Atmel Microcontrollers
CAN Tutorial
Introduction

- 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
Why CAN?

- **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
What is CAN?

- 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
ISO-OSI* Reference Model

1. Physical Layer
2. Data Link Layer
3. Network Layer
4. Transport Layer
5. Session Layer
6. Presentation Layer
7. Application Layer

HLPs: CANopen, DeviceNet, OSEK/V**

Partially implemented by Higher Layer Protocols (HLP)

*) OSI - Open System Interconnection
**CAN Tutorial**

**CAN Bus Logic**

Two logic states on the CAN bus:

- "1" = recessive
- "0" = dominant

### CAN Protocol

<table>
<thead>
<tr>
<th>Node A</th>
<th>Node B</th>
<th>Node C</th>
<th>BUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>R</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>R</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>R</td>
<td>R</td>
<td>D</td>
</tr>
<tr>
<td>R</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>R</td>
<td>D</td>
<td>R</td>
<td>D</td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>

"Wired-AND" function:
- As soon as one node transmits 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.
Typical CAN Node

µController

CAN Controller

Diff. CAN Line Driver

CAN Bus
(terminated by 120 Ohm on each side)
CAN Tutorial

CAN Bus (up to 40m @1Mb/s, up to 1km @50Kb/s)

Device #1

Device #2

Device #3

..............

Device #n

Dominant

Recessive

TXd

CANh

CANl

RXd

CAN Controller

CAN Transceiver
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

Data Stream

$ = \text{Staff Bits}$

CAN Bus Bit Stream
• Hard synchronization at Start Of Frame bit

- Intermission / Idle
- SOF
- ID10 ID9 ID8 ID7 ID6 ID5

All nodes synchronize on leading edge of SOF bit (Hard Synchronization)

• Re-Synchronization on each Recessive to Dominant bit

- Re-synch
- Re-synch
- Re-synch

CAN Protocol
• Length of one time quanta can be set to multiple of \( \mu \)Controller 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

Bit Rate [kbps] vs. CAN Bus Length [m]

Example based on CAN Bus Lines by twisted pair
Bus Frame
Duration in Data Bit

CAN - V2.0A
Duration in Data Bit

CAN - V2.0B
Duration in Data Bit

SOF  Start of Frame
CRC  Cyclic Redundancy Code
del  Delimiter
ACK  Acknowledge

EOF  End of Frame
IFS  Inter Frame Spacing
RTR  Remote Transmission Request
SRR  Substitute Remote Request
RB0/1  Reserved bits
DLC  Data Length Code

ARBITRATION  DATA
CTRL

Bit Stuffing

≥3

Frame Formats (1)
### CAN Protocol

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

Duration in Data Bit

Remote Frame

Duration in Data Bit

Bit Stuffing

Data Frame:
- **SOF**
- **ARBIT.**
- **CTRL**
- **DATA**
- **CRC**
- **del**
- **ACK**
- **del**
- **EOF**
- **IFS**

Remote Frame:
- **SOF**
- **ARBIT.**
- **CTRL**
- **CRC**
- **del**
- **ACK**
- **del**
- **EOF**
- **IFS**

Bit Stuffing:

- (\*\*) RTR = dominant
- (\*\*) RTR = recessive

Duration in Data Bit:

- 1
- 12-32
- 6
- 0...64
- 15
- 1
- 1
- 1
- 7
- ≥3
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
Three fundamental states define each node’s error signaling:
- Error active: Normal state, node can send all frames including 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 deadlocks by faulty nodes.

Fault Confinement (1)
• 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
Higher level protocols

- Why HLPs
- CANOpen
- DeviceNet
- CAN Kingdom
- OSEK/VDX
- SDS
- J1939
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 > 8 bytes
- 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 inter-operability of different devices
  - High speed real-time capability
- **Features**
  - Created by Allen-Bradley (Rockwell Automation 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 automation: 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
CAN Tutorial
CAN Kingdom

- CAN Kingdom is more than 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
• 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
Smart Distributed System (SDS)

• Features
  - Created by Honeywell
  - Close to DeviveNet, CAL & CANopen
CAN: a Large Field of Applications

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

- Heating Control
- Air Conditioning (AC)
- Security (fire, burglar...)
- Access Control
- Light Control
• Washing machines
• Dishes cleaner
• Self-service bottle distributors connected to internet
CAN Tutorial
Automotive & Transportation

- 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
- 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 Tutorial
CAN Controller: Mailbox concept (1)

Channel Nr. & Data Offset

Channel Status
Channel Control & DLC

Message Data (1)
ID Tag (4)
ID Masks (4)
TimStmp (2)

Ch. 14 - TimStmp (2)
Ch. 0 - TimStmp (2)
Ch. 14 - ID Tag (4)
Ch. 0 - ID Tag (4)
Ch. 14 - ID Masks (4)
Ch. 0 - ID Masks (4)
Ch. 14 - Data Buffer (8)
Ch. 0 - Data Buffer (8)
Ch. 14 - Control & DLC
Ch. 0 - Control & DLC

15 Channels

CAN family
- 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
• 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
• 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)
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 Tutorial
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 Tutorial

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)
• 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
• 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
• 15 Message objects (Channels), each with filtering, masking and FIFO buffer
• All Channel features programmable on-the-fly
- 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
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
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 Microcontrollers: T89C51CC01 Block Diagram

User Flash Memory
32k x 8

Boot Flash Memory
2K x 8

EEPROM
2K x 8

C51 X2 Core

ADC
10bit / 8 Channels

PCA
5 Channels

CAN Controller
15 Channels

Emulation Support Logic

RAM
1.2K x 8

Interrupt Unit

Prog. Watchdog Timer

Timers 0 / 1 / 2

Port 0

Port 1

Port 2

Port 3

Port 4

8 I/O

8 I/O

8 I/O

8 I/O

8 I/O

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)
CAN Tutorial

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 (X2 mode)
- Temperature: -40 to 85°C
- Voltage: 3 to 5 Volt +/-10%
- Packages: PLCC44, TQFP44, CA-BGA64
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 Tutorial

CAN Microcontrollers: T89C51CC02 Block Diagram

- User Flash Memory: 16k x 8
- Boot Flash Memory: 2K x 8
- EEPROM: 2K x 8
- C51 X2 Core
- ADC: 10bit / 8 Channels
- PCA: 2 Channels
- CAN Controller: 4 Channels
- RAM: 0.5K x 8
- Interrupt Unit
- Prog. Watchdog Timer
- Timers 0/1/2
- Port 1
- Port 2
- Port 3
- Port 4
- Emulation Support Logic
- Test Support Logic

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 Tutorial

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
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
• 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
### AT90CAN128 in CAN Family

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

**NEW**

AT90CAN128 in CAN Family

**AT90CAN128 in CAN Family**

<table>
<thead>
<tr>
<th>Feature</th>
<th>T89C51CC01</th>
<th>T89C51CC02</th>
<th>AT89C51CC03</th>
<th>AT90CAN128</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>8051 Architecture</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIPS</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>16</td>
</tr>
<tr>
<td>Flash program/boot</td>
<td>32KB/2KB</td>
<td>16KB/2KB</td>
<td>64KB/2KB</td>
<td>128KB/ up to 8KB</td>
</tr>
<tr>
<td>EEPROM</td>
<td>2KB</td>
<td>2KB</td>
<td>2KB</td>
<td>4KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1.2KB</td>
<td>0.5KB</td>
<td>2.2KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Power Fail Detect</td>
<td>-</td>
<td>-</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>CAN Controller</td>
<td>15 Message Objects</td>
<td>4 Message Objects</td>
<td>15 Message Objects</td>
<td>15 Message Objects</td>
</tr>
<tr>
<td>SPI</td>
<td>-</td>
<td>-</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Supply (V)</td>
<td>3 to 5.5</td>
<td>3 to 5.5</td>
<td>3 to 5.5</td>
<td>2.7 to 5.5</td>
</tr>
<tr>
<td>ADC</td>
<td>10 bit / 8 channels</td>
<td>10 bit / 8 channels</td>
<td>10 bit / 8 channels</td>
<td>10 bit / 8 channels</td>
</tr>
<tr>
<td>PCA</td>
<td>5 channels</td>
<td>2 channels</td>
<td>5 channels</td>
<td>-</td>
</tr>
<tr>
<td>Timers 8bit</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Timers 0 / 2</td>
</tr>
<tr>
<td>Timers 16bit</td>
<td>Timers 0 / 1 / 2</td>
<td>Timers 0 / 1 / 2</td>
<td>Timers 0 / 1 / 2</td>
<td>Timer 1 / 3</td>
</tr>
<tr>
<td>UART (Hardware)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Port</td>
<td>Port 0 / 1 / 2 / 3</td>
<td>Port 1 / 2 / 3</td>
<td>Port 0 / 1 / 2 / 3</td>
<td>Port A/B/C/D/E/F/G</td>
</tr>
<tr>
<td>Bootloader</td>
<td>UART / CAN (DeviceNet – CANopen)</td>
<td>UART / CAN</td>
<td>Same UART / CAN as T89C51CC01</td>
<td>Hard : SPI, JTAG Soft : UART, CAN</td>
</tr>
<tr>
<td>Packages</td>
<td>TQFP44, PLCC44, CA-BGA64</td>
<td>SOIC24, SOIC28, TQFP32, PLCC28</td>
<td>TQFP44, PLCC44, BGA8*8</td>
<td>TQFP64, QFN64, BGA64</td>
</tr>
</tbody>
</table>
CAN REGISTERS
C51 base Core
### CANBT1

<table>
<thead>
<tr>
<th></th>
<th>BRP5</th>
<th>BRP4</th>
<th>BRP3</th>
<th>BRP2</th>
<th>BRP1</th>
<th>BRP0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### CANBT2

<table>
<thead>
<tr>
<th></th>
<th>SJW1</th>
<th>SJW0</th>
<th></th>
<th>PRS2</th>
<th>PRS1</th>
<th>PRS0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### CANBT3

<table>
<thead>
<tr>
<th></th>
<th>PHS2-2</th>
<th>PHS2-1</th>
<th>PHS2-0</th>
<th>PHS1-2</th>
<th>PHS1-1</th>
<th>PHS1-0</th>
<th>SMP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

BRP = prescaler. \( T_{scl} = \frac{(BRP+1)}{FCAN} \)
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 (1/2\( T_{scl} \) apart from center)
C51 CAN General Registers (2)

**CANGCON**

<table>
<thead>
<tr>
<th>ABRQ</th>
<th>OVRQ</th>
<th>TTC</th>
<th>SYNC TTC</th>
<th>AUTOBAUD</th>
<th>TEST</th>
<th>ENA</th>
<th>GRES</th>
</tr>
</thead>
</table>

General Control register

- **ABRQ**: Abort request. An ongoing 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
- **SYNC TTC**: 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

<table>
<thead>
<tr>
<th></th>
<th>OVFG</th>
<th></th>
<th>TBSY</th>
<th>RBSY</th>
<th>ENFG</th>
<th>BOFF</th>
<th>ERRP</th>
</tr>
</thead>
</table>

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
### General Interrupt

<table>
<thead>
<tr>
<th>CANIT</th>
<th>-</th>
<th>OVRTIM</th>
<th>OVRBUF</th>
<th>SERG</th>
<th>CERG</th>
<th>FERG</th>
<th>AERG</th>
</tr>
</thead>
</table>

**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

<table>
<thead>
<tr>
<th>ENRX</th>
<th>ENTX</th>
<th>ENERCH</th>
<th>ENBUF</th>
<th>ENERG</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>ENRX</td>
<td>ENTX</td>
<td>ENERCH</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th></th>
<th>ENCH14</th>
<th>ENCH13</th>
<th>ENCH12</th>
<th>ENCH11</th>
<th>ENCH10</th>
<th>ENCH9</th>
<th>ENCH8</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANEN1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### CANEN2

<table>
<thead>
<tr>
<th>ENCH7</th>
<th>ENCH6</th>
<th>ENCH5</th>
<th>ENCH4</th>
<th>ENCH3</th>
<th>ENCH2</th>
<th>ENCH1</th>
<th>ENCH0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANEN2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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**
Status Interrupt message object

SITi = 0 No Interrupt for channel i
SITi = 1 Interrupt request from channel i
Enable Interrupt message object

IECHi = Interrupt disable for channel i
SITi = 1 Interrupt enable for channel i
### CAN Tutorial
**C51 CAN General Register (10)**

#### CANTEC

<table>
<thead>
<tr>
<th>TEC7</th>
<th>TEC6</th>
<th>TEC5</th>
<th>TEC4</th>
<th>TEC3</th>
<th>TEC2</th>
<th>TEC1</th>
<th>TEC0</th>
</tr>
</thead>
</table>

#### CANREC

<table>
<thead>
<tr>
<th>REC7</th>
<th>REC6</th>
<th>REC5</th>
<th>REC4</th>
<th>REC3</th>
<th>REC2</th>
<th>REC1</th>
<th>REC0</th>
</tr>
</thead>
</table>

**CANTEC and CANREC: error counters**

- **Init.**
- **TEC**: Transmit Error Counter
- **REC**: Receive Error Counter

**Diagram:**
- **Error Active**
  - TEC > 127
  - REC > 127
- **Error Passive**
  - TEC < 127
  - REC < 127
- **Bus Off**
  - TEC = 255

**States:**
- 128 occurrences of consecutive recessive bit
- 11 consecutive recessive bit

---

CAN Registers

CAN Tutorial  16/03/2004  75
**CAN Message Object pointer register**

**CANPAGE**

<table>
<thead>
<tr>
<th>CHNB3</th>
<th>CHNB2</th>
<th>CHNB1</th>
<th>CHNB0</th>
<th>AINC</th>
<th>INDX2</th>
<th>INDX1</th>
<th>INDX0</th>
</tr>
</thead>
</table>

**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.
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.
CAN Tutorial
C51 CAN Message object register (2)

CANSTCH

<table>
<thead>
<tr>
<th>DLCW</th>
<th>TXOK</th>
<th>RXOK</th>
<th>BERR</th>
<th>SERR</th>
<th>CERR</th>
<th>FERR</th>
<th>AERR</th>
</tr>
</thead>
</table>

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
IDT = transmit Identifier or expected receive Identifier (see also mask)
RTRTAG : Remote transmit request tag
RB0TAG : reserved bit
### 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 : Remote transmit request tag
RB1TAG RB0TAG : reserved bit
### CAN Registers

#### CANIDM1

<table>
<thead>
<tr>
<th>IDMSK10</th>
<th>IDMSK9</th>
<th>IDMSK8</th>
<th>IDMSK7</th>
<th>IDMSK6</th>
<th>IDMSK5</th>
<th>IDMSK4</th>
<th>IDMSK3</th>
</tr>
</thead>
</table>

#### CANIDM2

<table>
<thead>
<tr>
<th>IDMSK2</th>
<th>IDMSK1</th>
<th>IDMSK0</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
</tr>
</thead>
</table>

#### CANIDM3

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
</tr>
</tbody>
</table>

#### CANIDM4

<table>
<thead>
<tr>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>RTRMSK</th>
<th>IDEMSK</th>
</tr>
</thead>
</table>

**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**

<table>
<thead>
<tr>
<th>IDMSK28</th>
<th>IDMSK27</th>
<th>IDMSK26</th>
<th>IDMSK25</th>
<th>IDMSK24</th>
<th>IDMSK23</th>
<th>IDMSK22</th>
<th>IDMSK21</th>
</tr>
</thead>
</table>

**CANIDM2**

<table>
<thead>
<tr>
<th>IDMSK20</th>
<th>IDMSK19</th>
<th>IDMSK18</th>
<th>IDMSK17</th>
<th>IDMSK16</th>
<th>IDMSK15</th>
<th>IDMSK14</th>
<th>IDMSK13</th>
</tr>
</thead>
</table>

**CANIDM3**

<table>
<thead>
<tr>
<th>IDMSK12</th>
<th>IDMSK11</th>
<th>IDMSK10</th>
<th>IDMSK9</th>
<th>IDMSK8</th>
<th>IDMSK7</th>
<th>IDMSK6</th>
<th>IDMSK5</th>
</tr>
</thead>
</table>

**CANIDM4**

<table>
<thead>
<tr>
<th>IDMSK4</th>
<th>IDMSK3</th>
<th>IDMSK2</th>
<th>IDMSK1</th>
<th>IDMSK0</th>
<th>RTRMSK</th>
<th>-</th>
<th>IDEMSK</th>
</tr>
</thead>
</table>

**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)
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)

<table>
<thead>
<tr>
<th>MSG7</th>
<th>MSG6</th>
<th>MSG5</th>
<th>MSG4</th>
<th>MSG3</th>
<th>MSG2</th>
<th>MSG1</th>
<th>MSG0</th>
</tr>
</thead>
</table>

**CANMSG**
### CANTCON

<table>
<thead>
<tr>
<th>TPESC7</th>
<th>TPESC6</th>
<th>TPESC5</th>
<th>TPESC4</th>
<th>TPESC3</th>
<th>TPESC2</th>
<th>TPESC1</th>
<th>TPESC0</th>
</tr>
</thead>
</table>

Prescaler for Timer Clock control (clock for CANTIMH/L)
The prescaler clock input is Fcan/6
### CAN Tutorial

### C51 CAN timer register (2)

**CANTIMH**

<table>
<thead>
<tr>
<th>CANGTIM15</th>
<th>CANGTIM14</th>
<th>CANGTIM13</th>
<th>CANGTIM12</th>
<th>CANGTIM11</th>
<th>CANGTIM10</th>
<th>CANGTIM9</th>
<th>CANGTIM8</th>
</tr>
</thead>
</table>

**CANTIML**

<table>
<thead>
<tr>
<th>CANGTIM7</th>
<th>CANGTIM6</th>
<th>CANGTIM5</th>
<th>CANGTIM4</th>
<th>CANGTIM3</th>
<th>CANGTIM2</th>
<th>CANGTIM1</th>
<th>CANGTIM0</th>
</tr>
</thead>
</table>

**CAN general timer (16 bit) receives clock from the prescaler CANTCON**
**CAN Tutorial**

**Timestamp register 1 per message object (3)**

**CAN STAMP**

**CANSTAMPH**

<table>
<thead>
<tr>
<th>TIMSTMP15</th>
<th>TIMSTMP14</th>
<th>TIMSTMP13</th>
<th>TIMSTMP12</th>
<th>TIMSTMP11</th>
<th>TIMSTMP10</th>
<th>TIMSTMP9</th>
<th>TIMSTMP8</th>
</tr>
</thead>
</table>

**CANSTAMPL**

<table>
<thead>
<tr>
<th>TIMSTMP7</th>
<th>TIMSTMP6</th>
<th>TIMSTMP5</th>
<th>TIMSTMP4</th>
<th>TIMSTMP3</th>
<th>TIMSTMP2</th>
<th>TIMSTMP1</th>
<th>TIMSTMP0</th>
</tr>
</thead>
</table>

**CAN TIME STAMP (16 bit)**

CANTIM value stored in CANSTAMP with TXOK or RXOK

One TimeStamp register per message object
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
Detail remote frame with automatic reply

**CAN Registers**
CAN Tutorial
C51 CAN Detail remote frame

Message object in transmission by CAN controller:

Message object in reception by CAN controller:

Message object in reception by user:

Message object in transmission by user:

Message object stay in reception:

Message object stay in transmission:

Remote frame:

Data frame (deferred):

Modified by user: \( m \)

Modified by CAN: \( c \)
CAN Tutorial

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

CAN Registers
• 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
<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x01</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x02</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x08</td>
<td>CANMSG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x09</td>
<td>CANSTIMH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0A</td>
<td>CANSTM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0B</td>
<td>CANID0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0C</td>
<td>CANID1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0D</td>
<td>CANID2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0E</td>
<td>CANID3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0F</td>
<td>CANID4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x10</td>
<td>CANDM0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x11</td>
<td>CANDM1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x12</td>
<td>CANDM2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x13</td>
<td>CANDM3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x14</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** The table above represents the AVR CAN Registers map. Each register is assigned a specific address and bit configuration.
<table>
<thead>
<tr>
<th>CANBT1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
</tr>
<tr>
<td>CANBT2</td>
</tr>
<tr>
<td>-</td>
</tr>
<tr>
<td>CANBT3</td>
</tr>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

BRP = prescaler.  \(T_{cl} = (BRP+1)/CLK_{IO\_Freq}\)
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 (1/2\(T_{cl}\) apart from center)
**CANGCON**

<table>
<thead>
<tr>
<th>ABRQ</th>
<th>OVRQ</th>
<th>TTC</th>
<th>SYNCTTC</th>
<th>LISTEN</th>
<th>TEST</th>
<th>ENA/STB#</th>
<th>SWRES</th>
</tr>
</thead>
</table>

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
### 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

<table>
<thead>
<tr>
<th></th>
<th>OVFG</th>
<th></th>
<th>TXBSY</th>
<th>RXBSY</th>
<th>ENFG</th>
<th>BOFF</th>
<th>ERRP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### CANIT

<table>
<thead>
<tr>
<th>CANIT</th>
<th>BOFFIT</th>
<th>OVRTIM</th>
<th>BXOK</th>
<th>SERG</th>
<th>CERG</th>
<th>FERG</th>
<th>AERG</th>
</tr>
</thead>
</table>

**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)
### General Interrupt enable register

<table>
<thead>
<tr>
<th>ENIT</th>
<th>ENBOFF</th>
<th>ENRX</th>
<th>ENTX</th>
<th>ENERR</th>
<th>ENBX</th>
<th>ENERG</th>
<th>ENOVRT</th>
</tr>
</thead>
</table>

- **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
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
**CAN Tutorial**

**AVR CAN General register (8)**

<table>
<thead>
<tr>
<th>SIT14</th>
<th>SIT13</th>
<th>SIT12</th>
<th>SIT11</th>
<th>SIT10</th>
<th>SIT9</th>
<th>SIT8</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN</td>
<td>SIT1</td>
<td>CANSIT1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SIT7</th>
<th>SIT6</th>
<th>SIT5</th>
<th>SIT4</th>
<th>SIT3</th>
<th>SIT2</th>
<th>SIT1</th>
<th>SIT0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN</td>
<td>SIT2</td>
<td>CANSIT2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Status Interrupt message object**

SITi = 0 No Interrupt for MOBi  
SITi = 1 Interrupt request from MOBi
Enable Interrupt message object

$\text{IEMOBi} = \text{Interrupt disable for MOBi}$
### CAN Registers

#### CANTEC

<table>
<thead>
<tr>
<th>TEC7</th>
<th>TEC6</th>
<th>TEC5</th>
<th>TEC4</th>
<th>TEC3</th>
<th>TEC2</th>
<th>TEC1</th>
<th>TEC0</th>
</tr>
</thead>
</table>

#### CANREC

<table>
<thead>
<tr>
<th>REC7</th>
<th>REC6</th>
<th>REC5</th>
<th>REC4</th>
<th>REC3</th>
<th>REC2</th>
<th>REC1</th>
<th>REC0</th>
</tr>
</thead>
</table>

**CANTEC and CANREC** and **error counters**

![Diagram showing error counters and states]

- **Error Active**: ERPP = 0, BOFF = 0
- **Error Passive**: TEC > 127 or REC > 127, TEC < 127 and REC < 127, ERPP = 0, BOFF = 1
- **Bus Off**: TEC > 255
- **Reset**: ERPP = 1, BOFF = 0

**128 occurrences of 11 consecutive recessive bit**
**CAN Page**

<table>
<thead>
<tr>
<th>MOBNB3</th>
<th>MOBNB2</th>
<th>MOBNB1</th>
<th>MOBNB0</th>
<th>AINC#</th>
<th>INDEX2</th>
<th>INDEX1</th>
<th>INDEX0</th>
</tr>
</thead>
</table>

**CANHPMOB**

<table>
<thead>
<tr>
<th>HPMOB3</th>
<th>HPMOB2</th>
<th>HPMOB1</th>
<th>HPMOB0</th>
<th>CGP3</th>
<th>CGP2</th>
<th>CGP1</th>
<th>CGP0</th>
</tr>
</thead>
</table>

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
### 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

<table>
<thead>
<tr>
<th>DLCW</th>
<th>TXOK</th>
<th>RXOK</th>
<th>BERR</th>
<th>SERR</th>
<th>CERR</th>
<th>FERR</th>
<th>AERR</th>
</tr>
</thead>
</table>

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
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
### CAN Registers

**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
## CANIDM1

<table>
<thead>
<tr>
<th>IDMSK10</th>
<th>IDMSK9</th>
<th>IDMSK8</th>
<th>IDMSK7</th>
<th>IDMSK6</th>
<th>IDMSK5</th>
<th>IDMSK4</th>
<th>IDMSK3</th>
</tr>
</thead>
</table>

## CANIDM2

<table>
<thead>
<tr>
<th>IDMSK2</th>
<th>IDMSK1</th>
<th>IDMSK0</th>
<th>---</th>
<th>---</th>
<th>---</th>
<th>---</th>
<th>---</th>
</tr>
</thead>
</table>

## CANIDM3

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
</tr>
</tbody>
</table>

## CANIDM4

<table>
<thead>
<tr>
<th>---</th>
<th>---</th>
<th>---</th>
<th>---</th>
<th>---</th>
<th>RTRMSK</th>
<th>---</th>
<th>IDEMSK</th>
</tr>
</thead>
</table>

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)
### CAN Registers

#### CANIDM1

<table>
<thead>
<tr>
<th>IDMSK28</th>
<th>IDMSK27</th>
<th>IDMSK26</th>
<th>IDMSK25</th>
<th>IDMSK24</th>
<th>IDMSK23</th>
<th>IDMSK22</th>
<th>IDMSK21</th>
</tr>
</thead>
</table>

#### CANIDM2

<table>
<thead>
<tr>
<th>IDMSK20</th>
<th>IDMSK19</th>
<th>IDMSK18</th>
<th>IDMSK17</th>
<th>IDMSK16</th>
<th>IDMSK15</th>
<th>IDMSK14</th>
<th>IDMSK13</th>
</tr>
</thead>
</table>

#### CANIDM3

<table>
<thead>
<tr>
<th>IDMSK12</th>
<th>IDMSK11</th>
<th>IDMSK10</th>
<th>IDMSK9</th>
<th>IDMSK8</th>
<th>IDMSK7</th>
<th>IDMSK6</th>
<th>IDMSK5</th>
</tr>
</thead>
</table>

#### CANIDM4

<table>
<thead>
<tr>
<th>IDMSK4</th>
<th>IDMSK3</th>
<th>IDMSK2</th>
<th>IDMSK1</th>
<th>IDMSK0</th>
<th>RTRMSK</th>
<th>-</th>
<th>IDEMSK</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th>MSG7</th>
<th>MSG6</th>
<th>MSG5</th>
<th>MSG4</th>
<th>MSG3</th>
<th>MSG2</th>
<th>MSG1</th>
<th>MSG0</th>
</tr>
</thead>
</table>

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)
Prescaler for Timer Clock control (clock for CANTIMH/L)
The prescaler clock input is CLKio_Freq/8
CAN Registers

### CAN Tutorial

#### CAN timer register (2) AVR CAN

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

<table>
<thead>
<tr>
<th>CANGTIM15</th>
<th>CANGTIM14</th>
<th>CANGTIM13</th>
<th>CANGTIM12</th>
<th>CANGTIM11</th>
<th>CANGTIM10</th>
<th>CANGTIM9</th>
<th>CANGTIM8</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANTIMH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CANGTIM7</th>
<th>CANGTIM6</th>
<th>CANGTIM5</th>
<th>CANGTIM4</th>
<th>CANGTIM3</th>
<th>CANGTIM2</th>
<th>CANGTIM1</th>
<th>CANGTIM0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANTIML</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CAN TIME STAMP (16 bit)
CANTIM value stored in CANSTAMP with TXOK or RXOK
One Time Stamp register per message object

<table>
<thead>
<tr>
<th>CANSTAMPH</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMSTMP15</td>
<td>TIMSTMP14</td>
</tr>
<tr>
<td>TIMSTMP13</td>
<td>TIMSTMP12</td>
</tr>
<tr>
<td>TIMSTMP11</td>
<td>TIMSTMP10</td>
</tr>
<tr>
<td>TIMSTMP9</td>
<td>TIMSTMP8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CANSTAMPL</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMSTMP7</td>
<td>TIMSTMP6</td>
</tr>
<tr>
<td>TIMSTMP5</td>
<td>TIMSTMP4</td>
</tr>
<tr>
<td>TIMSTMP3</td>
<td>TIMSTMP2</td>
</tr>
<tr>
<td>TIMSTMP1</td>
<td>TIMSTMP0</td>
</tr>
</tbody>
</table>
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.
How to serve a CAN Reception Interrupt

- 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**