set CANIDT1 and CANIDT2
  - Set CANMSG with message content (Auto-increment)
  - Enable Message as Tx/6 Bytes/2.0B — CANCONCH = 0x56
- Enable interrupts

C51 CAN How to serve a CAN reception Interrupt

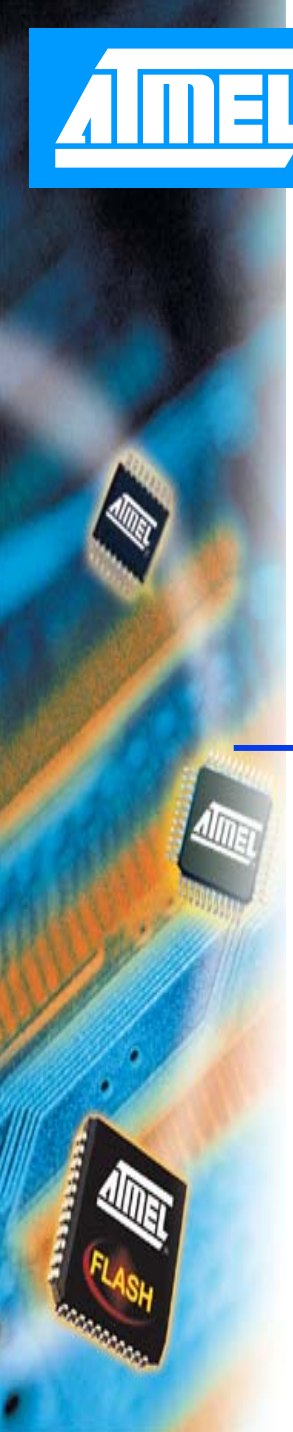
- Read CANGIT for CANIT and possible errors
- Read CANSIT1 & CANSIT2 to identify the channel (channel_I)
- Program the CANPAGE with the CHNB=I, AINC and INDEX
- Read CANSTCH for RXOK or a possible error
- For n=0 to n=7 : read CANMSG to read the message.
- Read CANSTMPH CANSTMPL (if time stamp is used)
- Rewrite CANCONCH CONCH[1:0] = 10 to re-enable the channel for a new reception



CAN REGISTERS
AVR base Core


Buffer MOb14
Buffer MOb2
Buffer MOb1
Buffer MOb0
Size=120 Bytes
CAN Data Buffers
Control
Status
IDtag+IDmask
Time Stamp
MOb14
MOb2
MOb1
MOb0
Message Objets
Low priority
MOb
Scanning
High priority
Gen. Control
Gen. Status
Enable MOb
Interrupt
Bit Timing
Line Error
CAN Timer
LCC
MAC
PLS
CAN Channel
TxDcan
RxDcan
Mailbox

CAN Tutorial
AVR CAN Registers map



Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0xFF)	Reserved									
(0xFE)	Reserved									
(0xFD)	Reserved									
(0xFC)	Reserved									
(0xFB)	Reserved									
(0xFA)	CANMSG	MSG 7	MSG 6	MSG 5	MSG 4	MSG 3	MSG 2	MSG 1	MSG 0	page 258
(0xF9)	CANSTMH	TIMSTM15	TIMSTM14	TIMSTM13	TIMSTM12	TIMSTM11	TIMSTM10	TIMSTM9	TIMSTM8	page 258
(0xF8)	CANSTML	TIMSTM7	TIMSTM6	TIMSTM5	TIMSTM4	TIMSTM3	TIMSTM2	TIMSTM1	TIMSTM0	page 258
(0xF7)	CANIDM1	IDMSK28	IDMSK27	IDMSK26	IDMSK25	IDMSK24	IDMSK23	IDMSK22	IDMSK21	page 257
(0xF6)	CANIDM2	IDMSK20	IDMSK19	IDMSK18	IDMSK17	IDMSK16	IDMSK15	IDMSK14	IDMSK13	page 257
(0xF5)	CANIDM3	IDMSK12	IDMSK11	IDMSK10	IDMSK9	IDMSK8	IDMSK7	IDMSK6	IDMSK5	page 257
(0xF4)	CANIDM4	IDMSK4	IDMSK3	IDMSK2	IDMSK1	IDMSK0	RTRMSK	–	IDEMSK	page 257
(0xF3)	CANIDT1	IDT28	IDT27	IDT26	IDT25	IDT24	IDT23	IDT22	IDT21	page 256
(0xF2)	CANIDT2	IDT20	IDT19	IDT18	IDT17	IDT16	IDT15	IDT14	IDT13	page 256
(0xF1)	CANIDT3	IDT12	IDT11	IDT10	IDT9	IDT8	IDT7	IDT6	IDT5	page 256
(0xF0)	CANIDT4	IDT4	IDT3	IDT2	IDT1	IDT0	RTRTAG	RB1TAG	RB0TAG	page 256
(0xEF)	CANCDMOB	CONMOB1	CONMOB0	RPLV	IDE	DLC3	DLC2	DLC1	DLC0	page 255
(0xEE)	CANSTMOB	DLCW	TXOK	RXOK	BERR	SERR	CERR	FERR	AERR	page 254
(0xED)	CANPAGE	MOBNB3	MOBNB2	MOBNB1	MOBNB0	AINC	INDX2	INDX1	INDX0	page 254
(0xEC)	CANHPMOB	HPMOB3	HPMOB2	HPMOB1	HPMOB0	CGP3	CGP2	CGP1	CGP0	page 254
(0xEB)	CANREC	REC7	REC6	REC5	REC4	REC3	REC2	REC1	REC0	page 253
(0xEA)	CANTEC	TEC7	TEC6	TEC5	TEC4	TEC3	TEC2	TEC1	TEC0	page 253
(0xE9)	CANTTCH	TIMTTC15	TIMTTC14	TIMTTC13	TIMTTC12	TIMTTC11	TIMTTC10	TIMTTC9	TIMTTC8	page 253
(0xE8)	CANTTCL	TIMTTC7	TIMTTC6	TIMTTC5	TIMTTC4	TIMTTC3	TIMTTC2	TIMTTC1	TIMTTC0	page 253
(0xE7)	CANTIMH	CANTIM15	CANTIM14	CANTIM13	CANTIM12	CANTIM11	CANTIM10	CANTIM9	CANTIM8	page 253
(0xE6)	CANTIML	CANTIM7	CANTIM6	CANTIM5	CANTIM4	CANTIM3	CANTIM2	CANTIM1	CANTIM0	page 253
(0xE5)	CANTCON	TPRSC7	TPRSC6	TPRSC5	TPRSC4	TPRSC3	TPRSC2	TRPSC1	TPRSC0	page 252
(0xE4)	CANBT3	–	PHS22	PHS21	PHS20	PHS12	PHS11	PHS10	SMP	page 252
(0xE3)	CANBT2	–	SJW1	SJW0	–	PRS2	PRS1	PRS0	–	page 251
(0xE2)	CANBT1	–	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0	–	page 251
(0xE1)	CANSIT1	–	SIT14	SIT13	SIT12	SIT11	SIT10	SIT9	SIT8	page 250
(0xE0)	CANSIT2	SIT7	SIT6	SIT5	SIT4	SIT3	SIT2	SIT1	SIT0	page 250
(0xDF)	CANIE1	–	IEMOB14	IEMOB13	IEMOB12	IEMOB11	IEMOB10	IEMOB9	IEMOB8	page 250
(0xDE)	CANIE2	IEMOB7	IEMOB6	IEMOB5	IEMOB4	IEMOB3	IEMOB2	IEMOB1	IEMOB0	page 250
(0xDD)	CANEN1	–	ENMOB14	ENMOB13	ENMOB12	ENMOB11	ENMOB10	ENMOB9	ENMOB8	page 250
(0xDC)	CANEN2	ENMOB7	ENMOB6	ENMOB5	ENMOB4	ENMOB3	ENMOB2	ENMOB1	ENMOB0	page 250
(0xDB)	CANGIE	ENIT	ENBOFF	ENRX	ENTX	ENERR	ENBX	ENERG	–	page 249
(0xDA)	CANGIT	CANIT	BOFFIT	OVRTIM	BXOK	SERG	CERG	FERG	AERG	page 247
(0xD9)	CANGSTA	–	OVRG	–	TXBSY	RXBSY	ENFG	BOFF	ERPP	page 247
(0xD8)	CANGCON	ABRQ	OVRQ	TTC	SYNTTC	LISTEN	TEST	ENA/STB	SWRES	page 246
(0xD7)	Reserved									

AVR CAN General registers (1)

CANBT1

-	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0	-

CANBT2

-	SJW1	SJW0	-	PRS2	PRS1	PRS0	-

CANBT3

-	PHS2-2	PHS2-1	PHS2-0	PHS1-2	PHS1-1	PHS1-0	SMP

BRP = prescaller. Tscl = (BRP+1)/CLK_{IO}_Freq

SJW = resynchronization Jump bit

PRS = propagation to compensate physical delay

PHS2 = phase error compensation after sampling point

PHS1 = phase error compensation before sampling point

SMP = sample type : 0 for 1 sample, 1 for 3 samples (1/2Tscl apart from center)

CANGCON

ABRQ	OVRQ	TTC	SYNCTTC	LISTEN	TEST	ENA/STB#	SWRES

General Control register

ABRQ : Abort request. An on going transmission or reception will be completed before the abort.

OVRQ : Overload frame request (see frame format 4 page for details)

TTC : Set to select the Time Trigger Communication mode

SYNCTTC : select Start of Frame or end of Frame for TTC synchronization

LISTEN : listening mode only when set

Test : factory reserved bit

ENA/STB# : Enable CAN standby when bit=0

SWRES : CAN Software Reset

CANGSTA

-	OVFG	-	TXBSY	RXBSY	ENFG	BOFF	ERRP

General Status register

OVFG : Overload Frame Flag (set while the overload frame is sent. No IT)

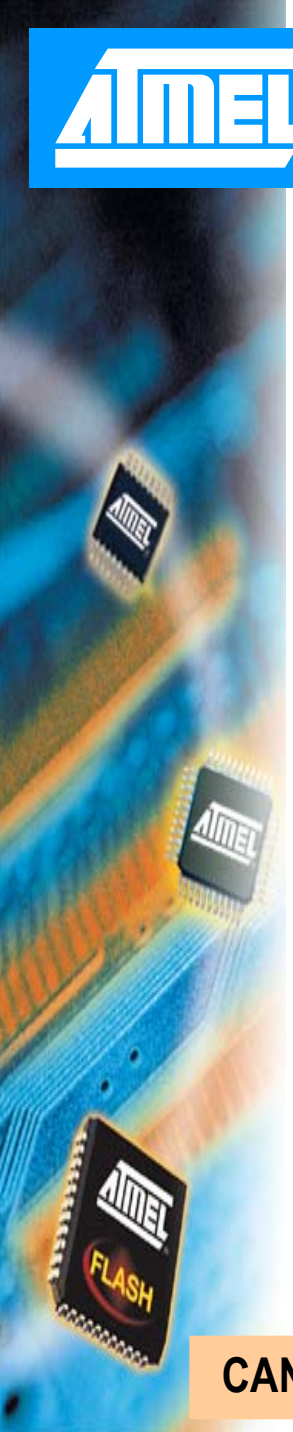
TXBSY : Transmitter Busy (Set while a transmission is in progress)

RXBSY : Receive Busy (set while a reception is in progress)

ENFG : Enable On Chip CAN controller Flag. Cleared after completion of on going transmit or receive after ENA has been cleared in CANGCON

BOFF : Bus Off indication

ERRP : Error Passive indication



CANGIT

CANIT	BOFFIT	OVRTIM	BXOK	SERG	CERG	FERG	AERG

General Interrupt

CANIT : CAN General IT status

BOFFIT : Bus Off IT

OVRTIM : Overrun CAN TIMER (roll over FFFF to 0000) can generate an INT if ETIM bit in IE1 is set

BXOK : Frame Buffer receive Interrupt

SERG : Stuffing error detected

CERG : CRC error detected (the faulty MOB will also get a CRC error in its CANSTMOB)

FERG : Form error

AERG : Acknowledge error general : A transmit message has not been acknowledged (ACK bit was red at 1)

CANGIE

ENIT	ENBOFF	ENRX	ENTX	ENERR	ENBX	ENERG	ENOVRT

General Interrupt enable register

ENIT : Enable all Interrupt except OVRTIM

ENBOFF : Enable Bus Off Interrupt

ENRX : Enable receive interrupt

ENTX : Enable Transmit Interrupt

ENERR : Enable message error (SERR, CERR, FERR, AERR) coming from any MOB (See ENERG below)

ENBX : Enable Frame Buffer Interrupt

ENERG : Enable general error

ENOVRT: Enable Timer Overrun Interrupt

CANEN1

-	ENMOB14	ENMOB13	ENMOB12	ENMOB11	ENMOB10	ENMOB9	ENMOB8

CANEN2

ENMOB7	ENMOB6	ENMOB5	ENMOB4	ENMOB3	ENMOB2	ENMOB1	ENMOB0

ENMOBi = 0 for unused MOB

ENMOBi = 1 for message enable (ready to send or ready to receive)

ENMOBi is set by rewriting the configuration in CANCDMOBi

CANSIT1

-	SIT14	SIT13	SIT12	SIT11	SIT10	SIT9	SIT8

CANSIT2

SIT7	SIT6	SIT5	SIT4	SIT3	SIT2	SIT1	SIT0

Status Interrupt message object

SITi = 0 No Interrupt for MOBi
SITi = 1 Interrupt request from MOBi

CANIE1

-	IEMOB14	IEMOB13	IEMOB12	IEMOB11	IEMOB10	IEMOB9	IEMOB8

CANIE2

IEMOB7	IEMOB6	IEMOB5	IEMOB4	IEMOB3	IEMOB2	IEMOB1	IEMOB0

Enable Interrupt message object

IEMOBi = Interrupt disable for MOBi

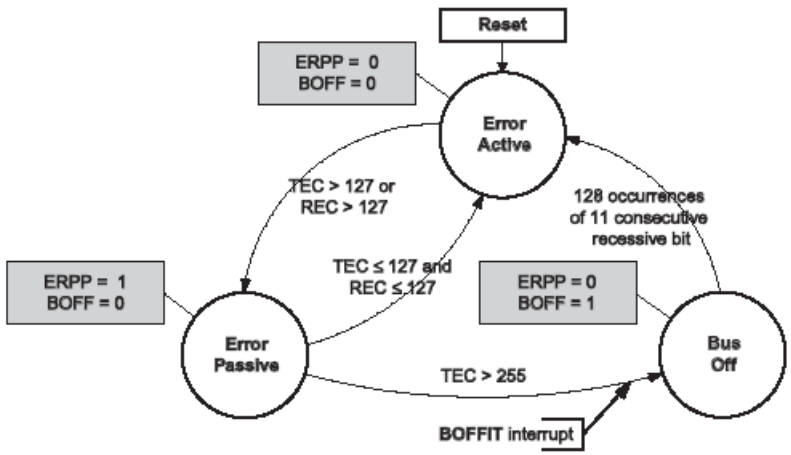
CANTEC

TEC7	TEC6	TEC5	TEC4	TEC3	TEC2	TEC1	TEC0

CANREC

REC7	REC6	REC5	REC4	REC3	REC2	REC1	REC0

CANTEC and CANREC : error counters



CAN MOB pointer and MOB priority pointer registers

CANPAGE

MOBNB3	MOBNB2	MOBNB1	MOBNB0	AINC#	INDX2	INDX1	INDX0

CANHPMOB

HPMOB3	HPMOBB2	HPMOB1	HPMOB0	CGP3	CGP2	CGP1	CGP0

MOBNB : Selection of the Message Object Buffer (MOB) : 0 to 14

AINC# : Auto-increment the index if AINC=0 . Successive access to CANMSG register will read or write the successive bytes (up to 8)

INDX : Byte location (0 to 7) in the data buffer array pointed by CANMSG.

HPMOB : Give the highest priority MOB with Interrupt request

CGP : General Purpose bit

Upon CAN Interrupt, CANHPMOB can be copied in CANPAGE

AVR CAN Message object register (1)

CANCDMOB

CONMOB1	CONMOB0	RPLV	IDE	DLC3	DLC2	DLC1	DLC0

CAN Message Object Configuration Register

CONMOB : 00 Disable
01 Enable Transmit
10 Enable Receive
11 Enable Receive Frame Buffer

RPLV : Automatic reply (0 to reply not ready, 1 to reply ready)

IDE : 0 for CAN2.0A (11 bit Identifier)
1 for CAN2.0B (29 bit identifier)

DLC : Data length code (0 to 8) indicate the number of valid Bytes expected from a received message. Give number of valid byte to transmit for a transmit message.

CANSTMOB

DLCW	TXOK	RXOK	BERR	SERR	CERR	FERR	AERR

Message Object status register

DLCW : Error number of receive byte is not equal to CANCDMOB DLC
TXOK : transmit OK
RXOK : receive OK
BERR : transmit bit error.
Recessive bit detected while a dominant bit was sent.
Or Dominant ack bit during an error frame transmission
SERR : Stuffing error
CERR : CRC error
FERR : Form error
AERR : Ack error : no detection of a dominant bit in the ack slot

Message object register IDT CAN2.0A (3)

CANIDT1

IDT10	IDT9	IDT8	IDT7	IDT6	IDT5	IDT4	IDT3

CANIDT2

IDT2	IDT1	IDT0	-	-	-	-	-

CANIDT3

-	-	-	-	-	-	-	-

CANIDT4

-	-	-	-	-	RTRTAG	-	RB0TAG

IDT = transmit Identifier or expected receive Identifier (see also mask)
RTRTAG : RTRbit of the Remote Data Frame to send
RB0TAG : RB0 bit of the of Remote Data Frame to send

Message object register IDT CAN2.0B (4)

CANIDT1

IDT28	IDT27	IDT26	IDT25	IDT24	IDT23	IDT22	IDT21

CANIDT2

IDT20	IDT19	IDT18	IDT17	IDT16	IDT15	IDT14	IDT13

CANIDT3

IDT12	IDT11	IDT10	IDT9	IDT8	IDT7	IDT6	IDT5

CANIDT4

IDT4	IDT3	IDT2	IDT1	IDT0	RTRTAG	RB1TAG	RB0TAG

IDT = transmit Identifier or expected receive Identifier (see also mask)
RTRTAG : RTRbit of the Remote Data Frame to send
RB1TAG RB0TAG : RB1 RB0 bit of the Remote Data Frame to send

Message object register IDM CAN2.0A (5)

CANIDM1

IDMSK10	IDMSK9	IDMSK8	IDMSK7	IDMSK6	IDMSK5	IDMSK4	IDMSK3

CANIDM2

IDMSK2	IDMSK1	IDMSK0	-	-	-	-	-

CANIDM3

-	-	-	-	-	-	-	-

CANIDM4

-	-	-	-	-	RTRMSK	-	IDEMSK

IDM = Receive Identifier mask(see also IDT)

RTMSK : remote transmission request mask value : 0 comparison true forced, 1 bit comparison enable

IDEMSK : Identifier extension mask value : 0 comparison true forced 1 comparison enable (detect CAN2.0b reception while CAN2.0a expected)

Message object register IDM CAN2.0B (6)

CANIDM1

IDMSK28	IDMSK27	IDMSK26	IDMSK25	IDMSK24	IDMSK23	IDMSK22	IDMSK21

CANIDM2

IDMSK20	IDMSK19	IDMSK18	IDMSK17	IDMSK16	IDMSK15	IDMSK14	IDMSK13

CANIDM3

IDMSK12	IDMSK11	IDMSK10	IDMSK9	IDMSK8	IDMSK7	IDMSK6	IDMSK5

CANIDM4

IDMSK4	IDMSK3	IDMSK2	IDMSK1	IDMSK0	RTRMSK	-	IDEMSK

IDM = Receive Identifier mask(see also IDT)

RTMSK : remote transmission request mask value : 0 comparison true forced, 1 bit comparison enable

IDEMSK : Identifier extension mask value : 0 comparison true forced 1 comparison enable (detect CAN2.0A reception while CAN2.0B expected)

CANMSG

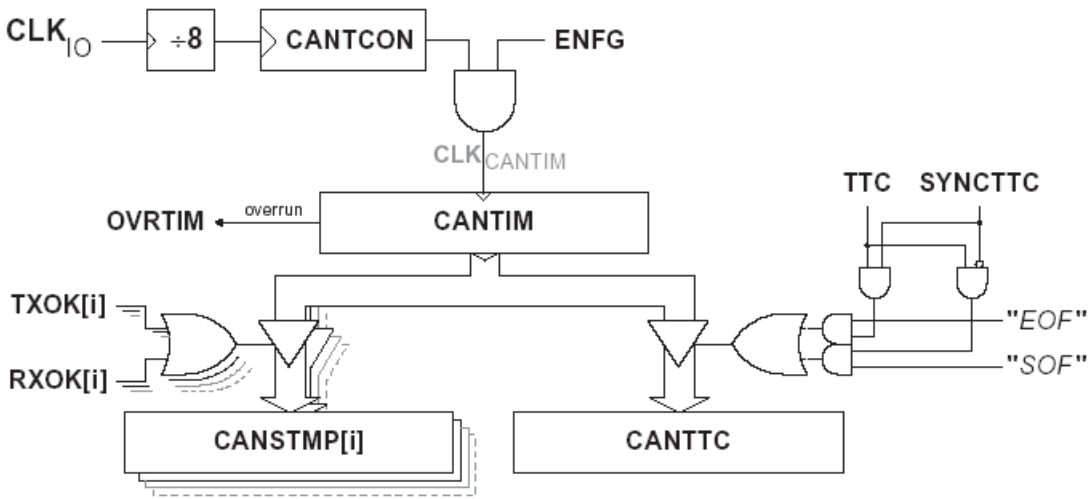
MSG7	MSG6	MSG5	MSG4	MSG3	MSG2	MSG1	MSG0

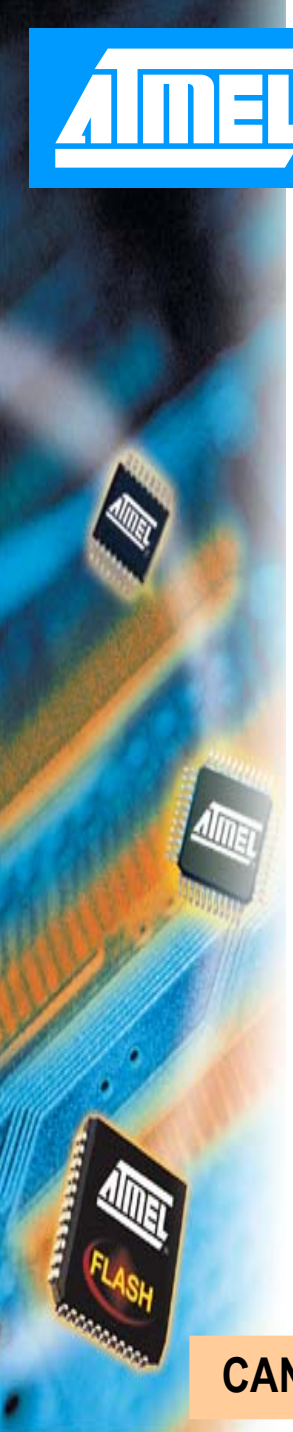
Can Message . If auto increment is programmed in CANPAGE, the index will be automatically incremented after each write or read into CANMSG (count = 0 to 7, then roll-over from 7 back to 0)

CANTCON

TPRESC7	TPRESC6	TPRESC5	TPRESC4	TPRESC3	TPRESC2	TPRESC1	TPRESC0

Prescaller for Timer Clock control (clock for CANTIMH/L)
The prescaller clock input is CLKio_Freq/8





CANTIMH

CANGTIM15	CANGTIM14	CANGTIM13	CANGTIM12	CANGTIM11	CANGTIM10	CANGTIM9	CANGTIM8

CANTIML

CANGTIM7	CANGTIM6	CANGTIM5	CANGTIM4	CANGTIM3	CANGTIM2	CANGTIM1	CANGTIM0

CAN general timer (16 bit) receives clock from the prescaller CANTCON

Timestamp register 1 per message object (3)

CANSTAMPH

TIMSTMP15	TIMSTMP14	TIMSTMP13	TIMSTMP12	TIMSTMP11	TIMSTMP10	TIMSTMP9	TIMSTMP8

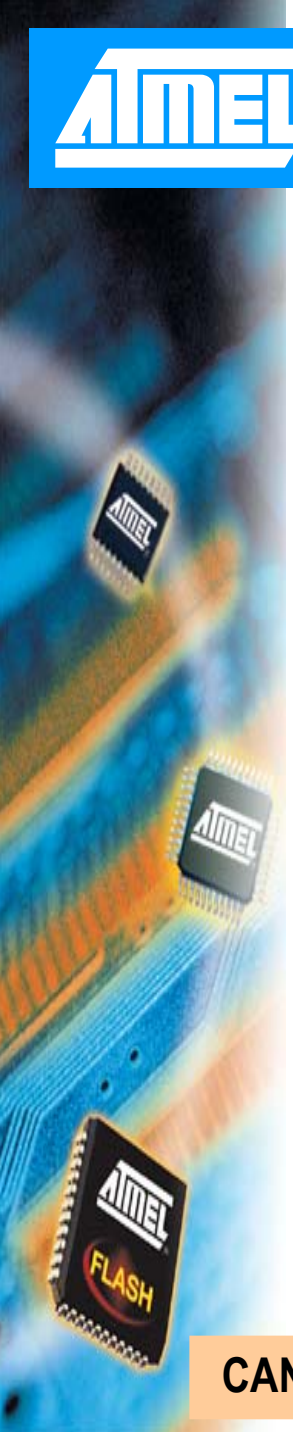
CANSTAMPL

TIMSTMP7	TIMSTMP6	TIMSTMP5	TIMSTMP4	TIMSTMP3	TIMSTMP2	TIMSTMP1	TIMSTMP0

CAN TIME STAMP (16 bit)

CANTIM value stored in CANSTAMP with TXOK or RXOK

One Time Stamp register per message object



CANTTCH

TIMTTC15	TIMTTC14	TIMTTC13	TIMTTC12	TIMTTC11	TIMTTC10	TIMTTC9	TIMTTC8

CANTTCL

TIMTTC7	TIMTTC6	TIMTTC5	TIMTTC4	TIMTTC3	TIMTTC2	TIMTTC1	TIMTTC0

CAN Time Trigger Communication TTC (16 bit)

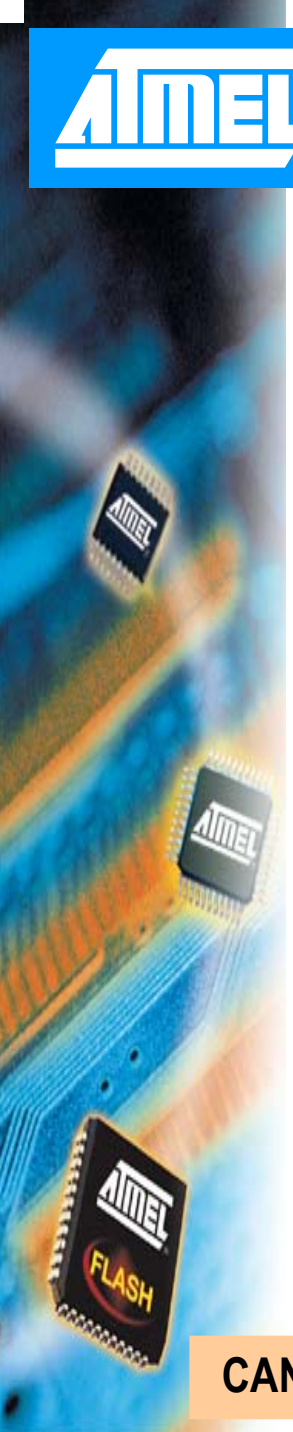
CANTIM value stored in CANTTC at start of Frame if SYNCTTC=1 or End Of Frame if SYNCTTC=0

Only one CANTTC register

How to start a CAN2.0B Transmission

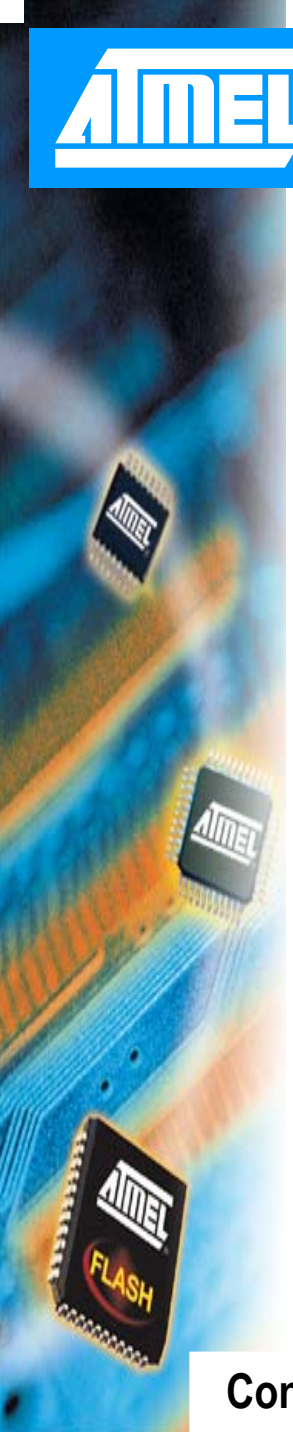
- **Reset CAN Controller** — **CANGCON = 0x01**
- **Disable CAN IT** — **CANGIE=0x00**
- **Initialize all Message Object (MOB = 0 to 14)**
  - CANPAGE = MOBnum << 4 (notice this will set INDx=0 and AutoINC)
  - Initialize CANCDMOB
- **Initialize BIT timings:** — **CANBT1, CANBT2, CANBT3**
- **Enable Interrupt :** — **CANGIE,CANIE2, CANIE1**
- **Enable CAN controller:** — **CANGCON = 0x02**
- **Configure one Message Object for TX**
  - Select CANPAGE
  - Set CANIDT1, CANIDT2, CANIDT3, CANIDT4
  - Set CANMSG with message content (Auto-increment)
  - Enable Message as Tx/6 Bytes/2.0B — CANCDMOB = 0x56

See the ATAVRCAN1m128 Software driver for source code. Above is only a commented example.



- Read CANGIT for possible errors
- Copy CANHPMOB into CANPAGE (will serve the highest priority MOB requesting attention see note below)
- Read CANSTMOB for RXOK or a possible error
- Read CANIDT[4:1] to load the receive ID
- Read DCLW in CANSTMOB to get the number of valid bytes in the CAN data field (1 to 8)
- For n=0 to n= DCLW : read CANMSG to read the Data message.
- Read CANSTMPH CANSTMPL (if time stamp is used)
- Rewrite CANCONCH CONCH[1:0] = 10 to re-enable the channel for a new reception

See the AT90CAN128 Software driver for source code. Above is only a commented example.



- CAN: The most used protocol in industrial & automotive applications
  - strong support by many HLPs & CAN tool vendors
- Atmel CAN unique feature in its range:
  - including an advanced powerful CAN Controller
  - In-System-Programming (ISP) of Program Flash via CAN bus
  - separated Flash memories for Program & Boot functions
- Atmel CAN: designed for industrial and automotive applications

Atmel CAN the ultimate CAN Controller