Open up a CAN with 56800/E Hybrid Controllers

Embedded Connectivity Summit 2004
October 4,5,6

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

Embedded Connectivity Summit

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

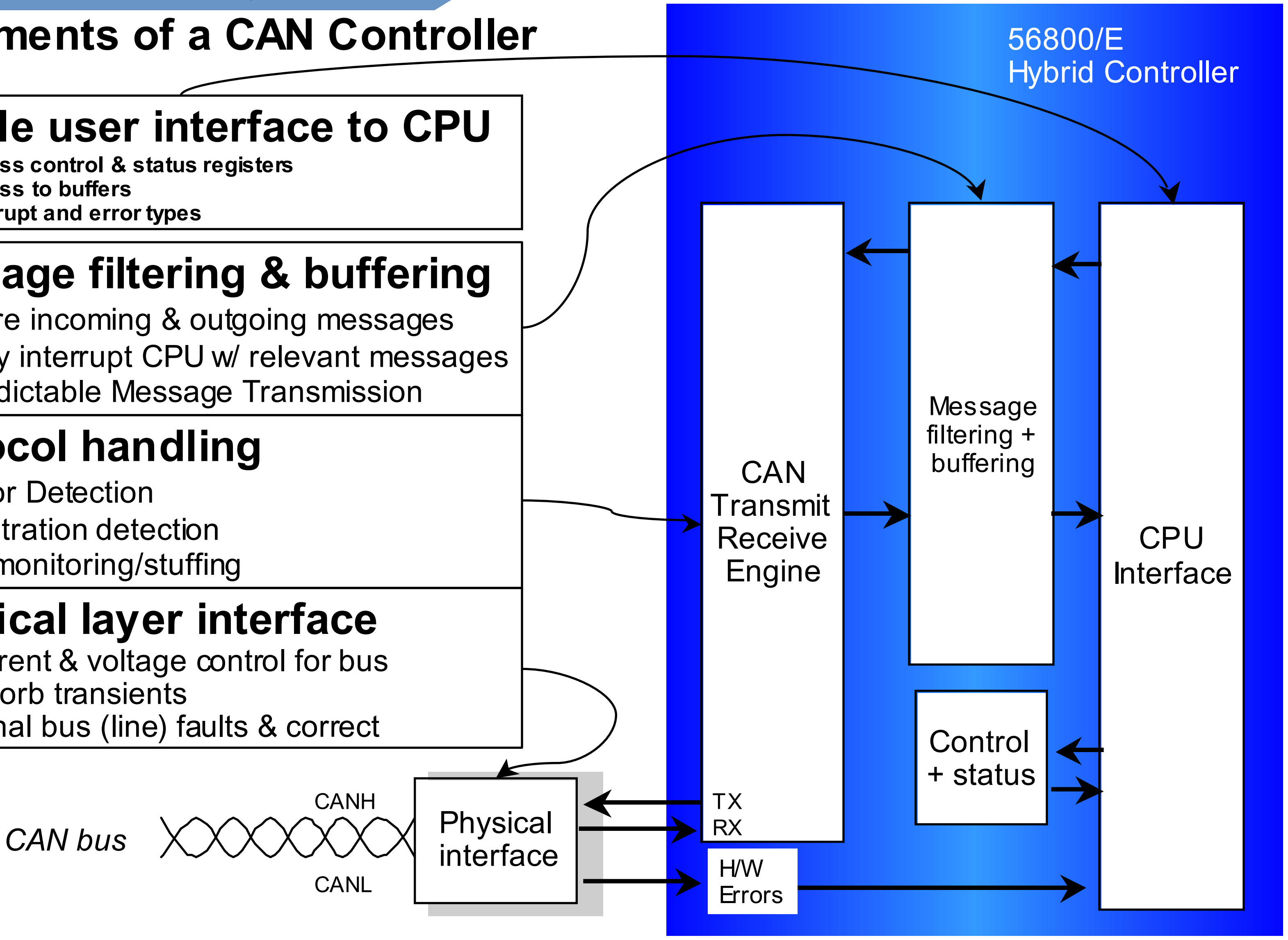
- ✓ Bit-oriented Serial Communications Protocol
- ✓ Variable bit rate: 5 Kbit/s up to 1 Mbit/s
- ✓ Peer-to-Peer: *any* node may transmit at *any* frame
- ✓ Multi-cast without routing: all nodes receive all messages
- ✓ CSMA/CR†: non-destructive bit-wise arbitration
- ✓ Prioritization of messages via 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)

Launched by Motorola freescale semiconductor

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 w/ 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

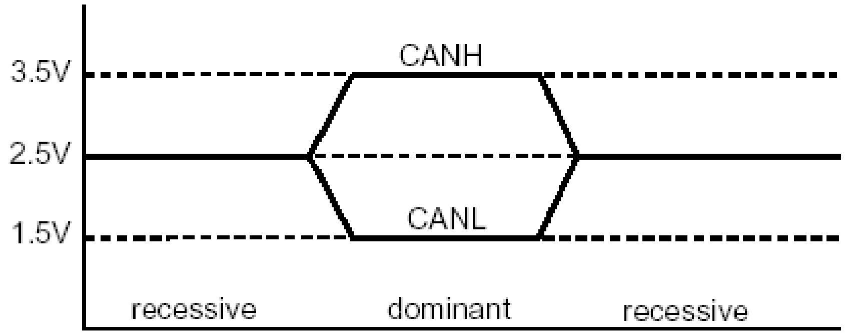


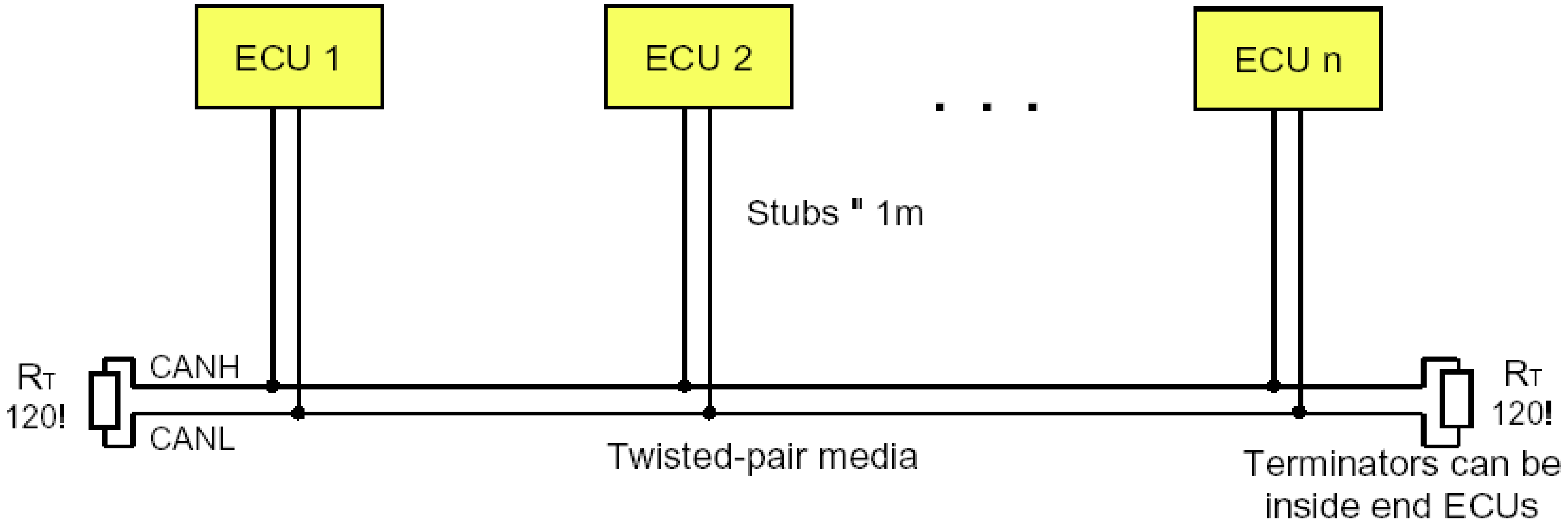
CAN Physical Interfaces - Automotive Standards

	High Speed Differential	Fault Tolerant Differential	Single Wire
Number of Bus Wires	2	2	1 (Ground Reference)
Maximum Bus Speed	***500*** **kbps**	**125 kbps** (Ltd by prop delay)	**33.33 kbps** (Ltd by prop delay)
Bus Topologies	Linear	Bus, Star, Ring	Bus, Star, Ring
Automotive Standards Documents	ISO 11898 SAE J2284	ISO - To Be Determined	SAE J2411
Parts Available	MC33989 (SBC), PCA82C250	MC33388, MC33889 (SBC Lite), MC33389 (SBC), PCA82C252	
Comments	Silicon solutions to common mode noise issues under investigation	Automatically switch to single wire reception when fault detected	Based on J1850-VPW technology with enhanced wakeup capability

CAN-C (J2284)

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





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.

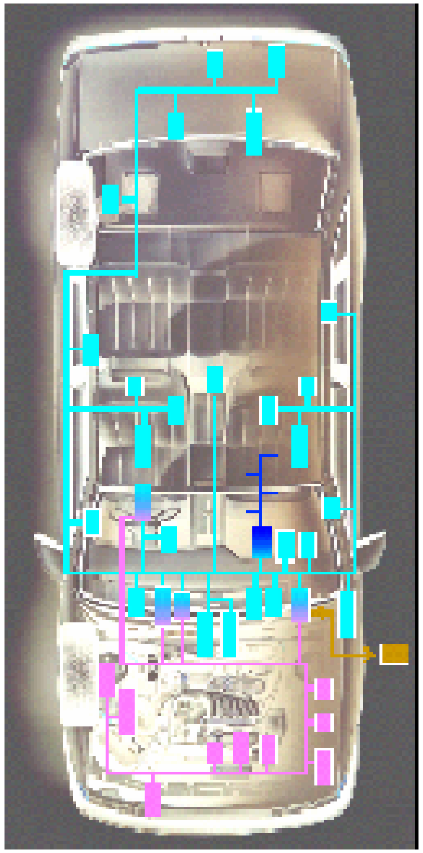
3. Filters messages.
   - Programmable hardware filter determines message accepance
   - Saves processor time by eliminating the processing of unwanted messages.
   - To achieve efficient filters on all nodes, select Identifiers carefully. Filtering allows any number of nodes to receive and simultaneously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

Applications

Embedded Connectivity Summit

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.

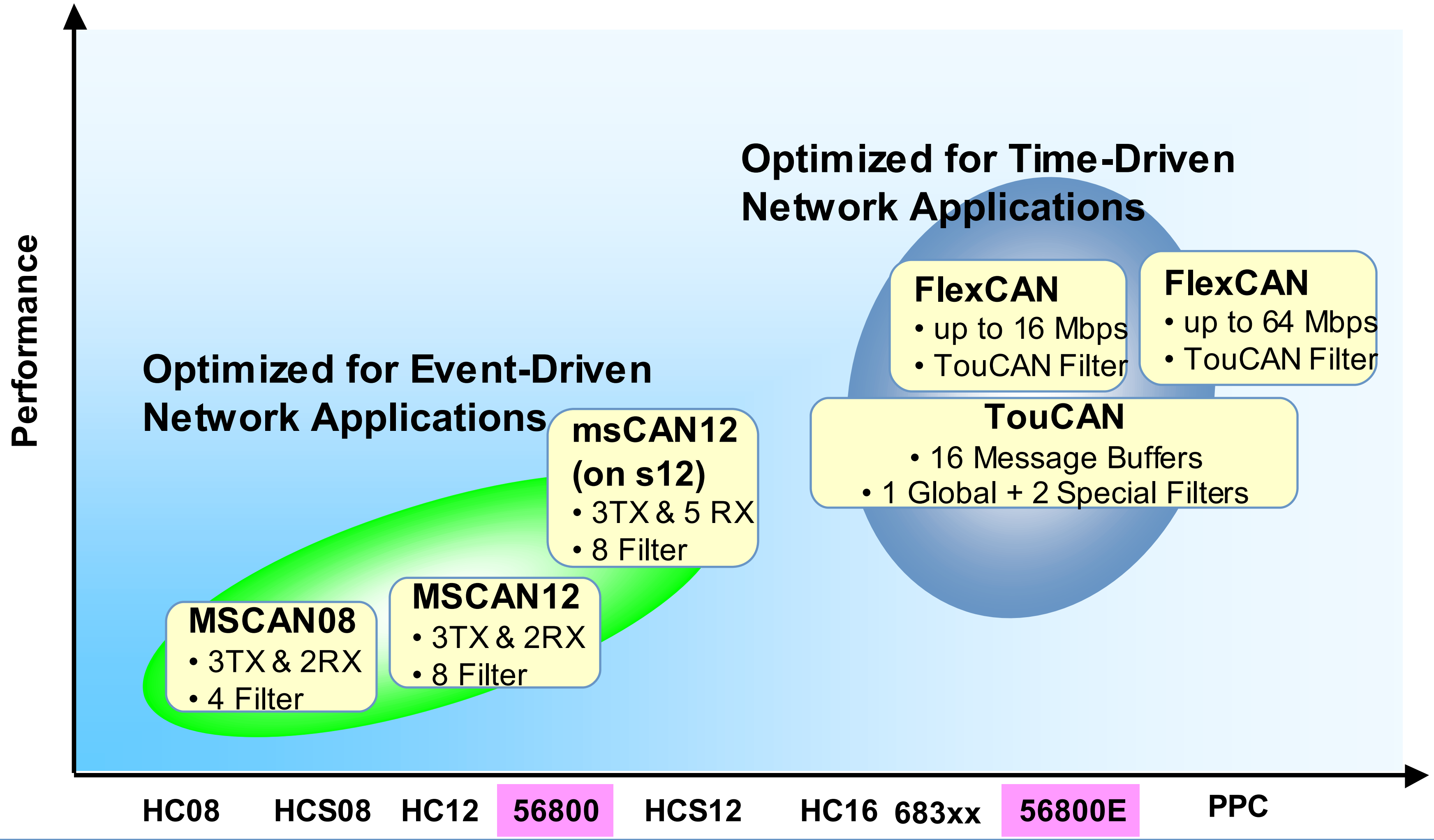
Medical: Hospitals control vital operating room components such as OR lights and tables, endoscope lights and cameras, insufflators, X-ray and ultrasound machines, video recorders, and video printers

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

Embedded Connectivity Summit

Freescale CAN Hardware Solutions



56800/E Hardware & Software

Embedded Connectivity Summit

dded Connectivity Summit

E 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

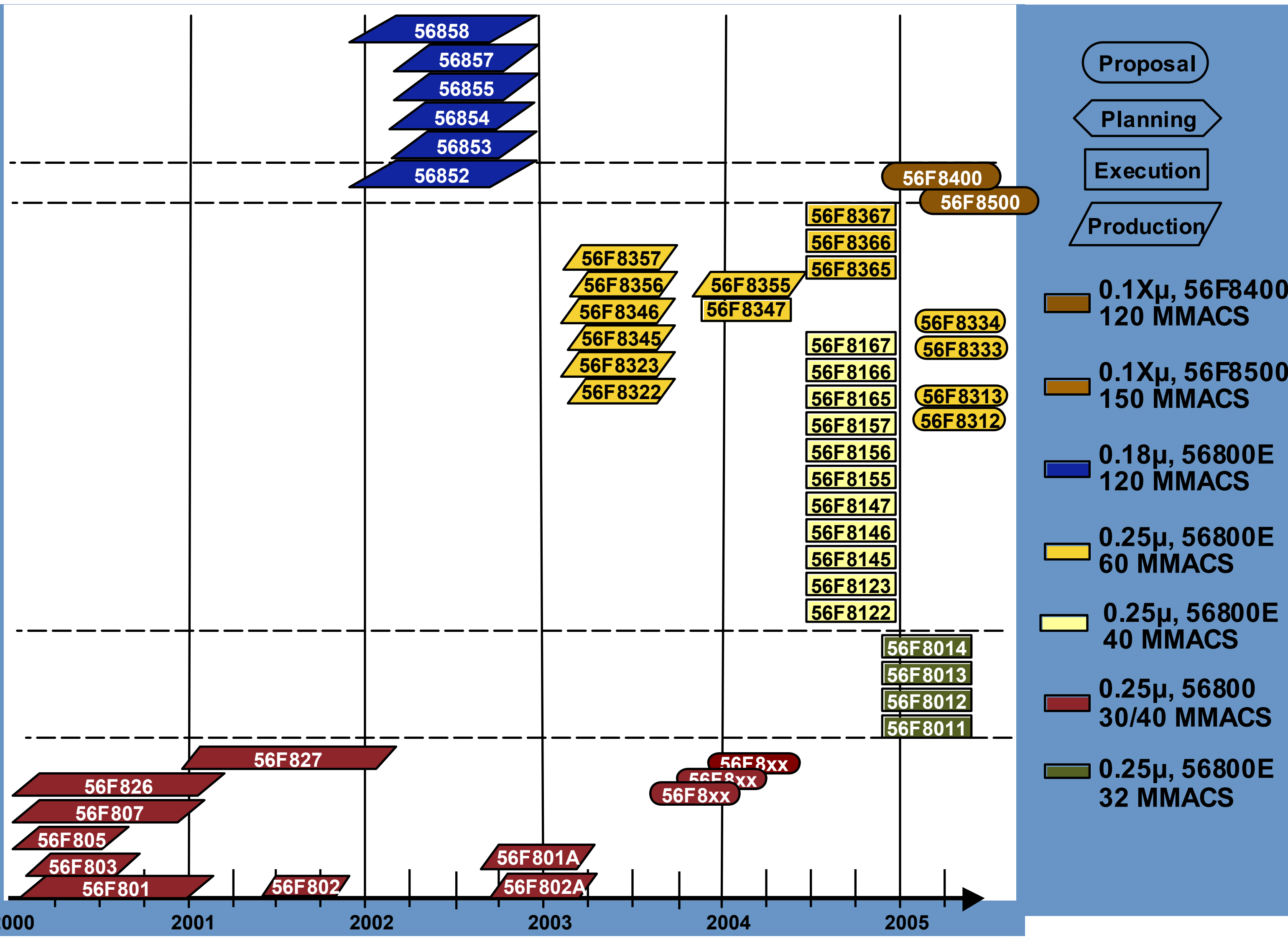
56F8300 Series
Automotive&Industrial, Flash-based, 60 MMACS, 16-512KB PFlash, 48–160 pins

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

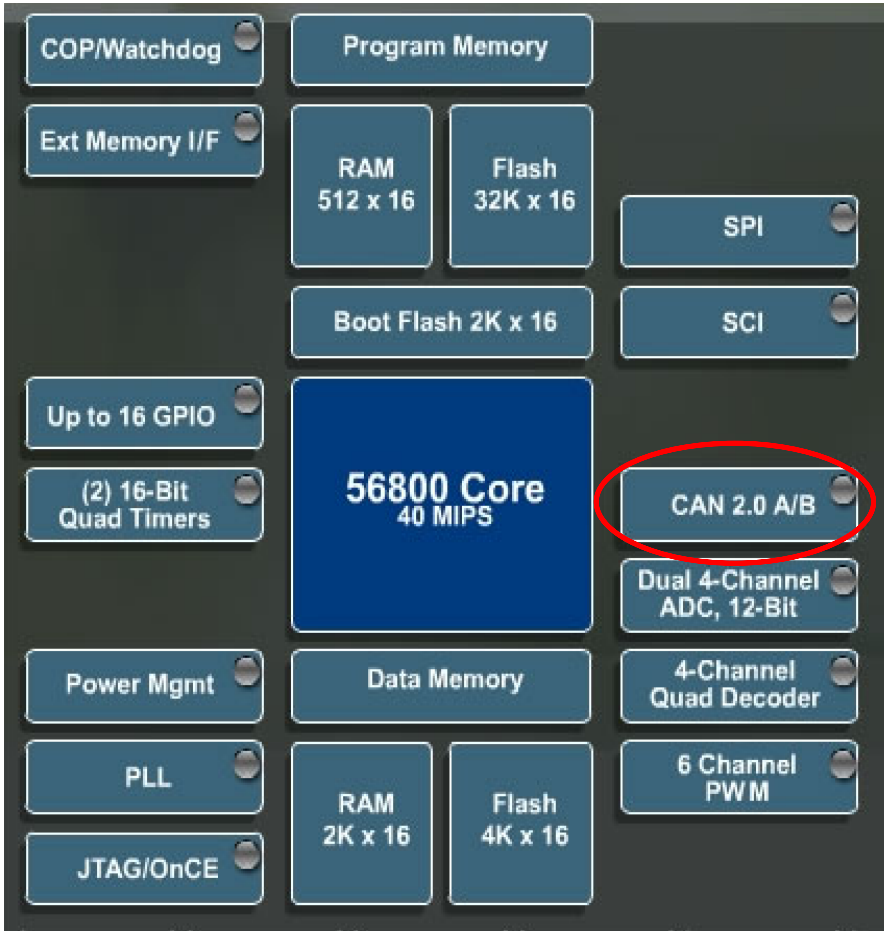
56F820 Series
General Purpose, Flash-based, 40 MMACS, 100–128 pins

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



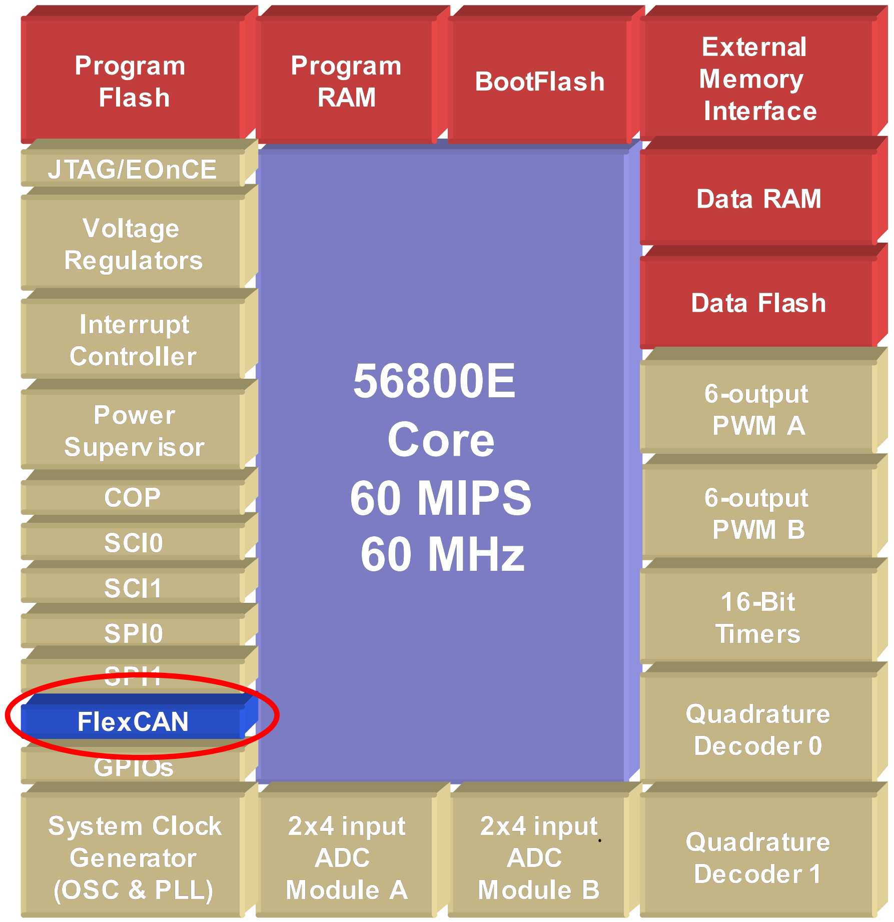
Launched by Motorola

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

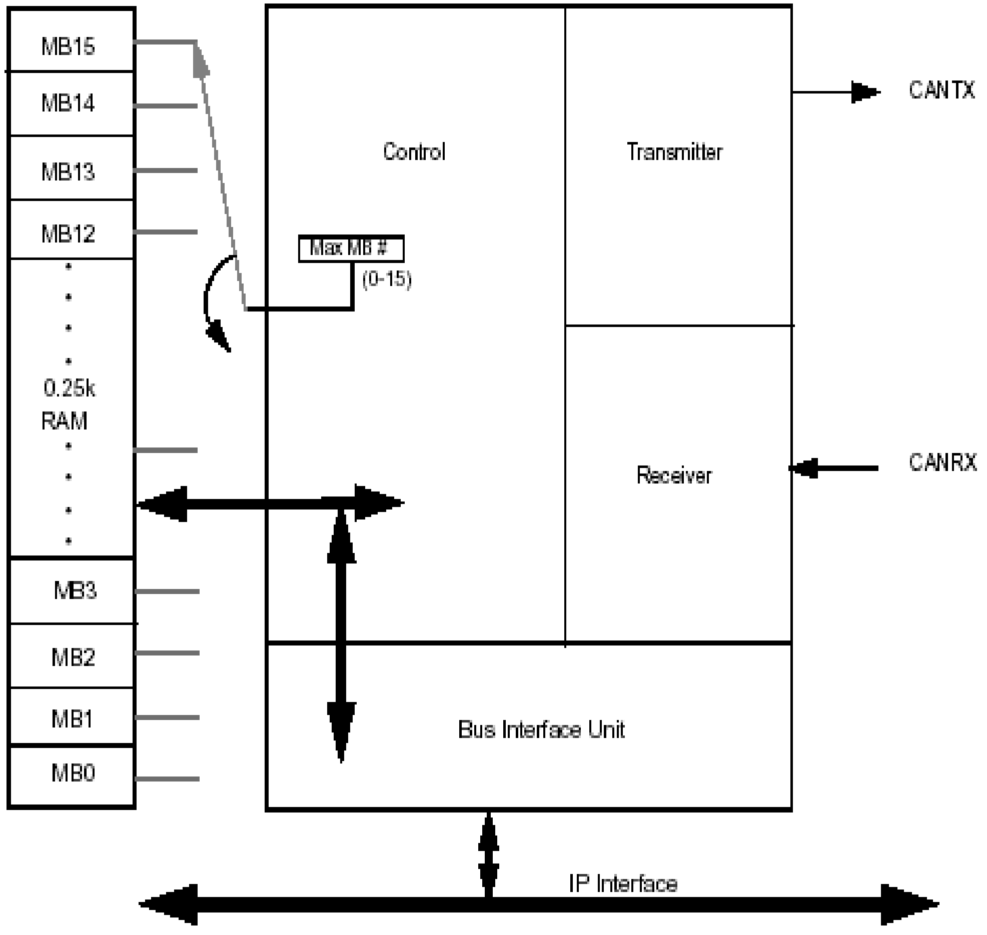


Figure 7-1. FlexCAN Block Diagram and Pinout

Typical Can System

- Two-wire differential physical interface
- Transceiver Provides
  - Transmit Drive
  - Wave Shaping
  - Receive/Compare Functions
  - Protection from defective CAN bus

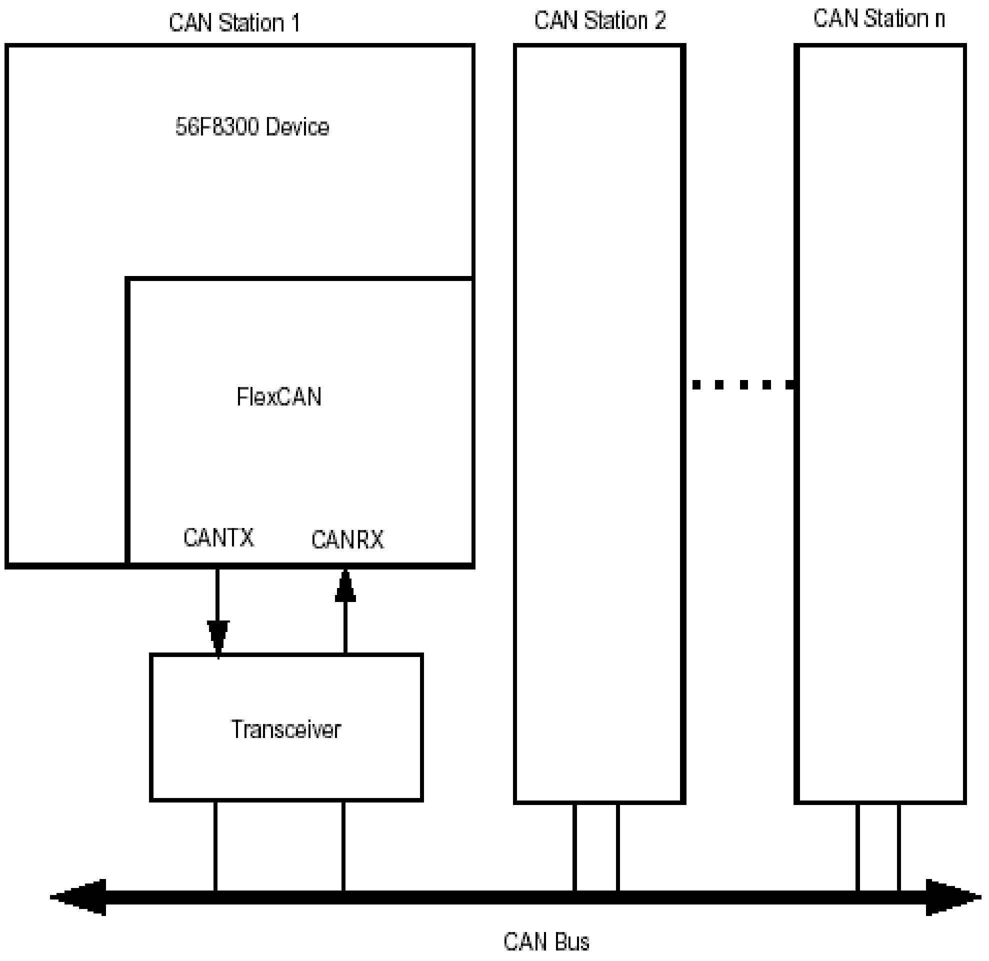


Figure 7-2. Typical CAN system

Message Buffer Structure

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

	15–8	7–4	3–0		
$0	TIME_STAMP	CODE	LENGTH		CONTROL/STATUS
$1	ID[28:18]	SR R	IDE	ID[17-15]	ID_HIGH
$2	ID[14-0]			RTR	ID_LOW
$3	DATA BYTE 0	DATA BYTE 1			
$4	DATA BYTE 2	DATA BYTE 3			
$5	DATA BYTE 4	DATA BYTE 5			
$6	DATA BYTE 6	DATA BYTE 7			
$7	Reserved				

Figure 7-3. Extended ID Message Buffer Structure

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

	15–8	7–4	3–0				
$0	TIME_STAMP	CODE	LENGTH				CONTROL/STATUS
$1	ID[28:18]	RTR	0	0	0	0	ID_HIGH
$2	16-BIT TIME STAMP						ID_LOW
$3	DATA BYTE 0	DATA BYTE 1					
$4	DATA BYTE 2	DATA BYTE 3					
$5	DATA BYTE 4	DATA BYTE 5					
$6	DATA BYTE 6	DATA BYTE 7					
$7	Reserved						

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)

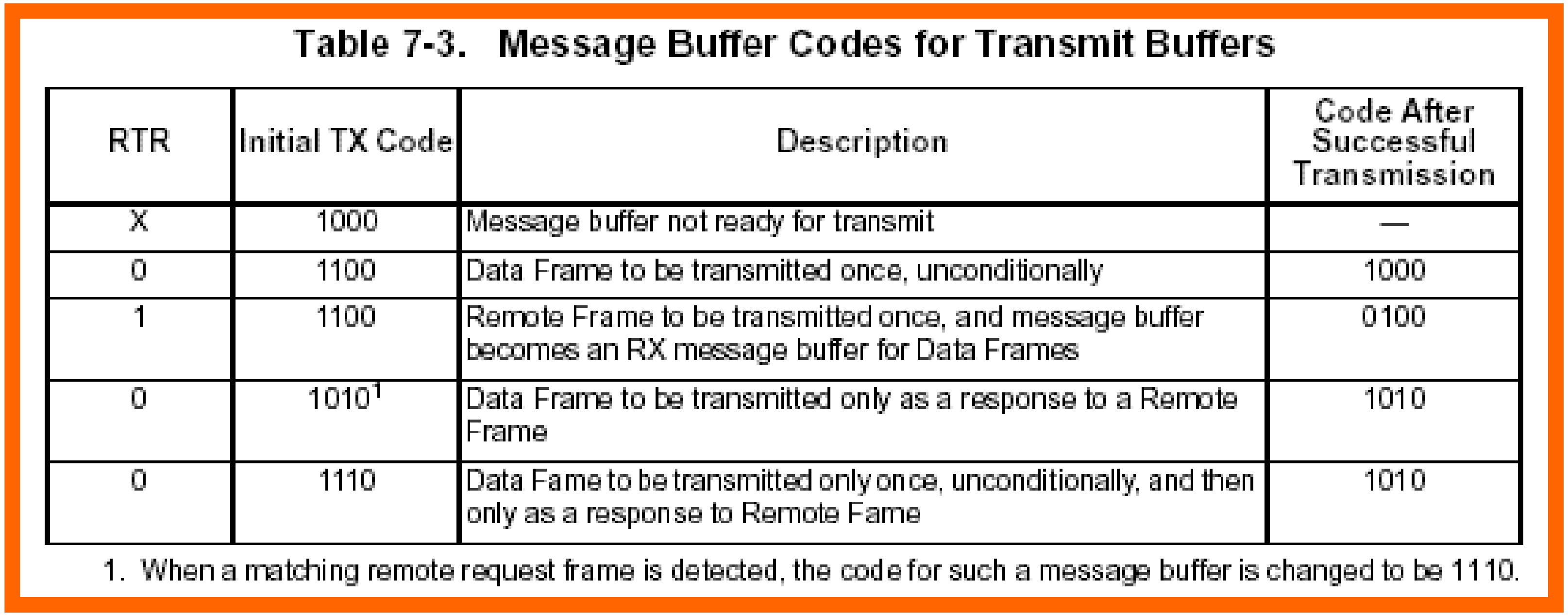


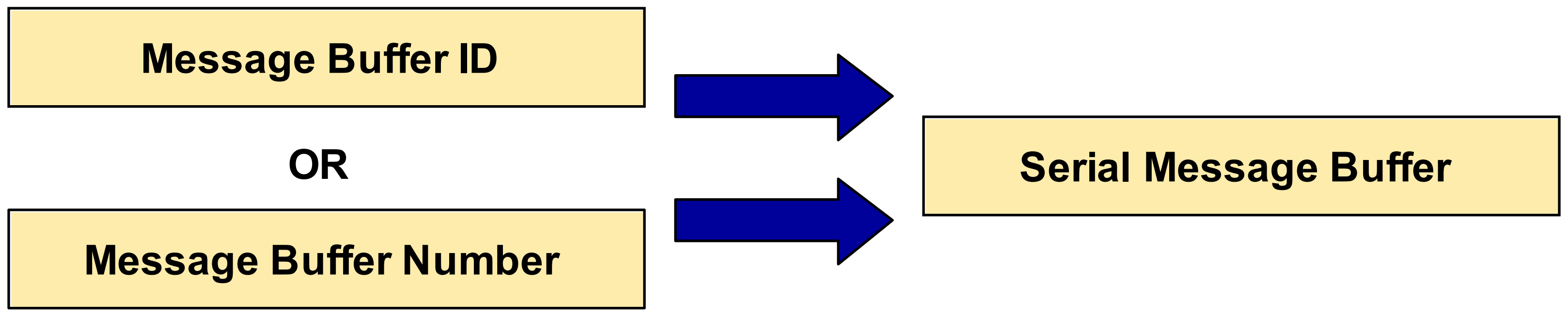
Table 7-3. Message Buffer Codes for Transmit Buffers

RTR	Initial TX Code	Description	Code After Successful Transmission
X	1000	Message buffer not ready for transmit	—
0	1100	Data Frame to be transmitted once, unconditionally	1000
1	1100	Remote Frame to be transmitted once, and message buffer becomes an RX message buffer for Data Frames	0100
0	1010[1]	Data Frame to be transmitted only as a response to a Remote Frame	1010
0	1110	Data Fame to be transmitted only once, unconditionally, and then only as a response to Remote Fame	1010

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)**

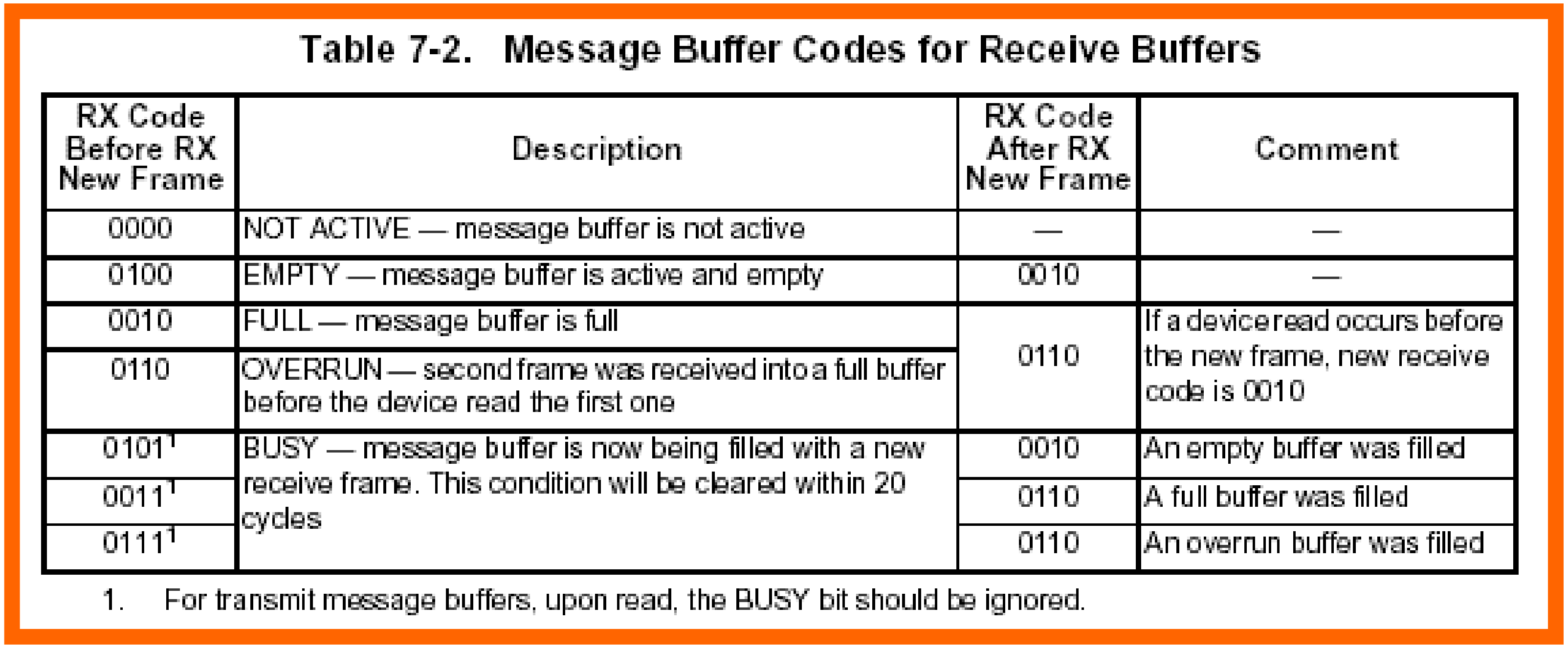


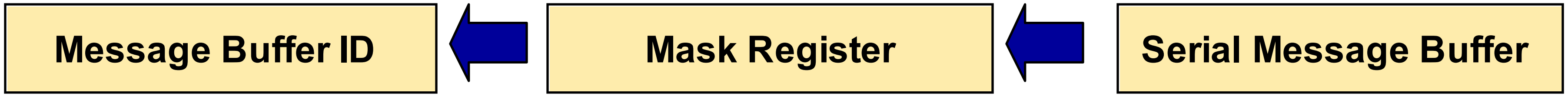
Table 7-2. Message Buffer Codes for Receive Buffers

RX Code Before RX New Frame	Description	RX Code After RX New Frame	Comment
0000	NOT ACTIVE — message buffer is not active	—	—
0100	EMPTY — message buffer is active and empty	0010	—
0010	FULL — message buffer is full	0110	If a device read occurs before the new frame, new receive code is 0010
0110	OVERRUN — second frame was received into a full buffer before the device read the first one		
0101[1]	BUSY — message buffer is now being filled with a new receive frame. This condition will be cleared within 20 cycles	0010	An empty buffer was filled
0011[1]		0110	A full buffer was filled
0111[1]		0110	An overrun buffer was filled

1. For transmit message buffers, upon read, the BUSY bit should be ignored.

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.

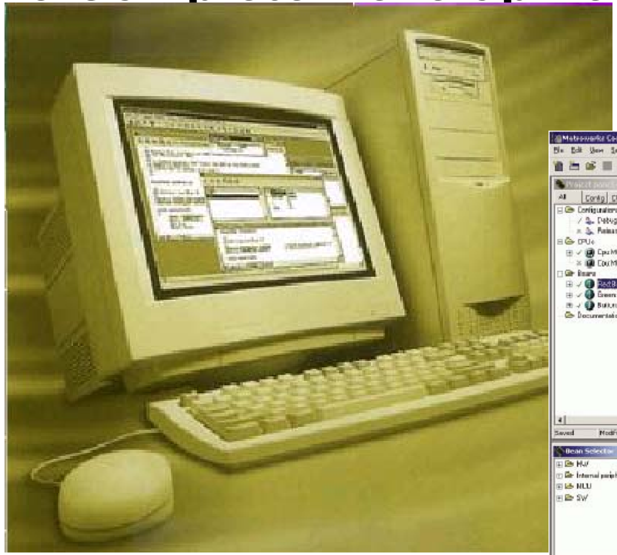
	Mask Register	Base ID ID28…..ID18	IDE	Extended ID ID17…..ID0	Matching Msg Buffer
FlexCAN Configuration	Global Mask	11111111110	-	111111100000000001	
	RX Buffer 14	01111111111	-	111111100000000000	
	MB2 ID	11111111000	0	---------------	
	MB3 ID	11111111000	1	010101010101010101	
	MB14 ID	11111111000	1	010101010101010101	
Received Msg ID	SMB	11111111001	1	010101010101010101	MB3
	SMB	11111111001	0	---------------	MB2
	SMB	11111111001	1	010101010101010100	-
	SMB	01111111000	0	---------------	-
	SMB	01111111000	1	010101010101010101	MB14
	SMB	10111111000	1	010101010101010101	-

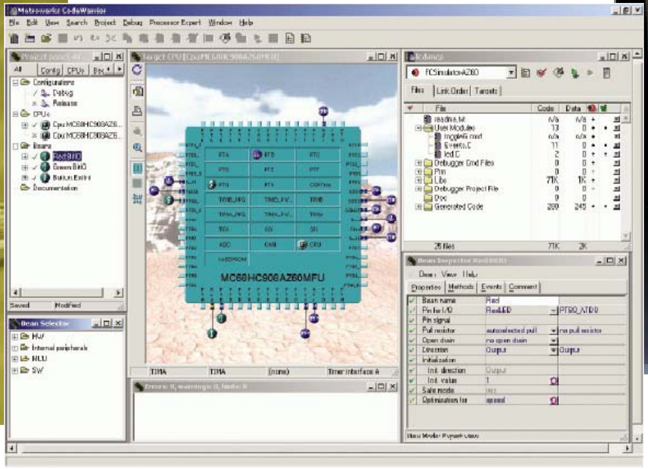
RX Process

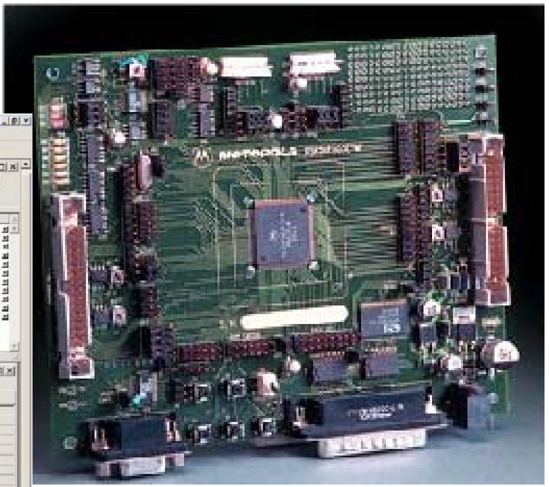
- **Software - Read a receive frame from its MB:**
  - ✓ **Read Control/Status word (mandatory—activates internal lock for this buffer)**
  - ✓ **Read ID (Optional - essential only if a mask was used)**
  - ✓ **Read Data field word(s)**
  - ✓ **Release internal lock by one of the following ways:**
    - 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 windows 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™

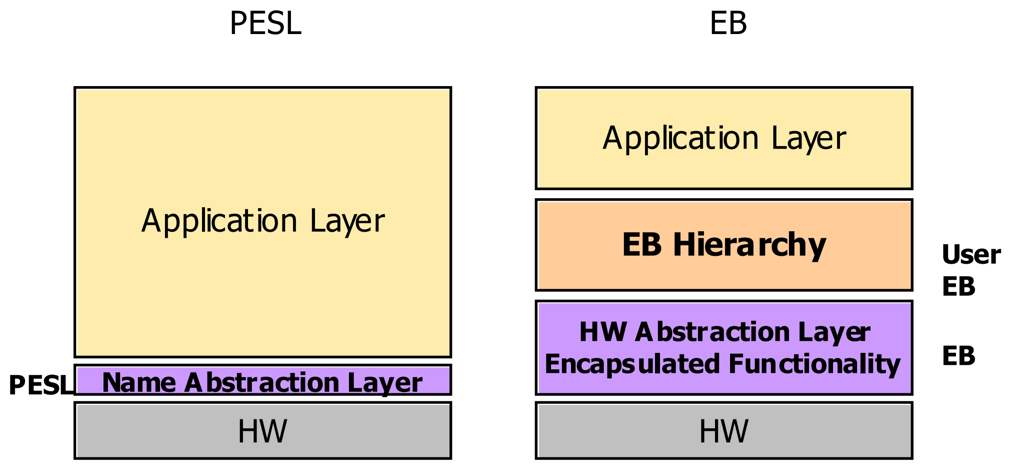
- Supports rapid application development
- Enables component oriented programming
- Provides expert advice if necessary
- Delivers instant functionality of generated code
- Provides tested ready-to-use code

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

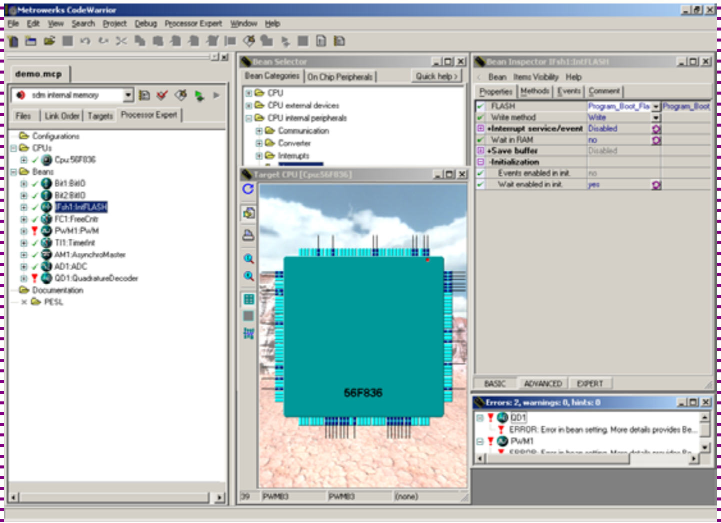
- Developed by experienced programmers of embedded systems
- Expert knowledge system is working on the background of PE and checks all the settings
- Provides context help and access to CPU/MCU vendor documentation
- All EB delivered by UNIS are tested according to ISO testing procedures (UNIS is ISO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

Processor Expert TM
The Human Interface to the World of Silicon
21ST CENTURY INFORMATION TECHNOLOGY AND CONTROL
IST prize Winner
UNIS
Developers of Embedded Beans TM ...
PROCESSOR EXPERT



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

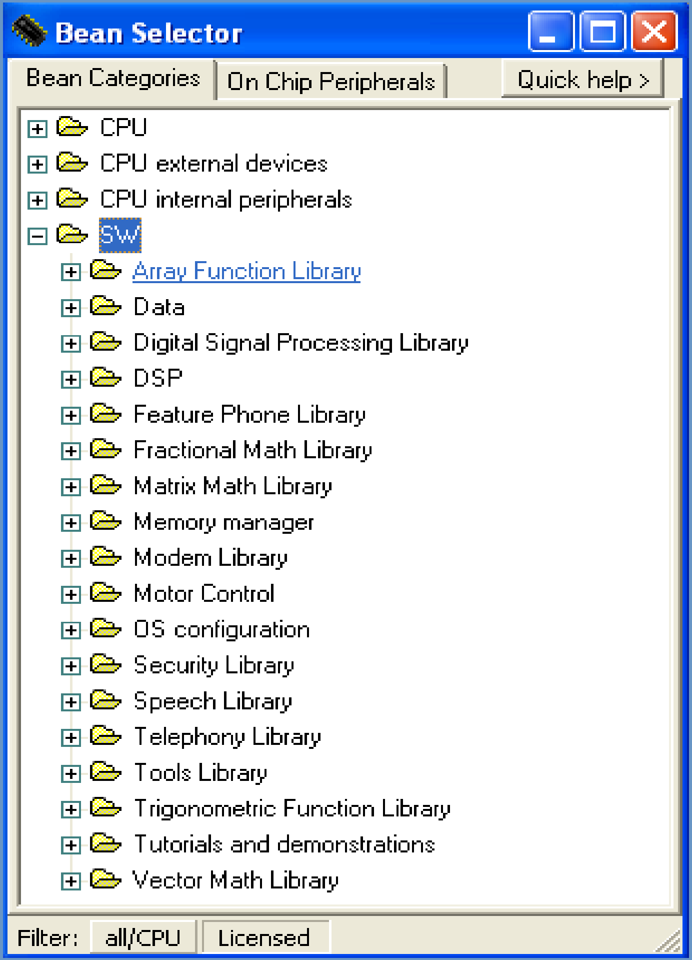
Modem Libraries
- V.8bis, V.21, V.22bis, V.42bis

Security Libraries
- RSA, DES, 3DES,

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

Math Libraries
- Matrix, Fractional, Vector
- Trigonometric

Tools Library
- Cycle Count, FIFO, FileIO, Test



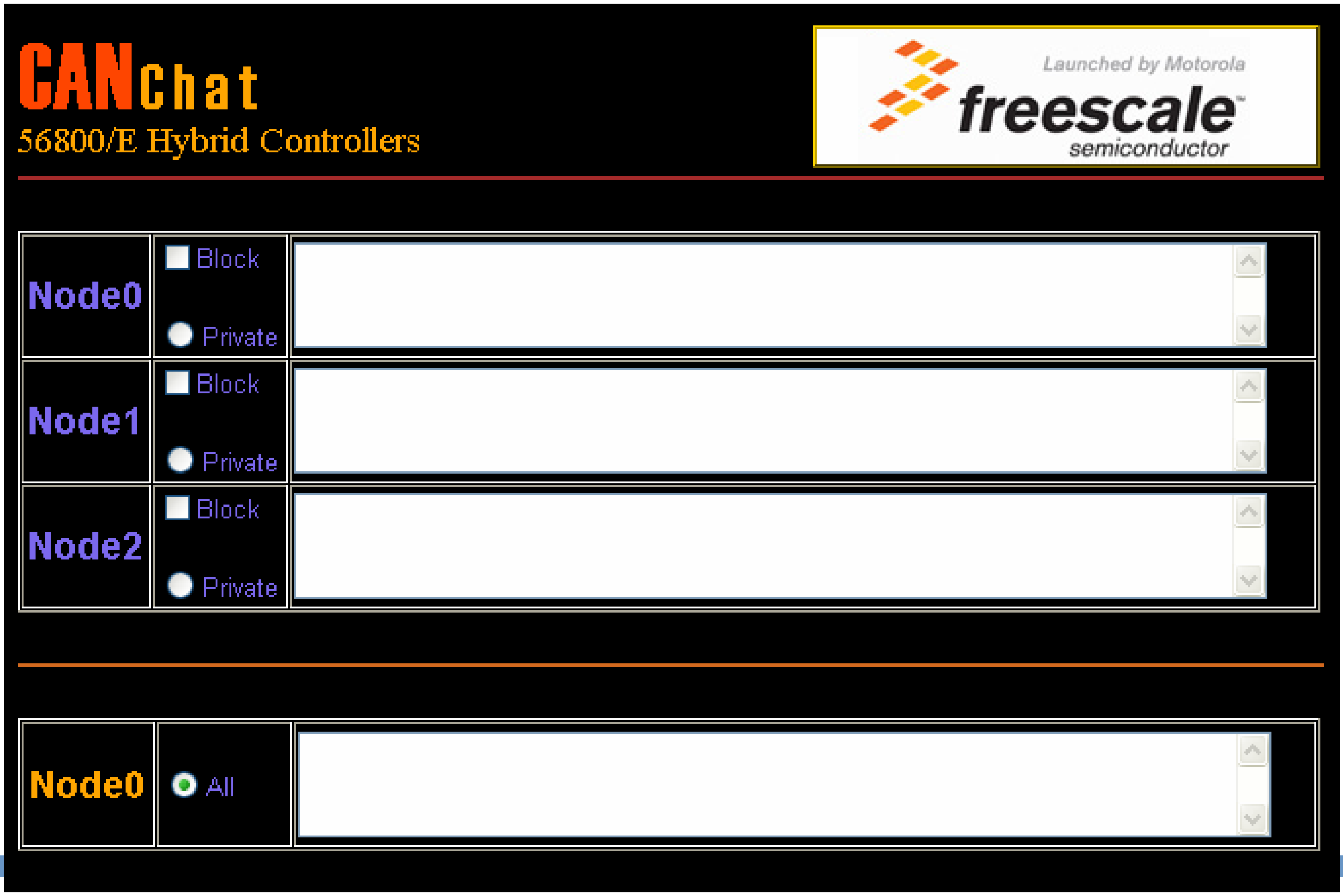
Developing Applications

Embedded Connectivity Summit

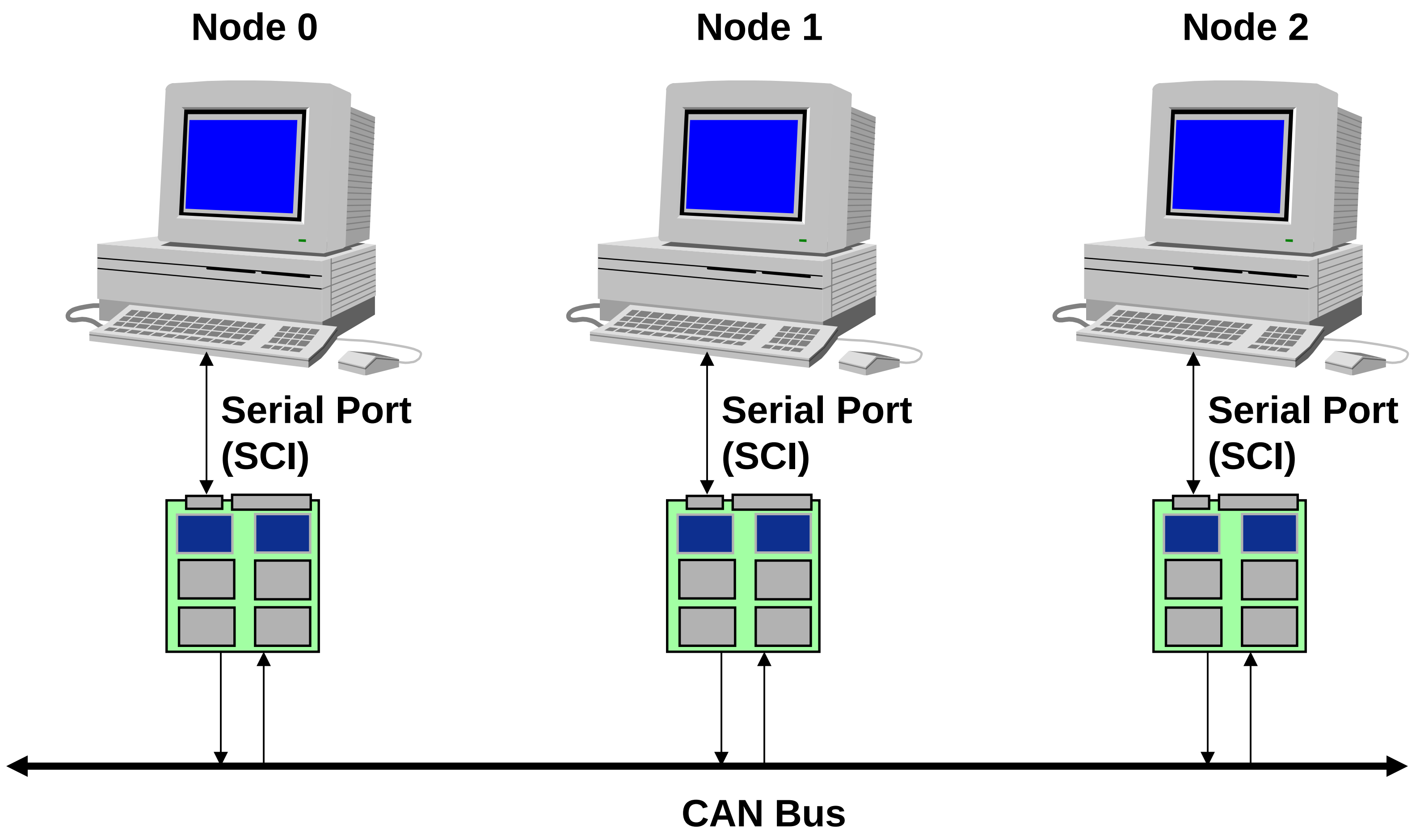
Launched by Motorola
freescale semiconductor

Task Description

Develop a "Chat Room" application that uses FlexCAN acceptance filters to implement Broadcast and Private communication channels and Blocking.



System Block Diagram



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)

Msg Destination	Msg Source
001 – to Node 0	001 – from Node 0
010 – to Node 1	010 – from Node 1
100 – to Node 2	100 – from Node 2
000 – Broadcast	

ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21	ID20	ID19	ID18
			0	0	0	0	0			

Examples:		
	Node 0 broadcasting a message:	00000000001
	Node 1 private message to Node 2:	01000000010
	Node 2 broadcasting a message:	00000000100

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)

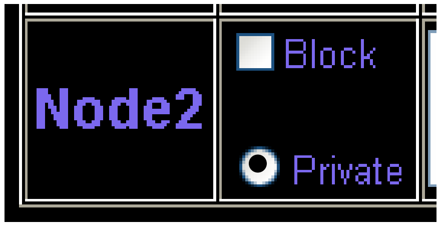
Example: Accept messages broadcast to all.

	Dest				Src			
GMR:	1 1 1	0	0	0	0 0	0	0	0
MB0 ID:	0 0 0	0	0	0	0 0	0	0	0

Broadcast messages are sent with 000 in the destination field!

Design Message Filtering Scheme

- ❖ Accept Private messages (source = Node ID)


Node2 ☐ Block ◉ Private

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

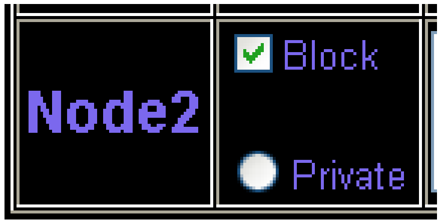
Example: Accept messages specifically destined for Node 2 (ID=100)

```
            Dest             Src
GMR:        1 1 1 0 0 0 0 0 0 0 0
MB0 ID:     1 0 0 0 0 0 0 0 0 0 0
```

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)

	Dest	Src
GMR:	1 1 1 0 0 0 0 0	1 0 0
MB0 ID:	0 1 0 0 0 0 0 0	0 0 0

Messages are sent with NodeID in the Src field!

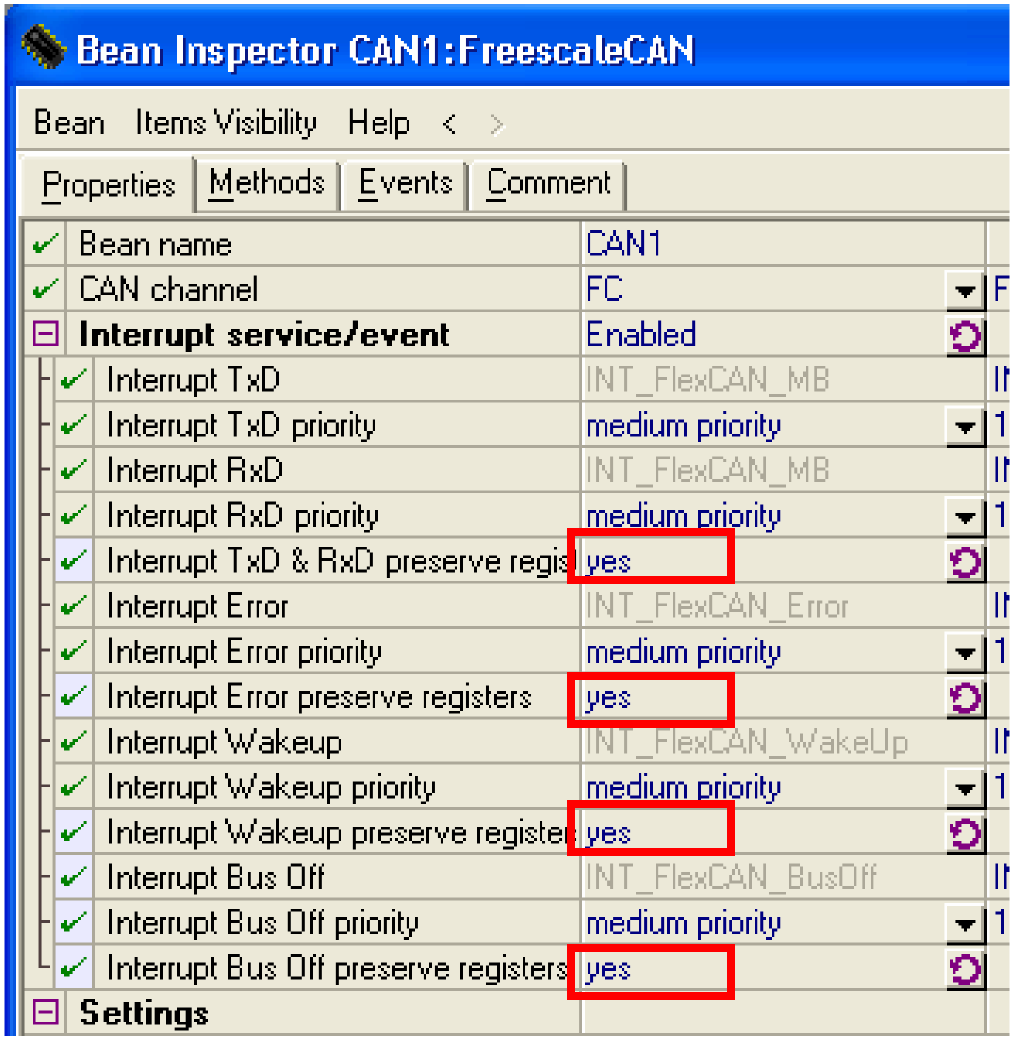
NodeID 0	001
NodeID 1	010
NodeID 2	100

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.



✓ Bean name	CAN1
✓ CAN channel	FC
Interrupt service/event	Enabled
✓ Interrupt TxD	INT_FlexCAN_MB
✓ Interrupt TxD priority	medium priority
✓ Interrupt RxD	INT_FlexCAN_MB
✓ Interrupt RxD priority	medium priority
✓ Interrupt TxD & RxD preserve regis	yes
✓ Interrupt Error	INT_FlexCAN_Error
✓ Interrupt Error priority	medium priority
✓ Interrupt Error preserve registers	yes
✓ Interrupt Wakeup	INT_FlexCAN_WakeUp
✓ Interrupt Wakeup priority	medium priority
✓ Interrupt Wakeup preserve registe	yes
✓ Interrupt Bus Off	INT_FlexCAN_BusOff
✓ Interrupt Bus Off priority	medium priority
✓ Interrupt Bus Off preserve registers	yes
Settings	

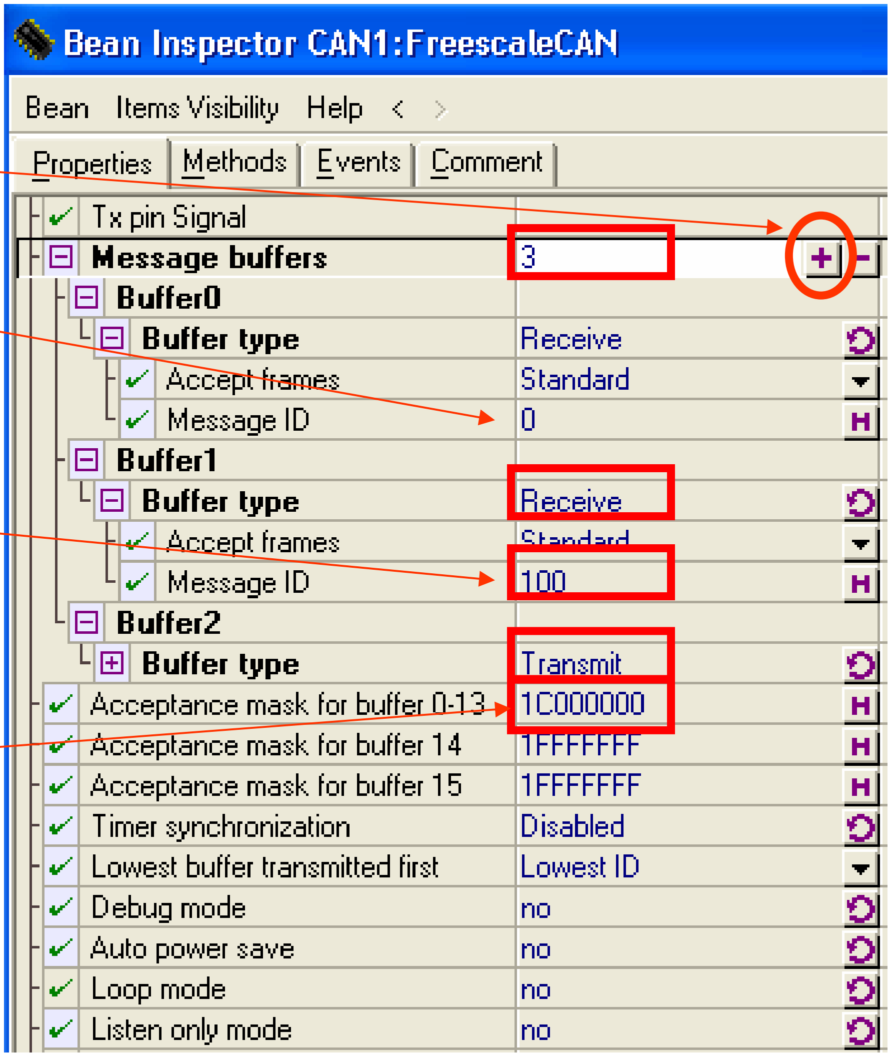
Steps:

- 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

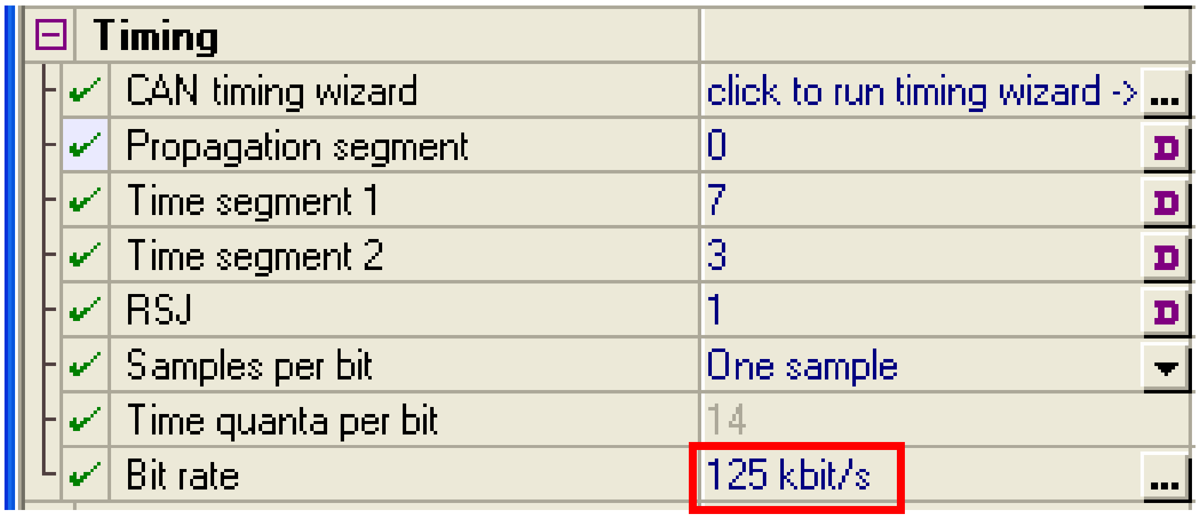


Bean Inspector CAN1:FreescaleCAN

Tx pin Signal	
Message buffers	3
Buffer0	
Buffer type	Receive
Accept frames	Standard
Message ID	0
Buffer1	
Buffer type	Receive
Accept frames	Standard
Message ID	100
Buffer2	
Buffer type	Transmit
Acceptance mask for buffer 0-13	1C000000
Acceptance mask for buffer 14	1FFFFFFF
Acceptance mask for buffer 15	1FFFFFFF
Timer synchronization	Disabled
Lowest buffer transmitted first	Lowest ID
Debug mode	no
Auto power save	no
Loop mode	no
Listen only mode	no

Steps:

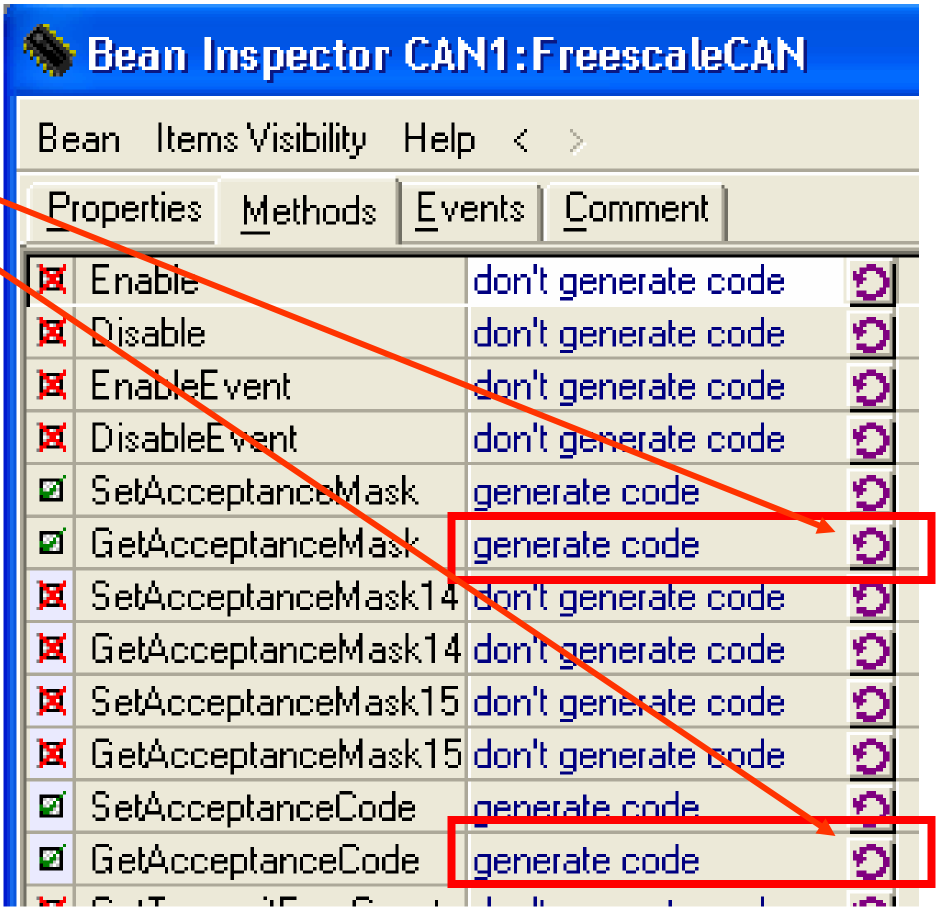
- Set bit rate to 125 kbps



Timing	
CAN timing wizard	click to run timing wizard ->
Propagation segment	0
Time segment 1	7
Time segment 2	3
RSJ	1
Samples per bit	One sample
Time quanta per bit	14
Bit rate	125 kbit/s

Steps:

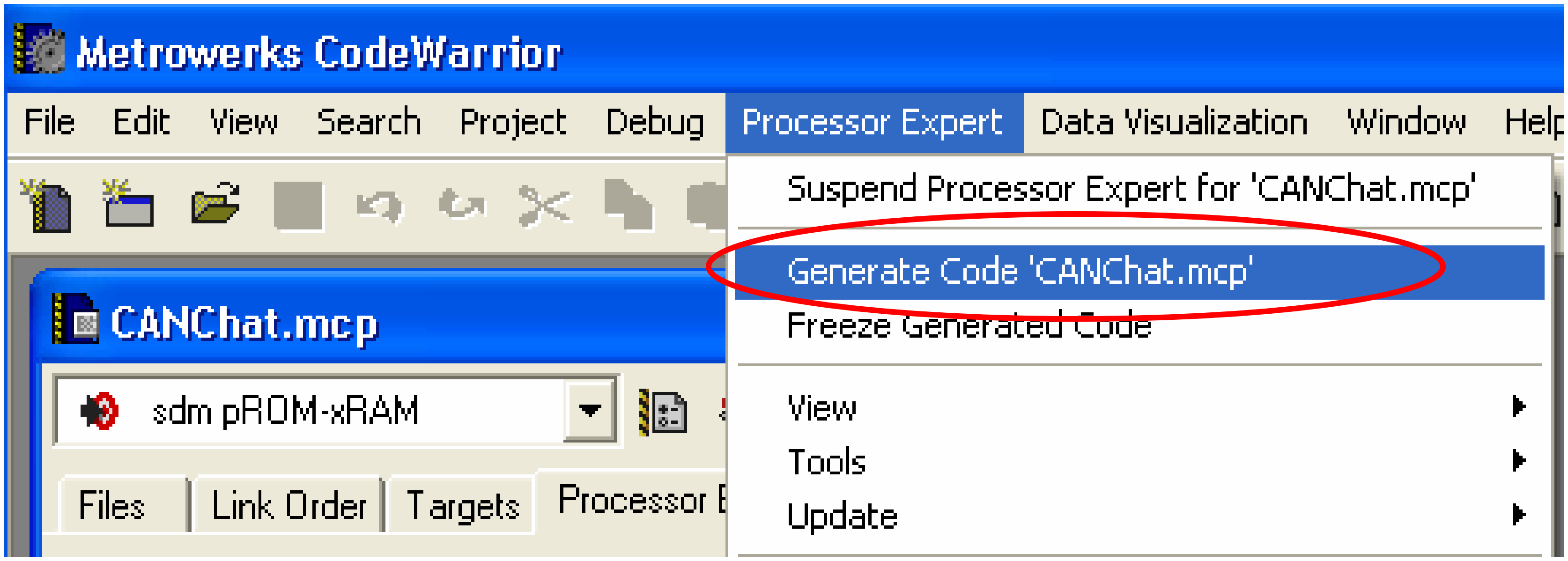
- Enable Code Generation



Bean Inspector CAN1:FreescaleCAN

✗	Enable	don't generate code
✗	Disable	don't generate code
✗	EnableEvent	don't generate code
✗	DisableEvent	don't generate code
✓	SetAcceptanceMask	generate code
✓	GetAcceptanceMask	generate code
✗	SetAcceptanceMask14	don't generate code
✗	GetAcceptanceMask14	don't generate code
✗	SetAcceptanceMask15	don't generate code
✗	GetAcceptanceMask15	don't generate code
✓	SetAcceptanceCode	generate code
✓	GetAcceptanceCode	generate code

Steps: Generate Code



Steps: Open source file CANChat.c

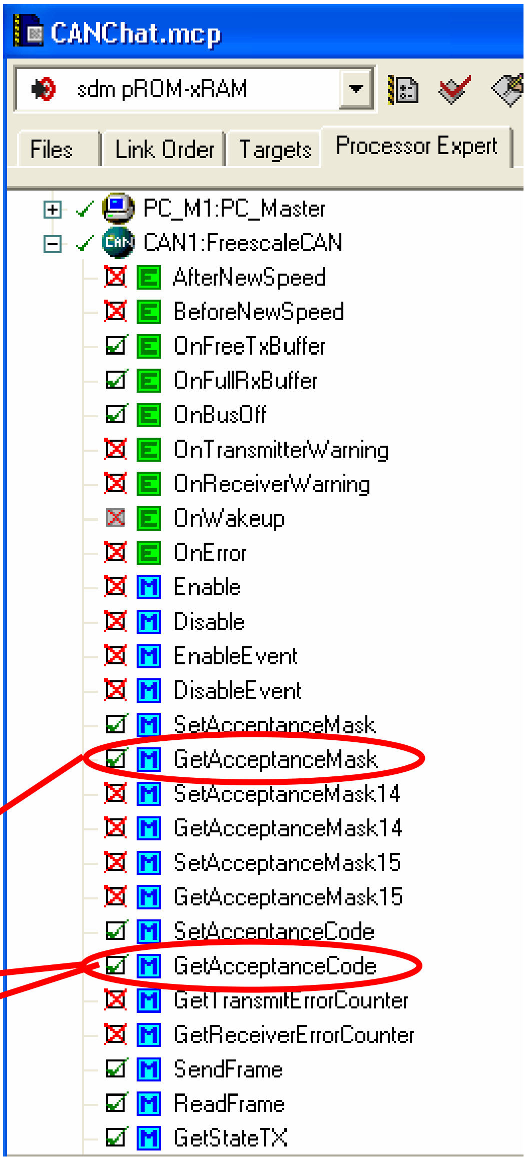
❖ **Configure NodeID: This determine the Source ID for transmission and corresponding MB1 ID should be configured for Private message receipt.**

```
/* 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

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

  /*** Processor Expert internal initialization. DON'T REMOVE
PE_low_level_init();
  /*** End of Processor Expert internal initialization. ***/

  /* Get Initialization MB IDs */
  GlobalMask_ID=0x1FFFFFFF& CAN1_GetAcceptanceMask();
  CAN1_GetAcceptanceCode(MB_RX_BROADCAST, \
                                                    &MsgBuffer0_ID);
  CAN1_GetAcceptanceCode(MB_RX_PRIVATE, \
                                                    &MsgBuffer1_ID);
```


CANChat.mcp

sdm pROM-xRAM

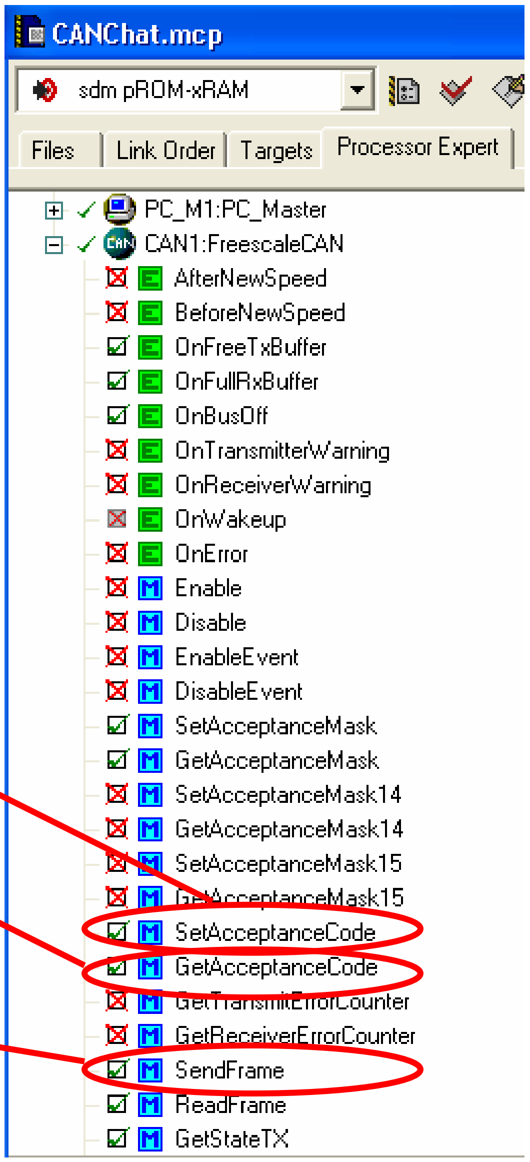
Files | Link Order | Targets | Processor Expert

- PC_M1:PC_Master
- CAN1:FreescaleCAN
  - AfterNewSpeed
  - BeforeNewSpeed
  - OnFreeTxBuffer
  - OnFullRxBuffer
  - OnBusOff
  - OnTransmitterWarning
  - OnReceiverWarning
  - OnWakeup
  - OnError
  - Enable
  - Disable
  - EnableEvent
  - DisableEvent
  - SetAcceptanceMask
  - GetAcceptanceMask
  - SetAcceptanceMask14
  - GetAcceptanceMask14
  - SetAcceptanceMask15
  - GetAcceptanceMask15
  - SetAcceptanceCode
  - GetAcceptanceCode
  - GetTransmitErrorCounter
  - GetReceiverErrorCounter
  - SendFrame
  - ReadFrame
  - GetStateTX

Embedded Connectivity Summit

Steps: Drag-n-drop methods (CANChat.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_Byte),
                      &Transmit_Byte, FALSE);
    if (txErr == ERR_OK)
```



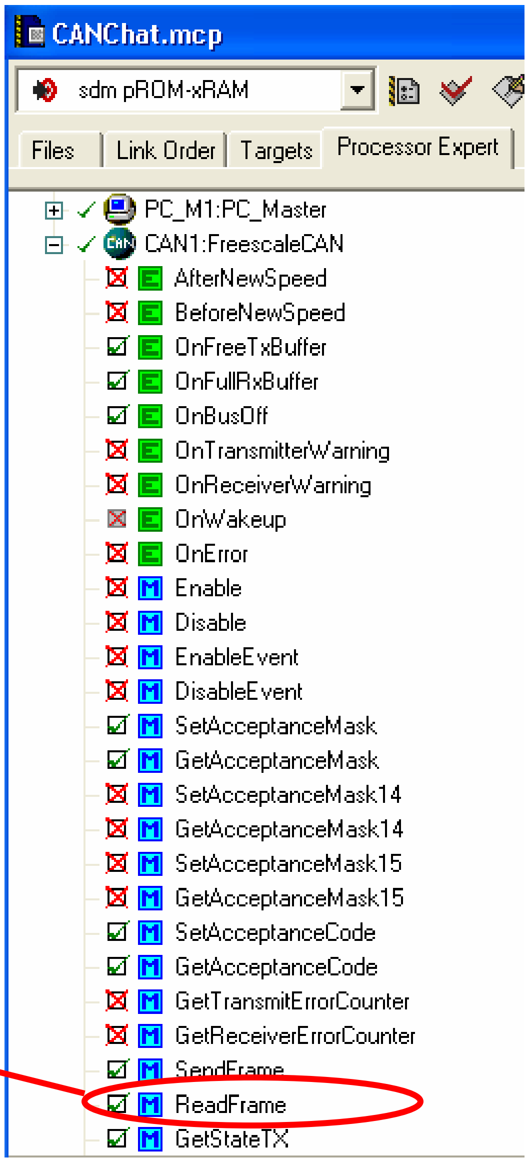
CANChat.mcp

sdm pROM-xRAM

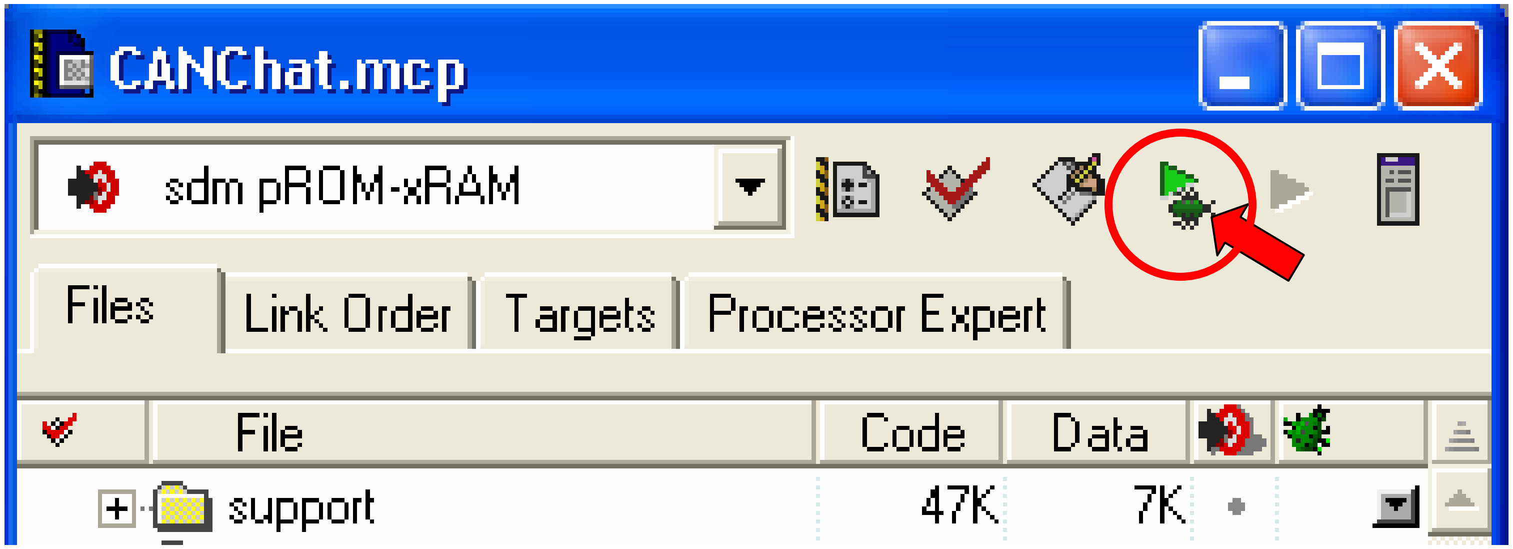
- ⊞ ✓ PC_M1:PC_Master
- ⊟ ✓ CAN1:FreescaleCAN
  - ☒ AfterNewSpeed
  - ☒ BeforeNewSpeed
  - ☑ OnFreeTxBuffer
  - ☑ OnFullRxBuffer
  - ☑ OnBusOff
  - ☒ OnTransmitterWarning
  - ☒ OnReceiverWarning
  - ☒ OnWakeup
  - ☒ OnError
  - ☒ Enable
  - ☒ Disable
  - ☒ EnableEvent
  - ☒ DisableEvent
  - ☑ SetAcceptanceMask
  - ☑ GetAcceptanceMask
  - ☒ SetAcceptanceMask14
  - ☒ GetAcceptanceMask14
  - ☒ SetAcceptanceMask15
  - ☒ GetAcceptanceMask15
  - ☑ SetAcceptanceCode
  - ☑ GetAcceptanceCode
  - ☒ GetTransmitErrorCounter
  - ☒ GetReceiverErrorCounter
  - ☑ SendFrame
  - ☑ ReadFrame
  - ☑ GetStateTX

Steps: Drag-n-drop methods (Events.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();
}
```

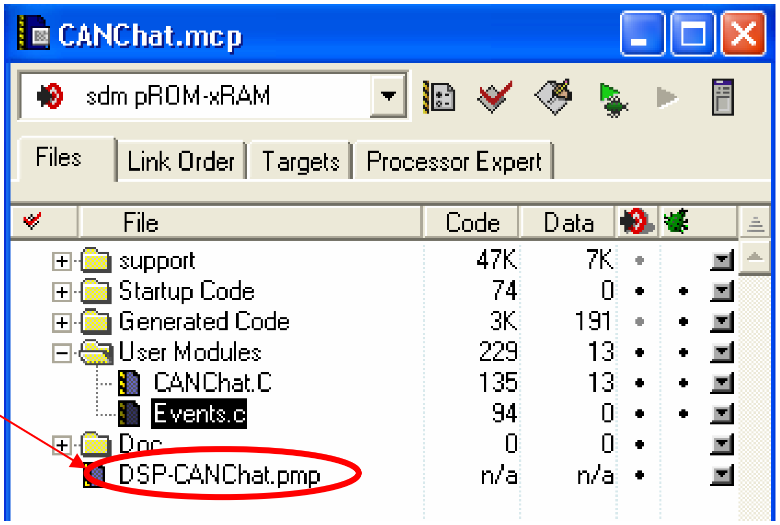


Build, Download, and Execute



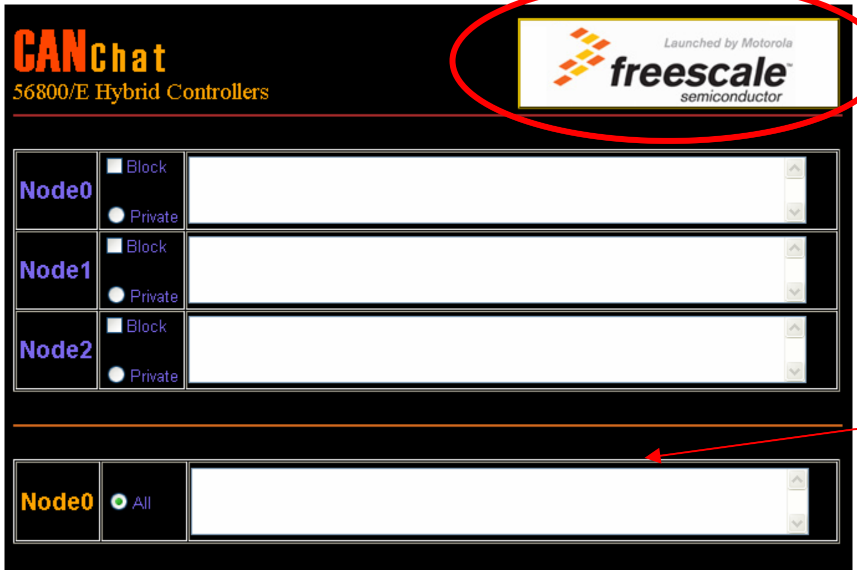
Launch PC Master Application

- Double Click CANChat.pmp to launch


CANChat.mcp

sdm pROM-xRAM

File	Code	Data
support	47K	7K
Startup Code	74	0
Generated Code	3K	191
User Modules	229	13
CANChat.C	135	13
Events.c	94	0
Doc	0	0
DSP-CANChat.pmp	n/a	n/a


CANChat
56800/E Hybrid Controllers

- Click Logo to activate and/or clear displays

- Type message in OutBox

Summary

Embedded Connectivity Summit

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.

56800E

Thank You!