Open up a CAN with 56800/E Hybrid Controllers
Goals

- Overview of Controller Area Network (CAN) communication protocol
- Identify popular application areas
- Introduce 56800/E hardware and software support
- Demonstrates the ease of developing CAN applications using CodeWarrior™ development tools with Processor Expert™ technology.
CAN Overview
Overview

- CAN Spec 2.0 developed by BOSCH GmbH
- Serial communications protocol for inter-processor communication
- Originally targeted Automotive to reduce the growing complexity of the wiring harness in modern car design.
- Applicable to other cost-sensitive and environmentally-demanding applications in the industrial sector.
- Low cost of CAN networks is realized by high performance 56800E with on-chip CAN modules
What is CAN?

• Controller Area Network

✓ Bit-oriented Serial Communications Protocol
✓ Variable bit rate: 5 Kbit/s up to 1 Mbit/s
✓ Peer-to-Peer: any node may transmit at any frame
✓ Multi-cast without routing: all nodes receive all messages
✓ CSMA/CR†: non-destructive bit-wise arbitration
✓ Prioritization of messages via the identifier
✓ Fault confinement
✓ Automatic retransmission of corrupted messages
✓ System-wide data consistency
✓ High level of error detection (< 10^-10)

† (Collision sensing multiple access with collision detection)
Requirements of a CAN Controller

- **Simple user interface to CPU**
  - Access control & status registers
  - Access to buffers
  - Interrupt and error types

- **Message filtering & buffering**
  - Store incoming & outgoing messages
  - Only interrupt CPU with relevant messages
  - Predictable Message Transmission

- **Protocol handling**
  - Error Detection
  - Arbitration detection
  - Bit monitoring/stuffing

- **Physical layer interface**
  - Current & voltage control for bus
  - Absorb transients
  - Signal bus (line) faults & correct

**CAN bus**
- **Physical interface**
  - CANH
  - CANL

**Phased Connectivity Summit**

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. © Freescale Semiconductor, Inc. 2004
## CAN Physical Interfaces - Automotive Standards

<table>
<thead>
<tr>
<th></th>
<th>High Speed Differential</th>
<th>Fault Tolerant Differential</th>
<th>Single Wire</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of Bus Wires</strong></td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Ground Reference)</td>
</tr>
<tr>
<td><strong>Maximum Bus Speed</strong></td>
<td><strong>500 kbps</strong></td>
<td><strong>125 kbps</strong></td>
<td><strong>33.33 kbps</strong></td>
</tr>
<tr>
<td></td>
<td>(Ltd by prop delay)</td>
<td>(Ltd by prop delay)</td>
<td></td>
</tr>
<tr>
<td><strong>Bus Topologies</strong></td>
<td>Linear</td>
<td>Bus, Star, Ring</td>
<td>Bus, Star, Ring</td>
</tr>
<tr>
<td><strong>Automotive Standards</strong></td>
<td>ISO 11898</td>
<td>ISO - To Be Determined</td>
<td>SAE J2411</td>
</tr>
<tr>
<td><strong>Documents</strong></td>
<td>SAE J2284</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Parts Available</strong></td>
<td>MC33989 (SBC), PCA82C250</td>
<td>MC3388, MC33889 (SBC Lite),</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MC33889 (SBC), PCA82C252</td>
<td></td>
</tr>
<tr>
<td><strong>Comments</strong></td>
<td>Silicon solutions to common mode noise issues under investigation</td>
<td>Automatically switch to single wire reception when fault detected</td>
<td>Based on J1850-VPW technology with enhanced wakeup capability</td>
</tr>
</tbody>
</table>
CAN-C (J2284)

- Linear bus topology
- 500kbps bit rate
- 40m max bus length
- Maximum of 16 nodes

CAN-C (J2284) diagram:

- CANH and CANL lines with voltage levels
  - 3.5V
  - 2.5V
  - 1.5V
- Twisted-pair media
- Stubs "1m"
- Terminators can be inside end ECUs
- ECU 1, ECU 2, ECU n
- R_T 120Ω
**CAN: Message Transfer**

- Information sent in fixed format *message frames*.
- Any node may start to transmit when bus is free (Bus Idle).
- If two or more nodes start transmitting in same *frame*, bus access conflict is resolved by *bit-wise arbitration*.
- **Highest priority** message wins bus access.
- **Arbitration Field** (message ID) determines the message priority.
- Transmitting nodes which lose arbitration become receivers and automatically re-transmit at next available time.
- No data or time wasted, someone always wins.
- **ID - Labels message contents** (no physical node addresses).
CAN: Message Transfer

- All nodes check consistency of messages received and will flag an inconsistent message to the entire network.

- All receiving nodes **acknowledge** a valid message.

- A message is received correctly by all nodes or no nodes.

- All nodes apply **Message Filtering** to decide whether to accept a message.

- Any number of nodes can simultaneously receive and accept a message (Multi-cast transmission).
CAN: Message Types

- Messages are one of four different types, called frames:
  - **Data Frame**: transmits up to 8 bytes of data
  - **Remote Transmission Request (RTR) Frame**: requests a Data Frame
  - **Error Frame**: indicates a bus error (independent of CPU)
  - **Overload Frame**: creates an extra delay between Data Frames or Remote Frames

- Only **Data** and **RTR** Frames can be transmitted under host control.
CAN Message Arbitration

- Applies to Data Frames and RTR Frames.
- Priority based on Message ID: lowest value = highest priority
- Message Identifiers must be unique, assigned during system design.
- When the bus is free, any node may start to transmit a message.
- If 2 or more nodes start to transmit at the same time, the bus access conflict is resolved by bit-wise arbitration using the Arbitration Field.

During transmission of the Arbitration Field, ALL transmitters compare:

Value of the bit transmitted (TX) AND Value of the bit monitored (received - RX) on the bus
CAN Message Arbitration

- Wired-OR mechanism:
  
  A *dominant* bit will ALWAYS overwrite a *recessive* bit.

- If RX = TX (dominant or recessive), the node may continue to transmit.

- If RX = dominant but TX = recessive bit:
  - Node has *lost the arbitration*
  - Node must *immediately stop transmitting*
  - Node continues to receive

- NO time or information is lost

- REQUIRES that all transmitters are synchronized
Arbitration Field

Contains the Message ID, which has three functions:

1. Defines the priority of the message.
   The message with the highest priority arbitration field wins access to the CAN bus and may continue to transmit the rest of the message. This requires that each message in a system is defined with a unique Identifier.

2. Labels the message.
   - Each message must have a unique Identifier
   - The ID may be used to label the message contents. For example, the message with Identifier 0x123 always contains the latest value from sensor A.

3. Filters messages.
   - Programmable hardware filter determines message acceptance
   - Saves processor time by eliminating the processing of unwanted messages.
   - To achieve efficient filters on all nodes, select Identifiers carefully. Filtering allows any number of nodes to receive and simultaneously act upon the same message, providing multicast communication.
Vehicles and Transportation

- 80% of an annual 100-million-unit market with perhaps 20 distinct applications.

- CAN is the in-vehicle network (IVN)
  - engine management
  - body electronics (e.g. door and roof control)
  - air conditioning
  - lightning
  - entertainment control

- Majority of the European carmakers use CAN-based IVNs. American and Far East manufacturers started implementing CAN-based IVNs.
Other Segments

The 20% of the market shared by all the other segments combined, however, represents thousands of applications most of which do not reach high volume

Factory automation: Control of assembly line manufacturing machinery enables automation. Typical applications include conveyors, production data recording, and other end-user configurable systems.

Medical: Hospitals control vital operating room components such as OR lights and tables, endoscope lights and cameras, insufflators, X-ray and ultrasound machines, video recorders, and video printers

Aviation: CAN is used as a backbone network in aircrafts for flight state sensors, navigation systems and research PCs driving displays installed in the cockpit.
Freescale CAN Products
Freescale CAN Hardware Solutions

- **Optimized for Event-Driven Network Applications**
  - MSCAN08
    - 3TX & 2RX
    - 4 Filter
  - MSCAN12
    - 3TX & 2RX
    - 8 Filter
  - msCAN12 (on s12)
    - 3TX & 5 RX
    - 8 Filter

- **Optimized for Time-Driven Network Applications**
  - FlexCAN
    - up to 16 Mbps
    - TouCAN Filter
  - FlexCAN
    - up to 64 Mbps
    - TouCAN Filter
  - TouCAN
    - 16 Message Buffers
    - 1 Global + 2 Special Filters

- **Performance**
  - HC08
  - HCS08
  - HC12
  - 56800
  - HCS12
  - HC16
  - 683xx
  - 56800E
  - PPC

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.
© Freescale Semiconductor, Inc. 2004
56800/E Hardware & Software
56800E Hybrid Controller Roadmap

**56850 Series**
- Telecom/voice, RAM-based, 120 MMACS, 81–144 pins

**56F8400 Series**
- Automotive, industrial, Flash-based, 120 MMACS, Flex-Ray, 2xCan, Improved PWM & ADC, low power, 48–160 pins

**56F8500 Series**
- Automotive, industrial, Flash-based, 150 MMACS, Flex-Ray, 2xCan, Improved PWM & ADC, low power, 48–160 pins

**56F8300 Series**
- Automotive & Industrial, Flash-based, 60 MMACS, 16–512KB PFlash, 48–160 pins

**56F8100 Family**
- Mid-Range Industrial and General Purpose, Flash Based, 40MMACs 48–144 pins

**56F8000 Series**
- Low end industrial and automotive control, Flash-based, 32 MMACS, 28–32 pins

**56F820 Series**
- General Purpose, Flash-based, 40 MMACs, 100–128 pins

**56F80x Series**
- Industrial controllers, Flash-based, 40 MMACs, 32–160 pins

**56F80**
- 0.25µ, 56800E 30/40 MMACS
- 0.25µ, 56800E 32 MMACS

**56F830**
- 0.25µ, 56800E 40 MMACS
- 0.18µ, 56800E 120 MMACS

**56F8400**
- 0.1µ, 568400 120 MMACS

**56F8500**
- 0.1µ, 568500 150 MMACS

**56F8355**
- 0.18µ, 568355 120 MMACS

**56F8356**
- 0.18µ, 568356 120 MMACS

**56F8346**
- 0.18µ, 568346 120 MMACS

**56F8345**
- 0.18µ, 568345 120 MMACS

**56F8323**
- 0.18µ, 568323 120 MMACS

**56F8322**
- 0.18µ, 568322 120 MMACS

**56F8313**
- 0.18µ, 568313 120 MMACS

**56F8312**
- 0.18µ, 568312 120 MMACS

**56F810**
- 0.25µ, 56810 30/40 MMACS

**56F811**
- 0.25µ, 56811 30/40 MMACS

**56F812**
- 0.25µ, 56812 30/40 MMACS

**56F813**
- 0.25µ, 56813 30/40 MMACS

**56F814**
- 0.25µ, 56814 30/40 MMACS

**56F815**
- 0.25µ, 56815 30/40 MMACS

**56F816**
- 0.25µ, 56816 30/40 MMACS
56F80x MSCAN Features

- Version 2.0B compliant
- Standard and extended data frames
- 0-8 bytes data length
- Programmable bit rate up to 1 Mbps
- Support for remote frames
- Double-buffered receive storage scheme
- Triple-buffered transmit storage scheme
- Flexible maskable identifier filter
- Programmable wake-up functionality with integrated low-pass filter
- Separate signaling and interrupt capabilities for all CAN RX/TX error states
- Three low power modes
- Based on the Motorola Scalable Controller Area Network (MSCAN12) definition as implemented on the MC68HC12
56F83xx FlexCAN Features

- Version 2.0 B compliant
- Standard and extended data frames
- 0-8 bytes data length
- Programmable bit rate up to 1Mbps
- Support for remote frames
- Double-buffered receive storage scheme
- Flexible maskable identifier filter
- Programmable wake-up functionality
- Separate signaling and interrupt capabilities for all CAN RX/TX error states
- Three low power modes
- Programmable first transmit scheme: Lowest ID or Lowest Message Buffer
- “Time Stamp”, based on 16-bit free-running timer with Global Network Synchronization
- Sixteen Flexible Message Buffers of 0-8 bytes Data Length, each configurable as RX or TX, all support Standard and Extended Messages
FlexCAN Block Diagram

- 16 Configurable Message Buffers
- Dedicated Peripheral RAM memory mapped as Register I/O
- Configurable Max MB to reduce matching process overhead
- Control Block performs matching process
- Bus Interface Unit provides 56800E Core data bus interface
- CAN TX/RX Serial Message Buffers (SMB) interface with External Transceiver required for connection to physical CAN bus
Typical Can System

- Two-wire differential physical interface

- Transceiver Provides
  - Transmit Drive
  - Wave Shaping
  - Receive/Compare Functions
  - Protection from defective CAN bus

Figure 7-2. Typical CAN system
**Message Buffer Structure**

- **Extended ID**
  - 29-bit Message ID (over 522K more IDs)
  - 8-bit Time Stamp (MSB of timer)
  - IDE - indicates Extended ID

<table>
<thead>
<tr>
<th>15-8</th>
<th>7-4</th>
<th>3-0</th>
<th>CONTROL/STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0</td>
<td>TIME_STAMP</td>
<td>CODE</td>
<td>LENGTH</td>
</tr>
<tr>
<td>$2</td>
<td>ID[14-0]</td>
<td>RTR</td>
<td>IDE_LOW</td>
</tr>
<tr>
<td>$3</td>
<td>DATA BYTE 0</td>
<td>DATA BYTE 1</td>
<td></td>
</tr>
<tr>
<td>$4</td>
<td>DATA BYTE 2</td>
<td>DATA BYTE 3</td>
<td></td>
</tr>
<tr>
<td>$5</td>
<td>DATA BYTE 4</td>
<td>DATA BYTE 5</td>
<td></td>
</tr>
<tr>
<td>$6</td>
<td>DATA BYTE 6</td>
<td>DATA BYTE 7</td>
<td></td>
</tr>
<tr>
<td>$7</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7-3. Extended ID Message Buffer Structure

- **Standard ID**
  - 11-bit Message ID
  - 16-bit Time Stamp

<table>
<thead>
<tr>
<th>15-8</th>
<th>7-4</th>
<th>3-0</th>
<th>CONTROL/STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0</td>
<td>TIME_STAMP</td>
<td>CODE</td>
<td>LENGTH</td>
</tr>
<tr>
<td>$1</td>
<td>ID[28:18]</td>
<td>RTR</td>
<td>0</td>
</tr>
<tr>
<td>$2</td>
<td>16-BIT TIME STAMP</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$3</td>
<td>DATA BYTE 0</td>
<td>DATA BYTE 1</td>
<td></td>
</tr>
<tr>
<td>$4</td>
<td>DATA BYTE 2</td>
<td>DATA BYTE 3</td>
<td></td>
</tr>
<tr>
<td>$5</td>
<td>DATA BYTE 4</td>
<td>DATA BYTE 5</td>
<td></td>
</tr>
<tr>
<td>$6</td>
<td>DATA BYTE 6</td>
<td>DATA BYTE 7</td>
<td></td>
</tr>
<tr>
<td>$7</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7-4. Standard ID Message Buffer Structure
TX Process

- Software – Configure MB for TX
  - Write Control/Status word to Deactivate TX MB (Code = 1000)
  - Write ID_High and ID_Low words
  - Write Data bytes
  - Write Control/Status word to Activate TX MB (active Code, Length)

Table 7-3. Message Buffer Codes for Transmit Buffers

<table>
<thead>
<tr>
<th>RTR</th>
<th>Initial TX Code</th>
<th>Description</th>
<th>Code After Successful Transmission</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>1000</td>
<td>Message buffer not ready for transmit</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1100</td>
<td>Data Frame to be transmitted once, unconditionally</td>
<td>1000</td>
</tr>
<tr>
<td>1</td>
<td>1100</td>
<td>Remote Frame to be transmitted once, and message buffer becomes an RX message buffer for Data Frames</td>
<td>0100</td>
</tr>
<tr>
<td>0</td>
<td>1010(^1)</td>
<td>Data Frame to be transmitted only as a response to a Remote Frame</td>
<td>1010</td>
</tr>
<tr>
<td>0</td>
<td>1110</td>
<td>Data Frame to be transmitted only once, unconditionally, and then only as a response to Remote Frame</td>
<td>1010</td>
</tr>
</tbody>
</table>

1. When a matching remote request frame is detected, the code for such a message buffer is changed to be 1110.
TX Process

- **Hardware**
  - Internal Arbitration selects the next TX MB based lowest ID or lowest Buffer number (configurable)

- **On Successful TX**
  - Value of Free-Running timer copied to Time Stamp field of MB
  - Code in Control/Status word updated
  - Flag Register (FCIFLAG1) bit set
RX Process

- **Software – Configure MB for RX**
  - Write Control/Status word to Deactivate MB (Code=0000)
  - Write ID_High and ID_Low words to set Acceptance Code
  - Write Control/Status word to Activate MB (Code=0100)

![Table 7-2: Message Buffer Codes for Receive Buffers](image-url)
RX Process

- Hardware
  - Acceptance Filtering ensures that only messages required by the application are transferred from Serial MB.

App Note: Mbps with identical IDs (acceptance codes) do NOT behave like a FIFO. The Lowest MB matching ID will get an overflow.

- On Successful RX
  - Value of Free-Running timer copied to Time Stamp field of MB
  - ID, Data, Length fields stored
  - Code in Control/Status word updated (Full, Overrun)
  - Flag Register (FCIFLAG1) bit set
## RX Process

**Acceptance Filter Exercise:** Match the received message ID to the Message Buffer.

<table>
<thead>
<tr>
<th>FlexCAN Configuration</th>
<th>Mask Register</th>
<th>Base ID ID28…..ID18</th>
<th>IDE</th>
<th>Extended ID ID17…..ID0</th>
<th>Matching Msg Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Mask</td>
<td>11111111110</td>
<td>-</td>
<td>111111100000000000001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RX Buffer 14</td>
<td>01111111111</td>
<td>-</td>
<td>111111100000000000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MB2 ID</td>
<td>11111111000</td>
<td>0</td>
<td>----------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MB3 ID</td>
<td>11111111000</td>
<td>1</td>
<td>01010101010101010101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MB14 ID</td>
<td>11111111000</td>
<td>1</td>
<td>01010101010101010101</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Received Msg ID</th>
<th>SMB 11111111001</th>
<th>1</th>
<th>01010101010101010101</th>
<th>MB3</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMB 11111111001</td>
<td>0</td>
<td>-----------</td>
<td>MB2</td>
<td></td>
</tr>
<tr>
<td>SMB 11111111001</td>
<td>1</td>
<td>01010101010101010100</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>SMB 01111111000</td>
<td>0</td>
<td>-----------</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>SMB 01111111000</td>
<td>1</td>
<td>01010101010101010101</td>
<td>MB14</td>
<td></td>
</tr>
<tr>
<td>SMB 10111111000</td>
<td>1</td>
<td>01010101010101010101</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
RX Process

- **Software - Read a receive frame from its MB:**
  - Read Control/Status word (mandatory—activates internal lock for this buffer)
  - Read ID (Optional - essential only if a mask was used)
  - Read Data field word(s)
  - Release internal lock by one of the following ways:
    - Read Free-Running Timer (Optional- releases internal lock).
    - Read Control/Status word of another MB
    - If not executed, the MB remains locked.

- **App Note:** Keep in mind that displaying MB Control/Status words in Debugger will activate internal lock and may cause errant behavior.
The Complete Development Environment

**CodeWarrior for 56800/E**

CodeWarrior™ for Motorola 56800/E is a window based visual IDE that includes an optimizing C compiler, assembler and linker, project management system, editor and code navigation system, debugger, simulator, scripting, source control, and third party plug in interface.

**Processor Expert™**

Processor Expert (PE) provides a Rapid Application Design (RAD) tool that combines easy-to-use component-based software application creation with an expert knowledge system. PE is fully integrated with the CodeWarrior for 56800/E.

**Hardware Tools**

The 56800/E solutions are supported with a complete set of evaluation modules which supply all required items for rapid evaluation and software and hardware development. In addition several command converter options exist for customer target system debugger connection.
Processor Expert Overview

**Processor Expert™**
- Supports rapid application development
- Enables component oriented programming
- Provides expert advice if necessary
- Delivers instant functionality of generated code
- Provides tested ready-to-use code

**Key Abstraction Technologies**

- **PESL**
  - Processor Expert System Library
  - Peripheral oriented
- **EB – an abstraction provider**
  - Embedded Beans
  - Functionality oriented
  - Real components for building of an application

**How Features of PE are Achieved**
- Developed by experienced programmers of embedded systems
- Expert knowledge system is working on the background of PE and checks all the settings
- Provides context help and access to CPU/MCU vendor documentation
- All EB delivered by UNIS are tested according to ISO testing procedures (UNIS is ISO certified company)
Processor Expert Features

- Available across 8/16-bit product lines
- Rapid application development
- Expert configuration system
- Instant functionality of generated code
- Two Peripheral programming levels
  - Embedded Beans
  - PESL
- Application Specific Algorithm Libraries
  - All SDK algorithm libraries ported
- Tested and ready-to-use code
# Application Specific Algorithm Libraries

## Memory Manager
- Dynamic allocation

## Feature Phone Library
- CallerID type 1&2, CallerID Parser, Generic Echo Cancellor

## DSP Library
- FIR, IIR, FFT, Auto Correlation, Bit Reversal

## Telephony Libraries
- AEC, AGC, Caller ID,
- CAS, CPT, CTG, DTMF
- G165, G168, G711
- G723, G726, G729

## Modem Libraries
- V.8bis, V.21, V.22bis, V.42bis

## Security Libraries
- RSA, DES, 3DES,

## Motor Control
- BLDC, ACIM, SR motor specific algorithms
- General purpose algorithms

## Math Libraries
- Matrix, Fractional, Vector
- Trigonometric

## Feature Phone Library
- CallerID type 1&2,
- CallerID Parser,
- Generic Echo Cancellor

## Security Libraries
- RSA, DES, 3DES,

## Motor Control
- BLDC, ACIM, SR motor specific algorithms
- General purpose algorithms

## Math Libraries
- Matrix, Fractional, Vector
- Trigonometric

## Tools Library
- Cycle Count, FIFO, FileIO, Test
Developing Applications
Task Description

Develop a “Chat Room” application that uses FlexCAN acceptance filters to implement Broadcast and Private communication channels and Blocking.
System Block Diagram

Node 0
- Serial Port (SCI)

Node 1
- Serial Port (SCI)

Node 2
- Serial Port (SCI)

CAN Bus
Approach

- Design Message IDs and Filtering Scheme to support Broadcast and Private communications as well as Blocking
- Use Processor Expert Beans to implement application
  - Freescale CAN
  - PC Master (SCI)
- Download and Execute on 56F8357 EVM
Design Message IDs and Filtering Scheme

Use Standard Format messages (ID28-ID18)

Messages can be sent in one of two formats:
- Broadcast – can be received by all Nodes
- Private – can only be received by a specific Node

Message reception can be blocked using CAN acceptance filtering.

CAN Chat PC Application variables
- PrivateNode – identifies the destination ID
  - 0x00 – Broadcast to all Nodes
  - 0x01 – Node 0
  - 0x02 – Node 1
  - 0x04 – Node 2
- BlockNodes – identifies Node(s) to disregard if message is received
  - 0x01 – Node 0
  - 0x02 – Node 1
  - 0x04 – Node 2
  - Values OR’d together to represent blocked nodes (e.g. 0x5 = Nodes 0 & 2)
Design Message IDs

Transmit Message IDs (Standard Format)

<table>
<thead>
<tr>
<th>Msg Destination</th>
<th>Msg Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>001 – to Node 0</td>
<td>001 – from Node 0</td>
</tr>
<tr>
<td>010 – to Node 1</td>
<td>010 – from Node 1</td>
</tr>
<tr>
<td>100 – to Node 2</td>
<td>100 – from Node 2</td>
</tr>
<tr>
<td>000 – Broadcast</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID28</th>
<th>ID27</th>
<th>ID26</th>
<th>ID25</th>
<th>ID24</th>
<th>ID23</th>
<th>ID22</th>
<th>ID21</th>
<th>ID20</th>
<th>ID19</th>
<th>ID18</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Examples:**
- Node 0 broadcasting a message: \[0000000001\]
- Node 1 private message to Node 2: \[0100000010\]
- Node 2 broadcasting a message: \[0000000100\]

**Note:** *Msg Source also used to specify window display number.*
Design Message Filtering Scheme

- Accept Broadcast messages

✓ Use Global Mask Register (GMR) to mask **Destination** fields
✓ Set Message Buffer 0 ID destination field to **Broadcast** (000)

Example: Accept messages broadcast to all.

<table>
<thead>
<tr>
<th>Dest</th>
<th>Src</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMR: 1 1 1 0 0 0 0 0 0 0 0 0</td>
<td>MB0 ID: 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

Broadcast messages are sent with 000 in the destination field!
Design Message Filtering Scheme

- Accept Private messages (source = Node ID)

- Use Global Mask Register (GMR) to mask Destination fields
- Set Message Buffer 1 ID destination field to Node ID

Example: Accept messages specifically destined for Node 2 (ID=100)

<table>
<thead>
<tr>
<th>Dest</th>
<th>Src</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMR:</td>
<td>1 1 1 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>MB0 ID:</td>
<td>1 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

Private messages are sent with NodeID in the destination field!

- NodeID 0: 001
- NodeID 1: 010
- NodeID 2: 100
Design Message Filtering Scheme

- **Blocking**

  - Use Global Mask Register (GMR) to mask Source fields
  - Set Message Buffer IDs source fields to (000)

**Example:** Block messages received from Node 2 (ID=100)

<table>
<thead>
<tr>
<th>Dest</th>
<th>Src</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMR:</td>
<td>1 1 1 0 0 0 0 0 1 0 0</td>
</tr>
<tr>
<td>MB0 ID:</td>
<td>0 1 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

Messages are sent with NodeID in the Src field!

- NodeID 0  001
- NodeID 1  010
- NodeID 2  100
Use PE to implement Application
Steps:

- Open Processor Expert
  Project named: CanChat.mcp

- Add FreescaleCAN Bean
  with following ISR configurations: Setting
  preserve registers to Yes will save the entire register
  context prior to entering the ISR enabling function calls
  within the ISR.
Steps:

- Add Message Buffers by clicking +
- Configure MB ID for Broadcast messages
- Configure MB ID for Private messages:
  - 100 for Node0
  - 200 for Node1
  - 400 for Node2
- Configures Global Mask Register
**Steps:**

- Set bit rate to 125 kbps

<table>
<thead>
<tr>
<th>Timing</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN timing wizard</td>
<td>click to run timing wizard -&gt;</td>
</tr>
<tr>
<td>Propagation segment</td>
<td>0</td>
</tr>
<tr>
<td>Time segment 1</td>
<td>7</td>
</tr>
<tr>
<td>Time segment 2</td>
<td>3</td>
</tr>
<tr>
<td>RSJ</td>
<td>1</td>
</tr>
<tr>
<td>Samples per bit</td>
<td>One sample</td>
</tr>
<tr>
<td>Time quanta per bit</td>
<td>14</td>
</tr>
<tr>
<td>Bit rate</td>
<td>125 kbit/s</td>
</tr>
</tbody>
</table>
Steps:

- Enable Code Generation
Steps: Generate Code
Steps: Open source file CANChat.c

- Configure NodeID: This determines the Source ID for transmission and corresponding MB1 ID should be configured for Private message receipt.

```c
/* Message ID Defines */
#define NODE0 0x001
#define NODE1 0x002
#define NODE2 0x004

/* PC Master Global Variables */
byte NodeID = NODE0;
```
Steps: Drag-n-drop methods (CANChat.c)

- Initialize Mask and IDs for CANChat display

```c
void main(void)
{
    byte txErr;
    byte PrevFilter = 0;
    byte PrevID = 0;

    /* Get Initialization MB IDs */
    GlobalMask_ID = 0x1FFFFFFF & CAN1_GetAcceptanceMask();
    CAN1_GetAcceptanceCode(MB_RX_BROADCAST, &MsgBuffer0_ID);
    CAN1_GetAcceptanceCode(MB_RX_PRIVATE, &MsgBuffer1_ID);
}
```
Steps: Drag-n-drop methods (CANChat.c)

```c
for(;;)
{
    if (NodeID != PrevID)
    {
        CAN1_SetAcceptanceCode(MB_RX_PRIVATE, (NodeID<<8));
        CAN1_GetAcceptanceCode(MB_RX_PRIVATE, &MsgBuffer1_ID);
        PrevID = NodeID;
    }

    /* Verify if PCMaster Sent a new byte */
    if (KEYPRESS)
    {
        txErr = CAN1_SendFrame(CAN_MB1_MASK, (PrivateNode*PRIVATE_SHIFT)|NodeID, DATA_FRAME, sizeof(Transmit_Ba...
    }
}
```
Steps: Drag-n-drop methods (Events.c)

```c
#pragma interrupt called
void CAN1_OnFullRxBuffer(word BufferMask)
{
    dword ID;
    byte type, len, format;
    byte rxBuff[8];
    byte rxErr=ERR_OK;
    byte rxBuffSelect;
    byte rxTemp;

    /* Get state of reception ... */
    rxBuffSelect = (byte) CAN1_GetStateRX();

    /* Read received data */
    rxErr= CAN1_ReadFrame(--rxBuffSelect, &ID, &type,
        &format, &len, rxBuff);

    /* Get NODE ID information from the Message ID */
    rxTemp = (byte)ID;

    Received_Bytes[rxTemp>>1]=rxBuff[0];
    LED1_On();
    TI1_Enable();
}
```
Build, Download, and Execute
Launch PC Master Application

- Double Click CANChat.pmp to launch
- Click Logo to activate and/or clear displays
- Type message in OutBox
Summary
Summary

- Understand Controller Area Network (CAN) basics
- Exposed to application areas outside of Automotive
- Introduced 56800/E hardware and software support
- Demonstrated the ease of developing CAN applications using CodeWarrior development tools with Processor Expert™ technology.
Thank You!