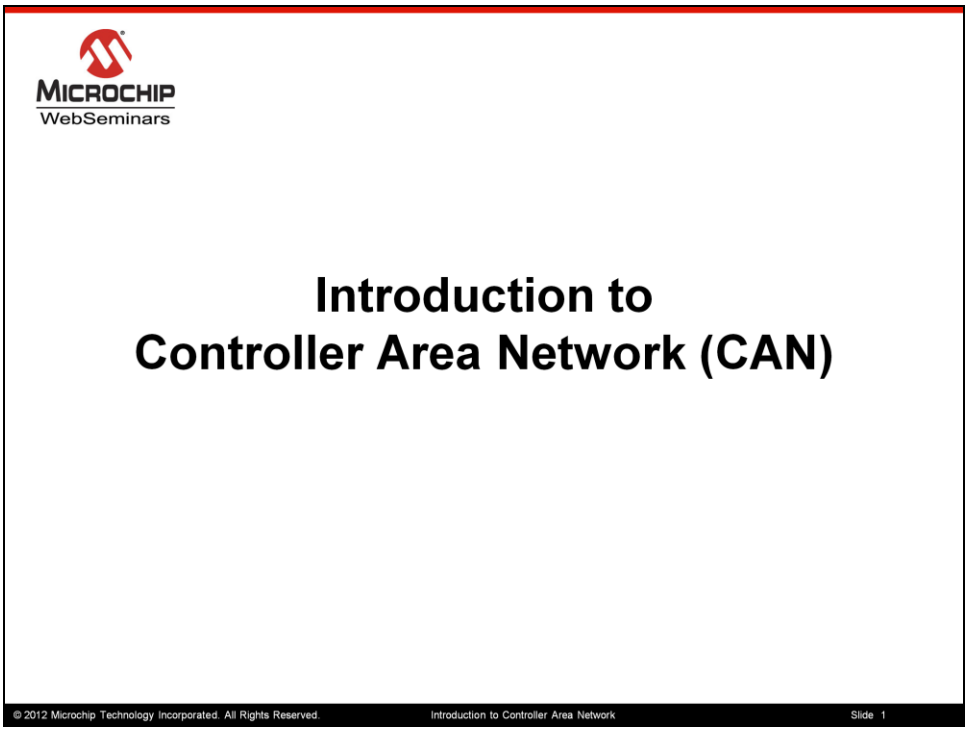
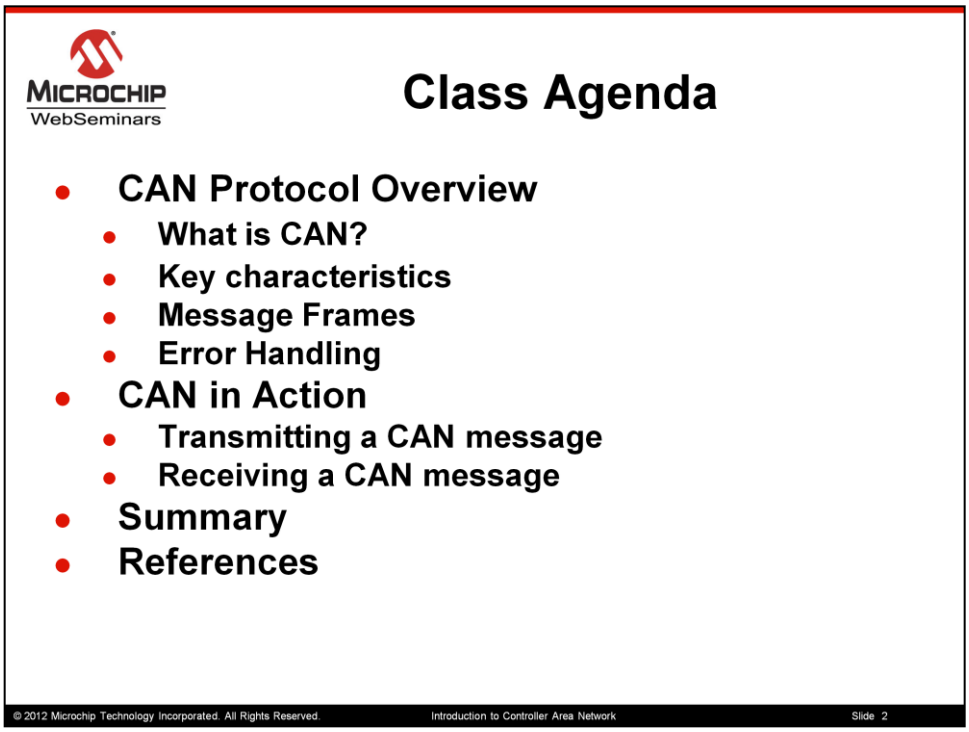


Welcome to the “Introduction to Controller Area Network” web seminar

My name is William Stuart, and I am a Applications Engineer for the Automotive Products Group within Microchip.

This web seminar today will last roughly 45 minutes and will provide you with a high level overview of CAN



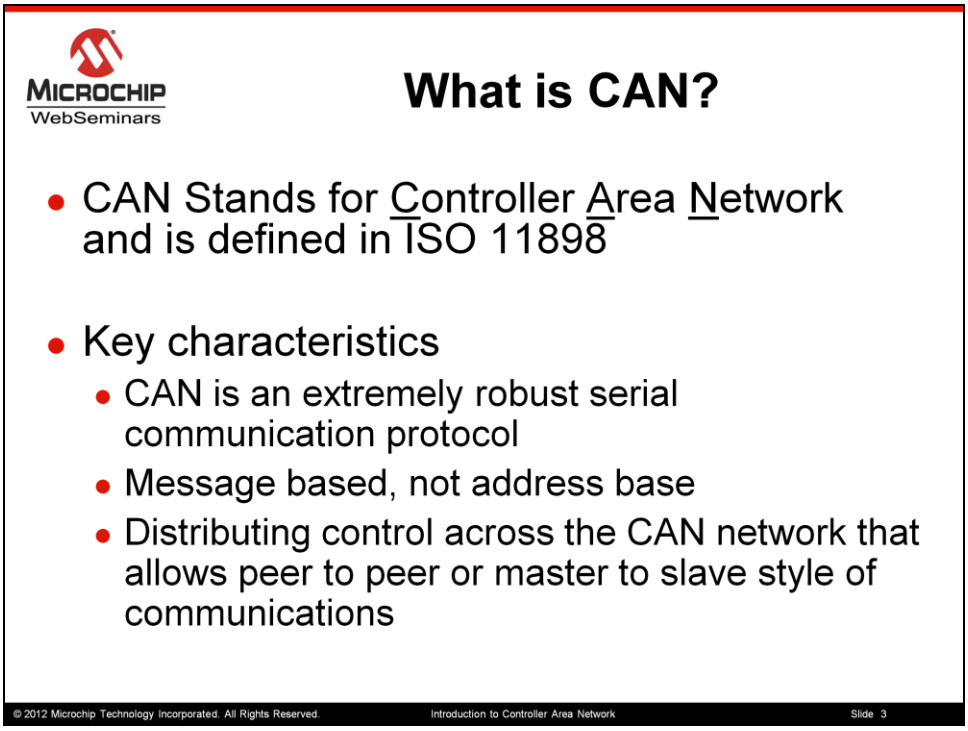
Today's agenda starts with defining what CAN is

I will then talk about the CAN Protocol

We will cover key characteristics, details about Message Structure, and Error Handling

next I will discuss in general terms how a PICmicro with the CAN model integrated transmits and receives CAN messages

Finally I will close out this Web seminar with the Summary and point you to some Additional Resources to help you continue your path onto learning more about CAN



So What is CAN?

CAN stands for "Controller Area Network" and is defined in the ISO 11898 specification

ISO 11898 is actually a family of specifications in which ISO11898-1 covers the datalink layer and ISO118980-2 and ISO118980-3 cover physical layers.

Just to name a few key characteristics about CAN

CAN is extremely Robust communication Protocol.

Here are three examples why

- Any CAN node on the BUS can detect errors in the message, and force the message to be destroyed and retransmitted… this feature helps to ensure that the message a node does receive contains valid data
- The CAN Frame requires that every node "acknowledge" the message before it can be processed by that node. This acknowledge can only come after various error condition checks; including a 15bit CRC on the message. If one CAN node finds an error with the message, the message is destroyed and retransmitted
- The specification defines three different error states for a CAN node to be in; with each error state giving the CAN node different levels of bus access. That was designed to limit faulty nodes from permanently taking down the CAN bus

We will cover these features in more details later throughout the web seminar

CAN is Serial communication in which All nodes on the CAN bus are attach to common connection using the same bitrate.

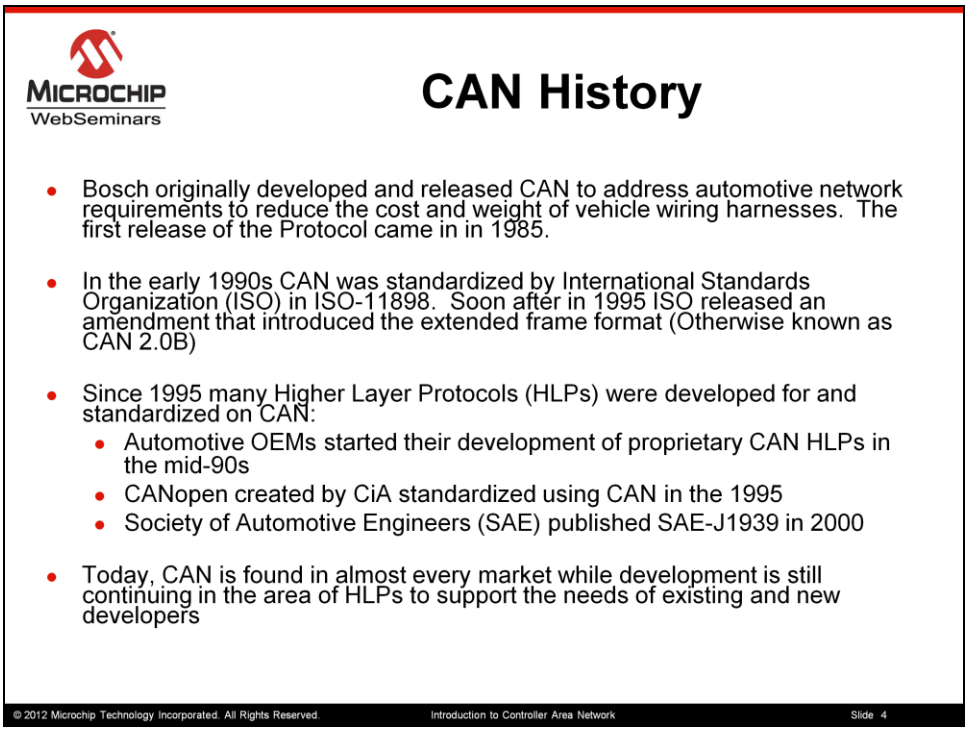
CAN is message based Not address based.

What this means is Messages are NOT transmitted from one node to another node based on the address of a CAN Node.

instead a CAN node will broadcast it's message to all nodes on the bus, and it is up to the receiving node to determine it should act on that message. Single or multiple nodes may act on the same data.

It is possible to add New nodes to a CAN bus without having to update all of nodes with addressing information

CAN allows for distributed control across a Network because of the reliability of the data. This allows designers the flexibility to setup Master/Slave or Peer/Peer also called Consumer producer/networks



To give you a little bit of history on CAN

Bosch originally developed CAN to address automotive network requirements to reduce the cost and weight of a vehicle wiring harness.

This protocol was first released 1985.

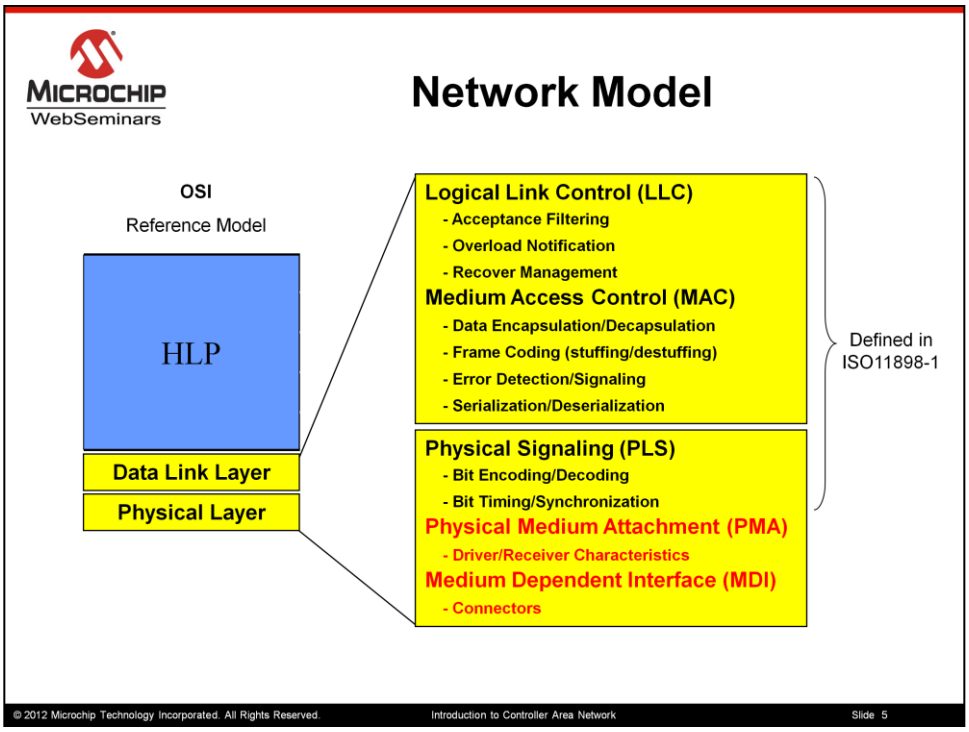
In the 1990s CAN was standardized by ISO 11898

Since then many Higher Layer Protocols (HLPs) were developed for and standardized on CAN:

Both proprietary and non-proprietary protocols exist

Today, CAN is found in almost every market while development is still continuing in the area of HLPs to support the needs of existing and new developers

CAN continues to be widely popular mainly because it is an extremely robust, but also because the sheer volumes in the automotive market have driven the costs of a CAN node down. This has allowed the protocol to grow rapidly since 1985 while expanding into other Markets like marine, industrial, and medical, Agricultural and construction, Military, Factory and Building automation.



Here I have shown the OSI 7 layer reference model.

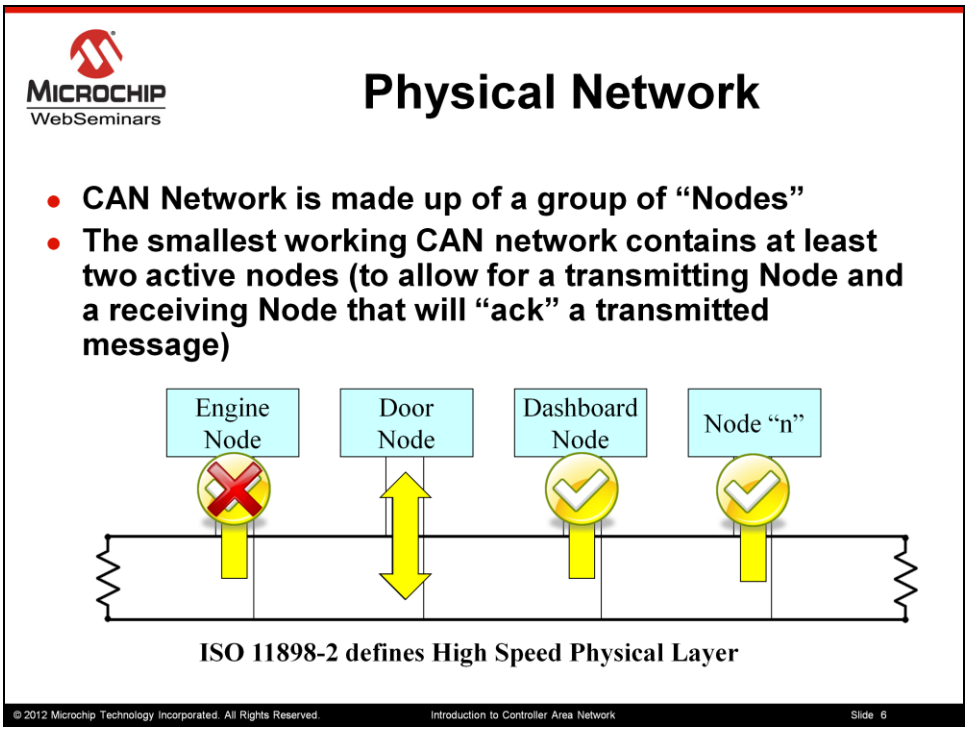
This model is a guideline for network developers that defines different levels that data should pass through to travel between two devices on the network

And if you were to map the CAN protocol onto this model, you would see that CAN would exist in the lower two layers (Datalink and part of the Physical layer)

And as for reference most HLPs would map to the upper 5 layers

The ISO 11898-1 doesn't cover HLPs.

It also doesn't specify what transceivers or connectors should be used for a CAN network



This slide shows a very simple CAN network. This example CAN network is made up of four nodes that are connected to a physical layer… which in this example is a differential bus with termination.

Even though we are using a differential bus for this example keep in mind that other possible physical layers exist.

ISO 11898-2 defines high speed physical. One important note is that Transmission speeds are defined up to 1 Mbps for total bus lengths of less than 40m

Now it is possible to lengthen this distance in the CAN BUS, by to do so the bitrate would have to be lowered for the entire CAN Network

In this Example the Door Node will transmit out a message that lets the other nodes on the bus know that the state of the door (closed or open).

While the door node is transmitting it is also receiving it’s transmitted message at the same time as the other nodes on the BUS. All nodes including the Door Node are checking that message for errors. At a very basic level ALL CAN messages are broadcast messages

If there are no errors with the message all nodes will “ack” the message by transmitting a dominant bit value onto the BUS while the Transmitting Node is listening.

This notifies the transmitter that at least one receiver validates the message.

But even though all nodes were required to check the message for errors it is still possible that one node (In this case the Engine Node) doesn’t care about the message and will discard it. The engine node has can implement two methods to discard this valid message. Both are implement during the design time of the Node.

The first method is to configure the hardware mask / filters to discard the valid message before the user application is ever notified.

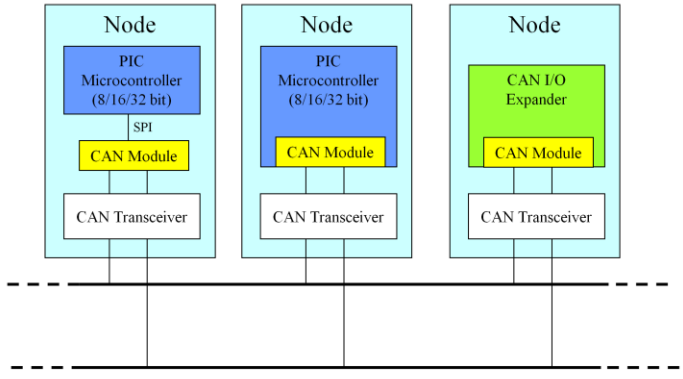
The second method is to notify the user application of the message and allow it to make the filtering decision

The first method is preferred when trying to prevent wasted CPU cycles, but there could be situations where it would make sense to have the application do the filtering.

MICROCHIP WebSeminars

Various types of "Nodes"

- Various types of "Nodes" can be found on a CAN Network



There are various types of "Nodes" that can be found on a CAN Network.

Here I am showing three

The node on the Left is an example of microcontroller that does not have an integrated CAN module, but the application requires that it still needs to connect to the CAN bus.

Typically you would see this with legacy applications that are now being requested by the Network designer to be on the BUS or other reason would be the application requires a certain microcontroller that DOES not come with an integrated CAN module.

In either case one of the simplest ways to this node to the CAN Bus is with standalone CAN module.

The next type of node you will find on a CAN bus is in the center, and this is an example of a node that has a microcontroller with an integrated CAN module (Which allows for a faster internal connection between the CAN module and microcontroller). This is the most common node you would see on the CAN bus. Microchip currently provides 8/16/32 bit PIC micros with the CAN module integrated.

The third example of a CAN node on the right shows a node with a CAN I/O expander

This integrate circuit provides I/O expansion for a CAN network without a microcontroller

This device would have peripheral like GPIO, A2D, or PWM

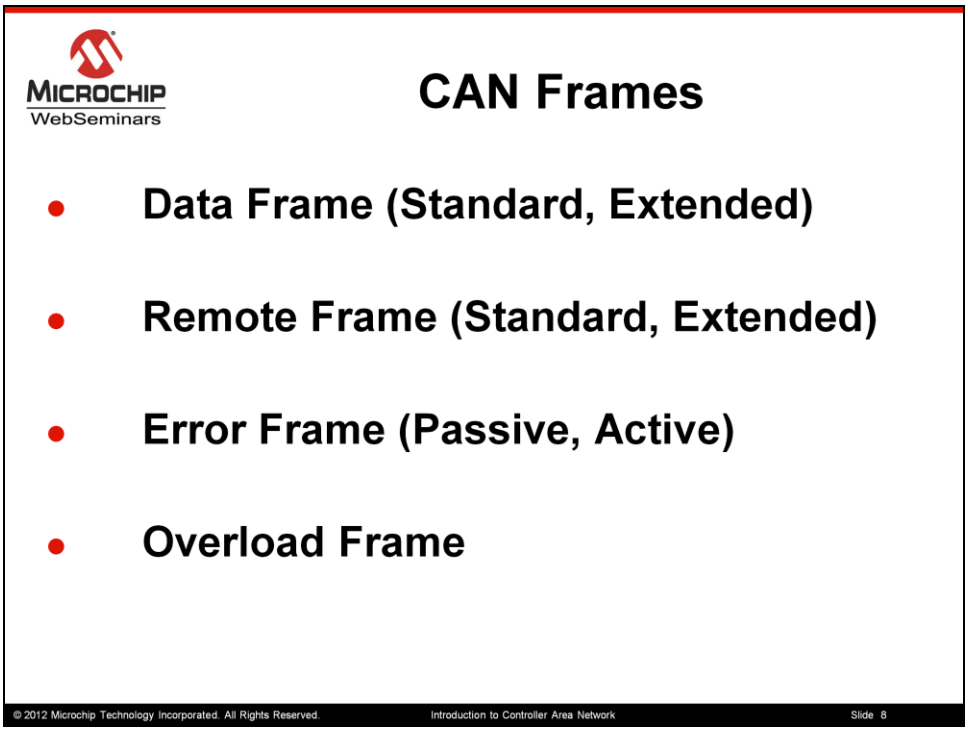
The Node can be configure with defaults to send out messages periodic or event driven on thresholds

Also other nodes can interact to read I/O or change Outputs

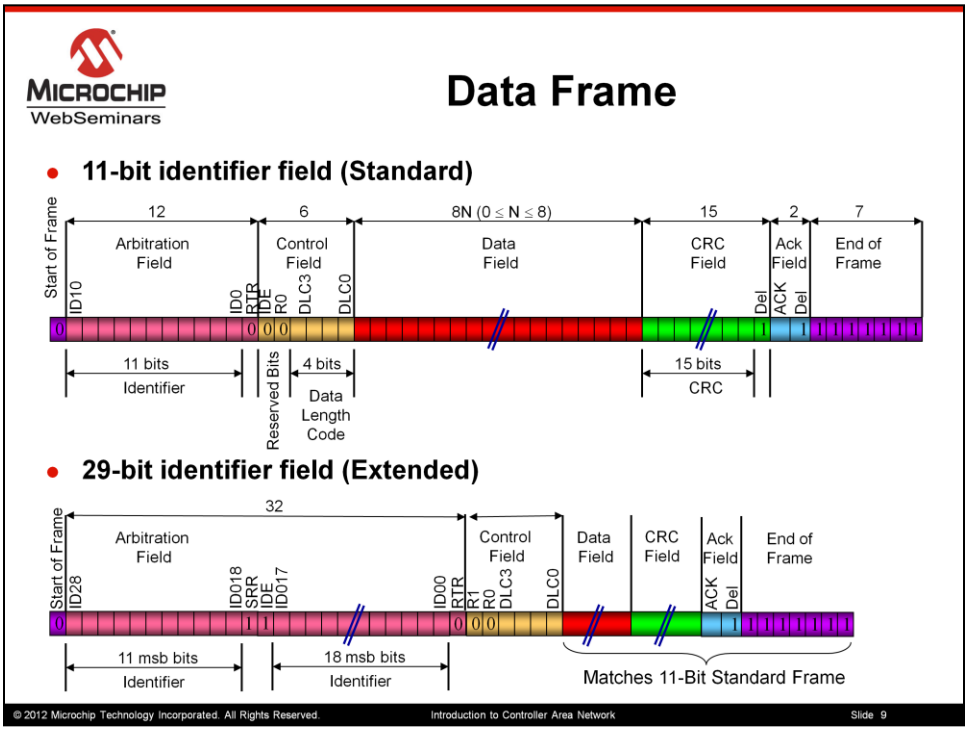
This node requires No microcontroller firmware to write or debug, but then lacks some flexibility with non proprietary High Layer Protocols

This device was designed for low cost simple sensor applications.

As you can see CAN nodes can execute very simple or complex applications



Now we will discuss what the various CAN frames exist in the ISO-11898 specification



Now lets take a look at the data frame

As you can see there are two formats to this frame; Standard and extended.

The major differences between the two formats is the Arbitration field.

The standard frame has an 11 bit id in the arbitration field that allows up to 2048 unique messages onto the BUS

And the extended frame has a 29 bit id that allows up over 536 million unique messages onto the BUS

Both standard and extended data frames can coexist on the same CAN BUS. The standard 11 bit data frame will always have priority over the same extended 29 bit frame with identical 11bit base id.

It is recommend that no two nodes transmit the same data frame in terms of ID… because this would break the arbitration process.

So lets take a closer look at the data frame, we will use the standard for this deep dive

The data frame starts with a Start of Frame which is a dominant bit…

Next comes the ID field… this is where the arbitration occurs and the node with the highest priority id (Lowest ID Number) wins the BUS. So in other a Node transmitting out a data frame with ID 0 will always "win" access to the BUS a nondestructive bitwise arbitration method is use to allow multiple nodes to contend for the BUS. This process allows the highest priority message to remain intact even though there were collisions detected. And this process takes place without corruption or delay of the highest priority message.

Which is one reason why CAN works great in safety critical applications. I will discuss arbitration more later.

The node that wins arb will continue to transmit while all other transmitters will stop transmitting and revert to receiving nodes. Every node on the bus is responsible to check the message for errors that range from bit stuffing to CRCs error.

So the transmitting Node will transmit out the control field that contains the IDE bit and data length code usually shorted to DLC. The IDE bit lets the receivers know if the message is standard or extended, while the DLC lets the receivers know if the message has between 0 and 8 data bytes.

Next the transmitting node will transmit out the data, then the CRC.

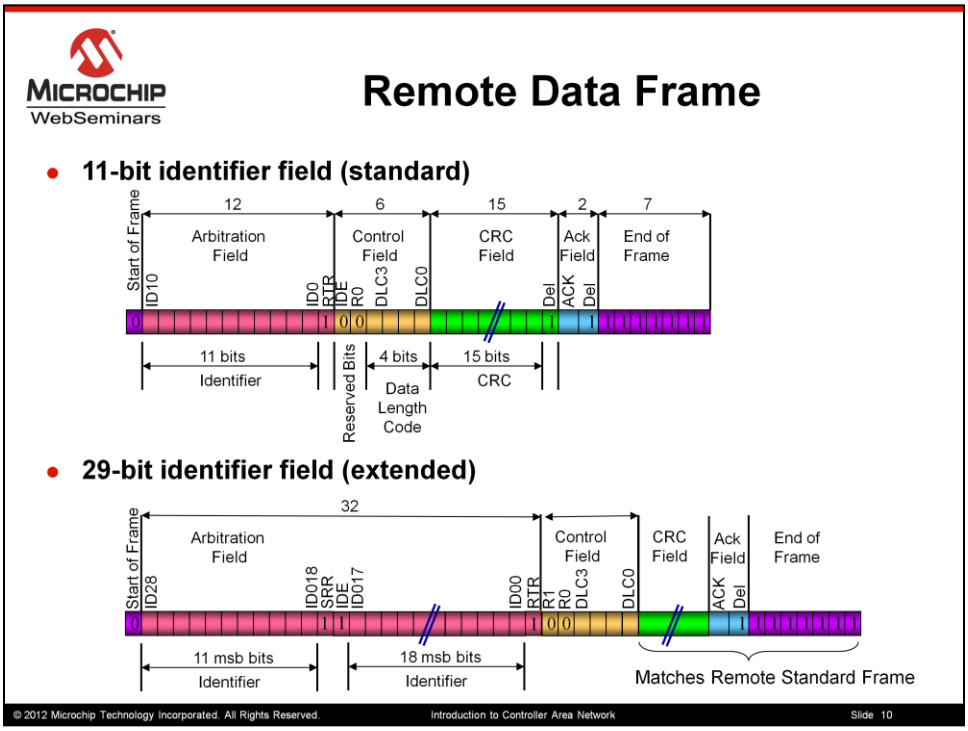
The CRC is calculated from the SOF to the Data Field

At the ack field the transmitter transmits a recessive bit on the bus and listens for a receiver to transmit a dominant. If up to that point there are no errors with this Data frame all receivers should transmit a dominant. This lets the transmitting node know that the receiving node received the message ok

At the End of frame and three bits of quiet time; also called inter-frame space; all nodes including that node that just transmitted will have a chance to transmit again.

Also note

The ID of the message determines the priority and gives context to the information in the data field



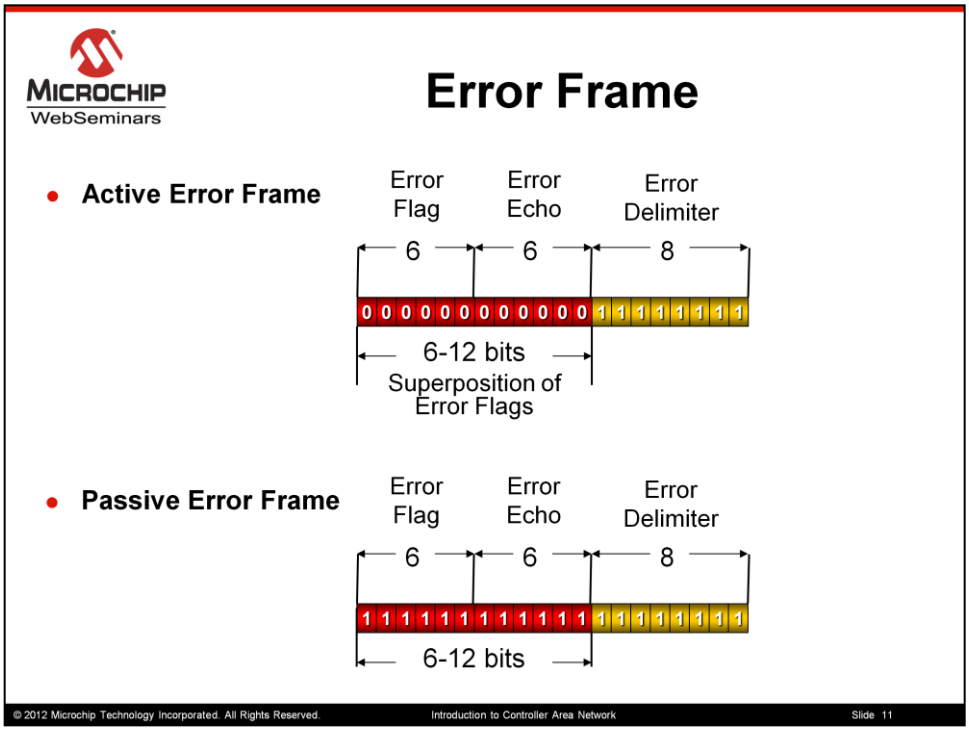
Remote Frames can come in either standard or extended formats and contain No data payload.

Remote frames are used when one node needs to request data from another node.

So for example Node A will transmit a remote frame request onto the BUS with an Identifier that matches a data frame that typically sent by Node B.

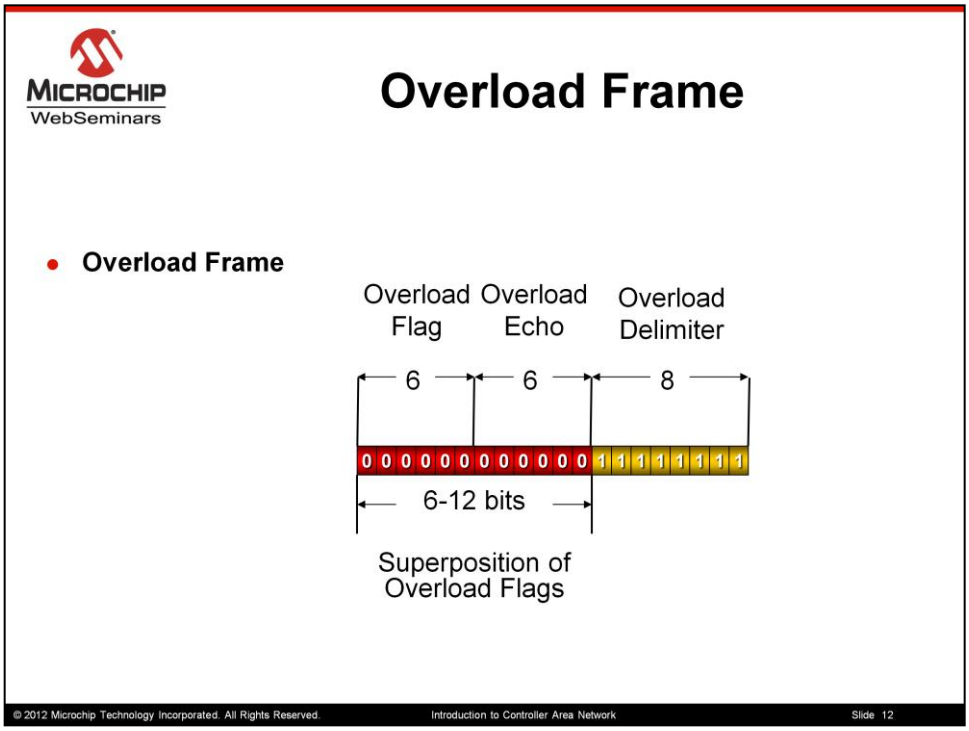
Upon seeing the RTR request Node B will transmit the data frame matching ID and data

Remote frames are not typically used in Non propriety HLP protocols, but can be found in some propriety protocols



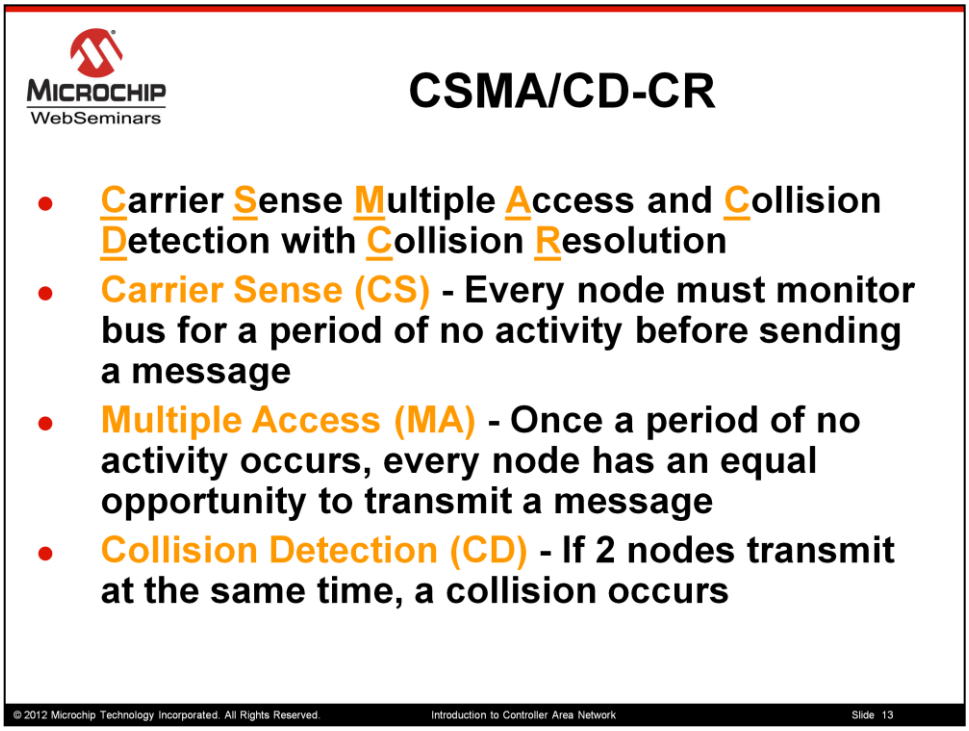
Error frames are transmitted by any nodes when they detect an error with the CAN Data or Remote Frame. If the node is in the error active state it will transmit out an active error frame. If the node is in the passive error state it will transmit out a passive error frame. The error frame can vary in length because it is possible for some noise localized next to a CAN node on the bus will flip a bit in data frame. If this occurs that node might see the bit flip while other nodes on the bus will not. Because of this "bit flip" the CRC calculation should be wrong, and it is possible that the bit stuffing rule may be violated. If that is the case that node will transmit out an error frame, and soon after the other nodes on the bus will start to see the bit stuffing rule being violated and they too will transmit out echoing error flags.

These error frames will "destroy" the current data or remote frame on the bus. The transmitting node will know that it's message wasn't received properly by all nodes, and will automatically attempt a retransmission at the next available quiet time on the bus



An overload frame is special version of the Error Frame that will not cause a retransmission of the last "destroyed" message. Instead it is used by a Node to request a delay between data or remote frames during the "inter-frame" space or quiet time on the CAN bus.

If a node needs more time to process the message at the protocol level it has the ability to transmit out up to two Overload frames before the next data or remote is present on the bus. What this does is request all nodes on the network to delay sending out the next Data or Remote Frames thus given that node more time to process the current message.



Here we can see that CAN employs the following Carrier Sense Multiple Access and Collision Detection with Collision Resolution

Carrier Sense is defined as

Every node must monitor bus for a period of no activity before sending a message

This period of no activity is called “inter-frame” space

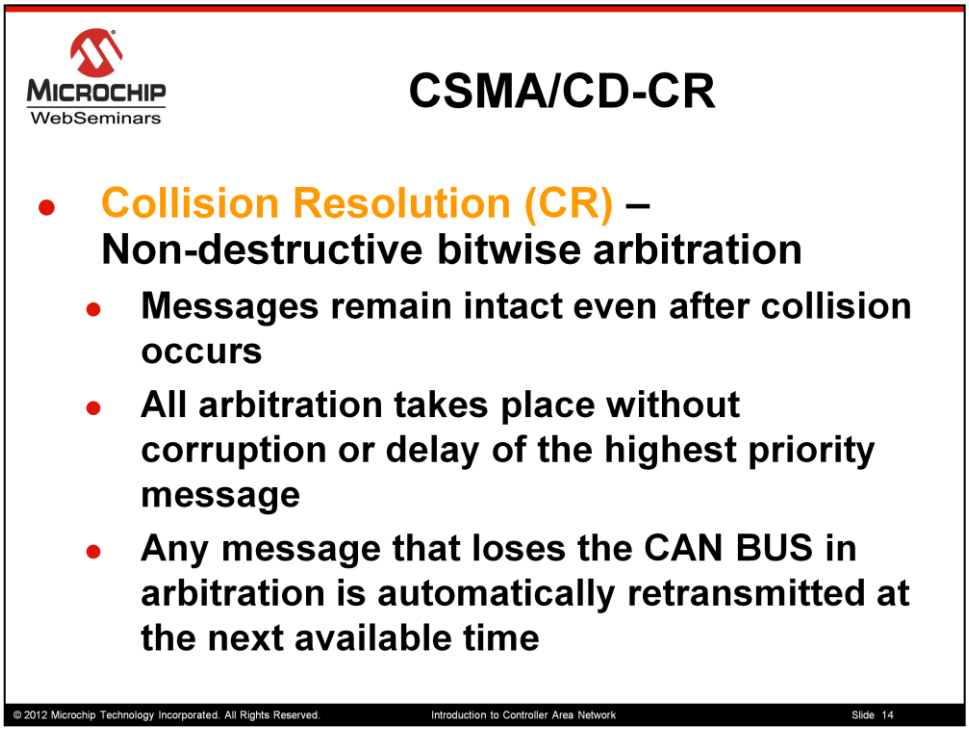
Multiple Access is defined as

Once a period of no activity occurs, every node has an equal opportunity to transmit a message

This starts the arbitration period at the SOF

Collision Detection is defined as

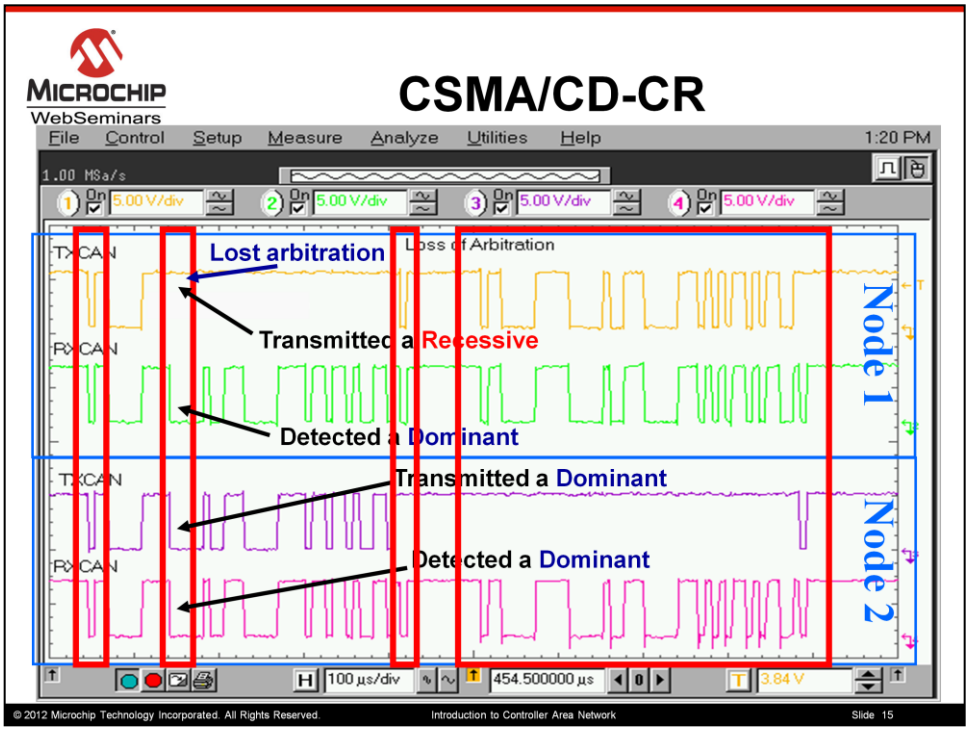
If 2 nodes transmit at the same time, a collision occurs and is detected by the node transmitting a recessive bit



Collision resolution refers to “non destructive bitwise arbitration”

To implement CSMA CDCR

The physical layer needs to support Dominant and recessive bit states; in which dominant bits wins arbitration over recessive bits



Here we show an example of two nodes attempting to transmit a message at the same time

This screen shot shows the communication between the transceiver and the CAN module which is going to be digital.

The two signals on top of the screen shot belong to Node 1 and they are the TX and RX digital lines between the transceiver and microcontroller

And the two signals on bottom are the digital connections between Node 2's transceiver and microcontroller

So starting at the left of the screen shot we can see that both nodes started transmitting a 11bit CAN frame with the dominant SOF.

And if you can recall the next portion of the CAN frame is the arbitration field in which the nodes will transmit out the ID for the data frame.

Both nodes will continue to transmit until there is a mismatch.

We can see that Node 1 transmits a 1 (which is a recessive) while Node 2 transmits a 0 (which is a dominant).

At this point Node 2 "wins" arb and gains access to be the sole transmitter on the CAN bus while Node 1; the losing node stops transmitting and becomes a receiver.

Node 2 continue transmitting out the rest of the ID, Ccontrol field, Data (If there is any) and then the CRC field. At the ACK field Node 2 will transmit out a recessive,

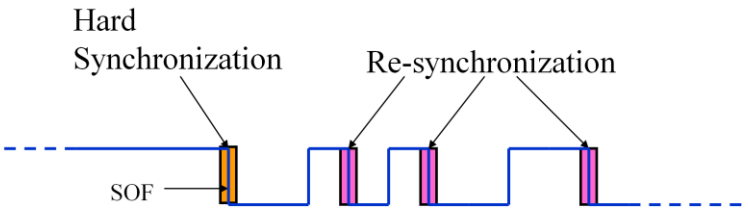
and we can see that Node 1 transmits out a dominant which means that the CAN frame contained no errors.

After the end of frame and a required 3 bit interframe space wait; Node 1 will transmit out its pending message while Node 2 listens

MICROCHIP
WebSeminars

Synchronization

- No clock in bit stream
- Receivers synchronize on recessive to dominant transitions
  - Hard Synchronization occurs at SOF and resets bit clock
  - Resynchronization occurs at recessive-to-dominant (1-to-0) edges and adjusts the bit clock as necessary

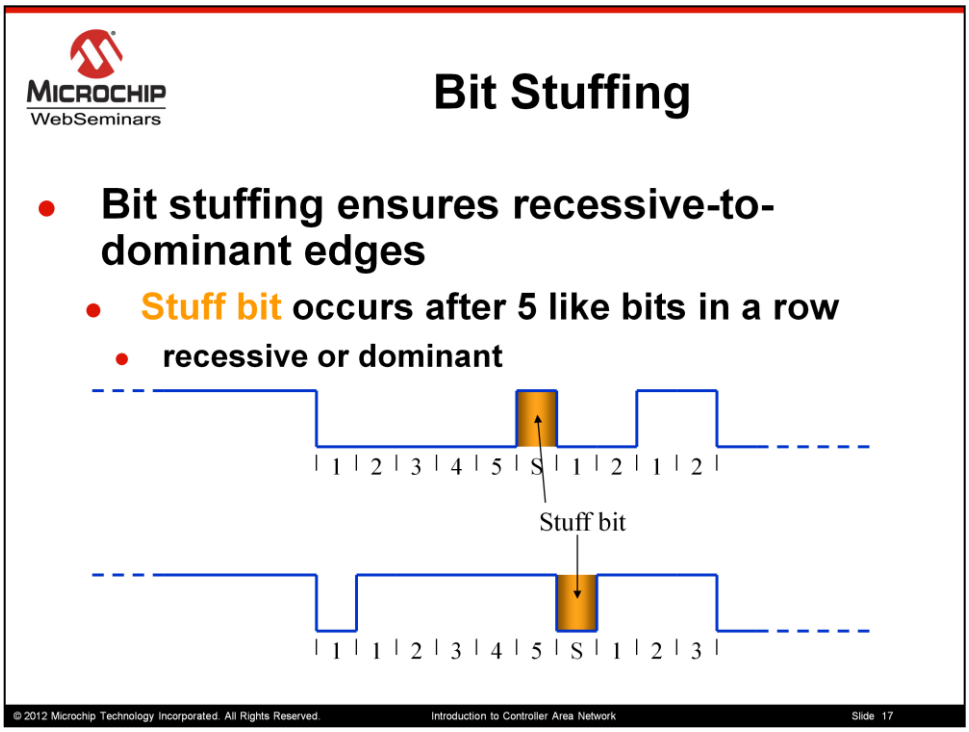


Each node has it own oscillator and internal CAN clock, so the question might arise, how is it possible that many nodes across a network can stay in Synchronizations without a clock line

This is achieved by the receivers synchronizing on recessive to dominant edges

CAN implements NON return to Zero (NRZ) on the physical bus signal. Which means there will be no edge between two like bits

So the next question is what happens if a node transmits all 0s for data… how will the nodes stay in sync with no edges?



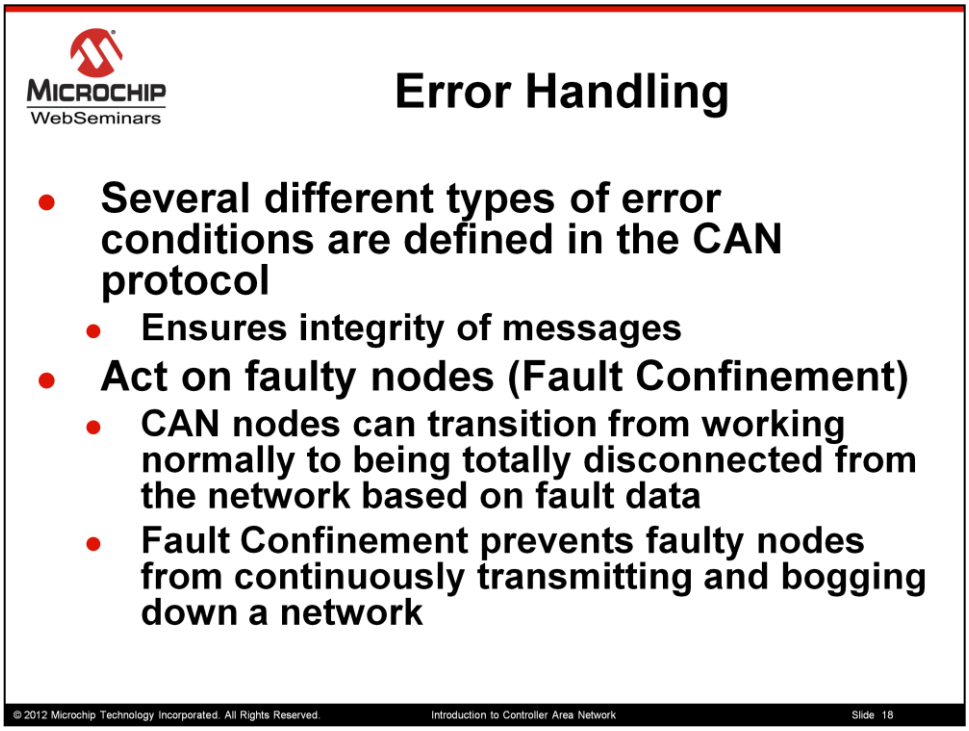
Nodes stay in sync with a rule called “Bit stuffing”.

“Bit stuffing” is introduced at the protocol level to ensure there are enough edges in a CAN frame to maintain Synchronization in the network

Bit stuffing rule simple states that after 5 like bits the protocol handler should add an additional bit of opposite polarity to the CAN frame to force an edge.

This stuffed bit is added by the CAN node transmitting out the message at the protocol level, and is expected and removed by the receivers at the protocol level so the user application is never aware of any stuffed bits

This bit stuffing rule Implies that the maximum time between synchronization edges is 10 bits



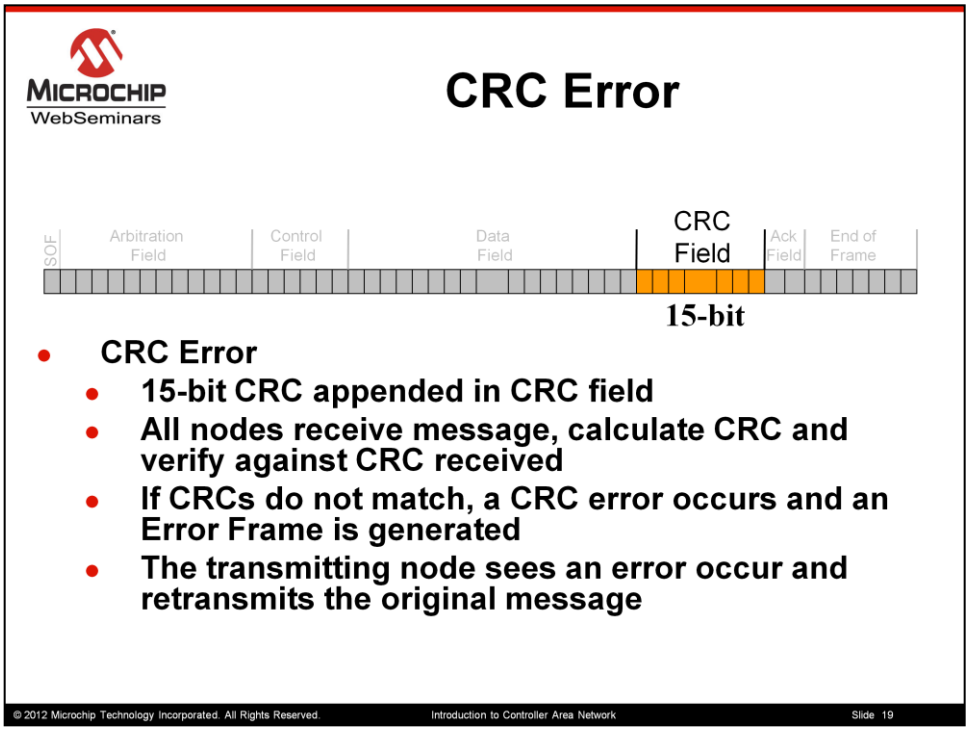
Now lets discuss error handling

For the CAN nodes to perform any sort of error handling they have to be able to detect error conditions. There are several different types of error conditions define in the CAN protocol that we will talk about shortly

These error conditions cause the CAN node to transmit out error frames, which in turn increment internal receive or transmit counters.

By using these counters the node is able to recognize if it is the source of the BUS problems and subsequently disconnect itself from the CAN Bus.

This action prevents a single node from commandeering the CAN bus with error frames thus preventing any valid data frames onto the bus.



So now lets discuss error conditions

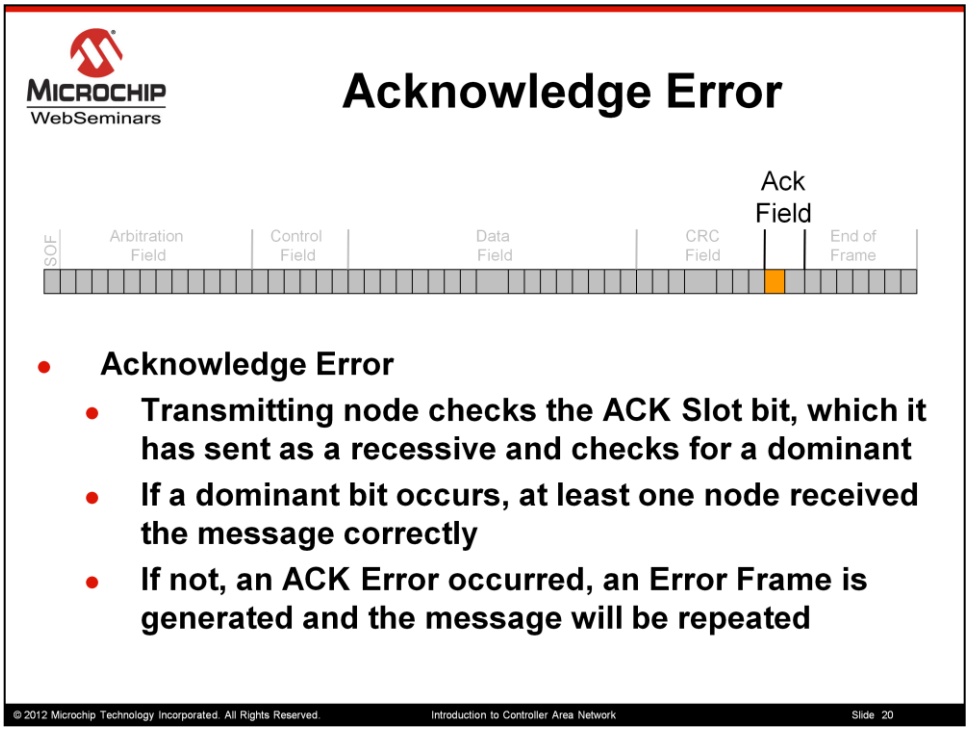
The first we will talk about is the CRC error

Recall that all receiving nodes need to calculate the CRC and verify it against the CRC received in the message

If CRCs do not match, a CRC error occurs and an Error Frame is generated

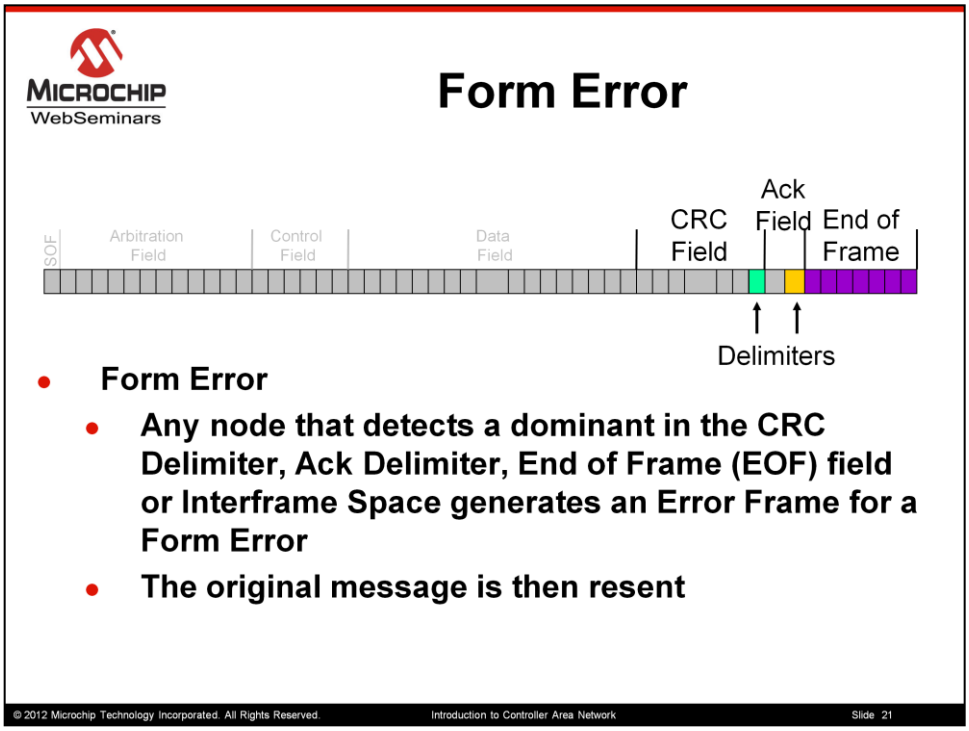
This causes the message to be destroyed and prevents any nodes from using that data

The transmitting node sees the error frame … and will then retransmit the original message at the next available time



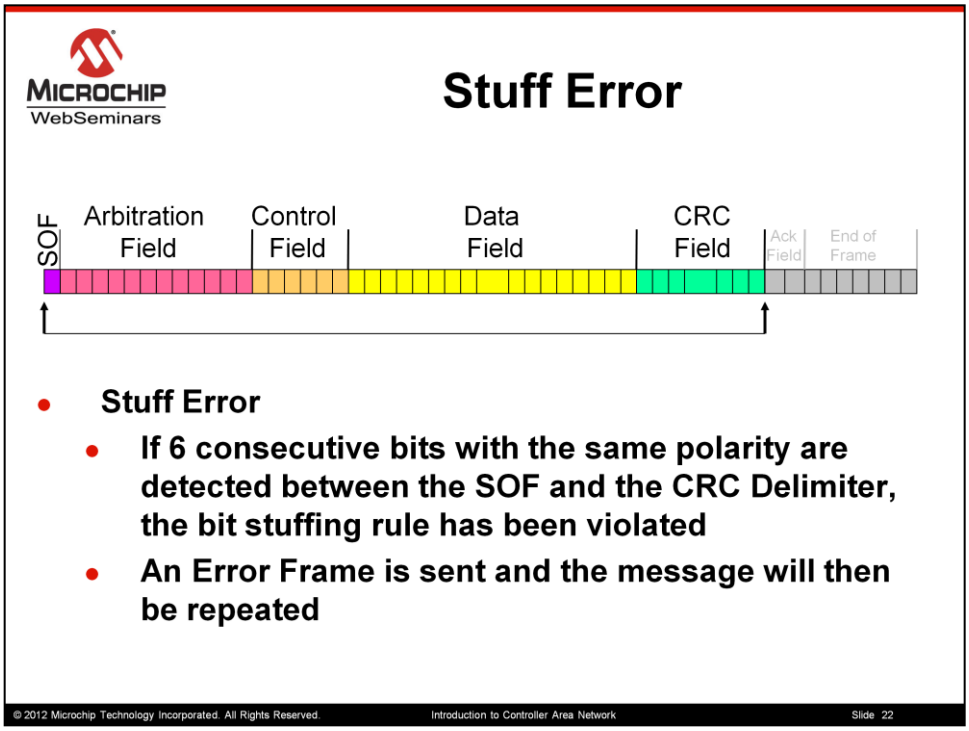
The Acknowledge error occurs when the transmitter of the CAN message detects a recessive bit in the ACK field.

The transmitter will then transmit out an error frame to destroy the message it just transmitted and will then retransmit the original message at the next available time



The form error occurs when the “recessive” slots in a CAN message become “dominant”

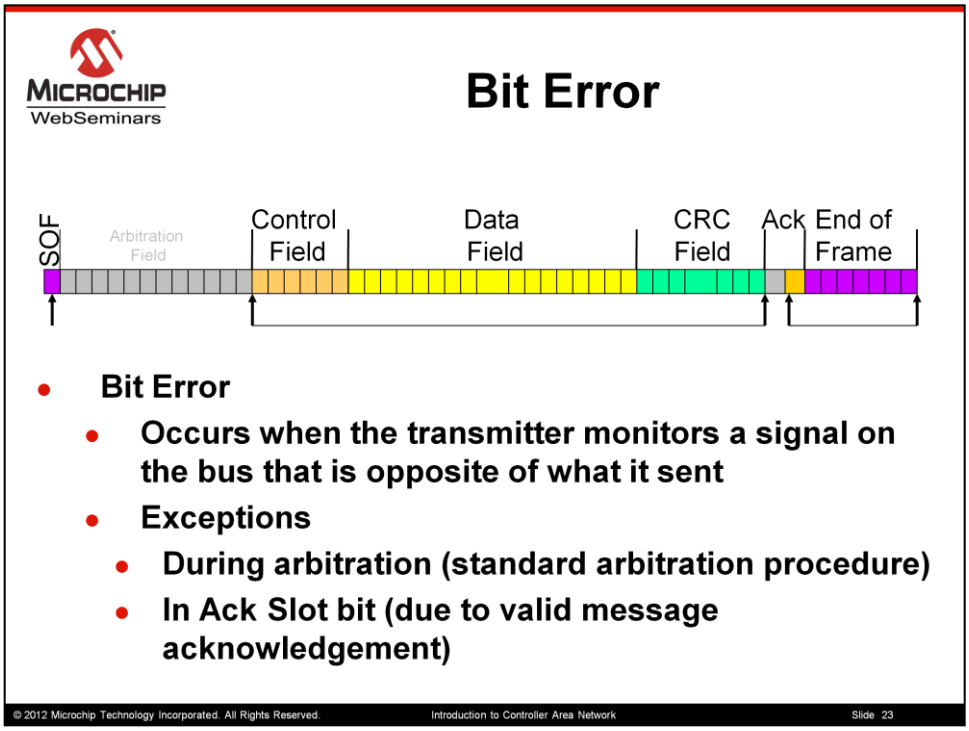
Any node detecting this will need to transmit out an error frame, thus destroying the current message, and the transmitter will need to retransmit at the next available time



The Stuff error occurs when any node detects 6 consecutive bit of the same polarity between the SOF and end of CRC field.

When the error is detected the node will transmit out an error frame, thus destroying the current message, and then the original message should be repeated by the transmitter at the next available time

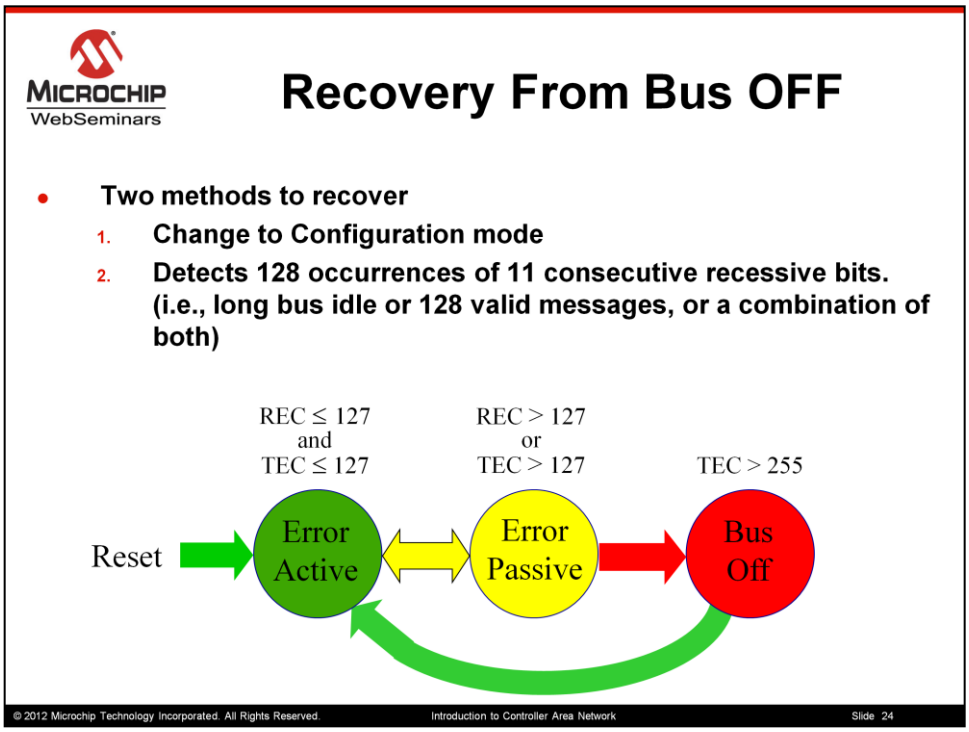
Keep in mind that error frames intentionally violate the bit stuffing rule.



The Bit error occurs when the transmitter monitors a signal on the bus different from what it sent.

This error is NOT checked during the ARB or the ACK field, that is because during those times all nodes have equal access to the bus.

When the transmit detects this error it will transmit out an error frame, thus destroying the current message, and then the transmitter will need to retransmit that original message at the next available time



Keep in mind that all of those error conditions we just discussed generate error frames, and these error frames increment internal transmit and receive Error counters.

And in some scenarios these counters can increment at different rates. Refer to the ISO11898 specification to learn more details.

Every CAN node will use the following state diagram when connected to the CAN BUS.

Each node starts off in Error active state… This is the state a normal CAN node should be in for normal bus activity

In this state the CAN node is able to transmit normal data , remote, overload and active error frames.

And when either the receive or transmit error counters increment to greater then 127 the node will enter the Error Passive State

In this state the CAN node can still transmit out data , remote, and overload frames. But now the node can only transmit passive error frames.

Recall the passive error frame consists of all ones, so if the node is still detecting and transmitting error frames … these error frames will be passive and should not cause the other nodes on the network to increment their receive counters

Once in Error Passive state the Node still have a chance to enter back into error active… this will happen when the error counters drop below 127 and below. The error counters will decrement every time there is a valid message on the bus

If the node continues to increment the Transmit error counter to greater then 255 it will enter bus off.

At this point the CAN module disconnects from the bus. In order words, other then sensing the bus levels it will not transmit or receive any message or error frames

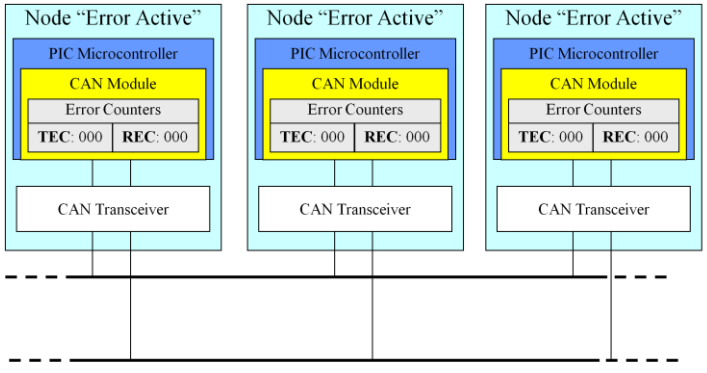
Once in the Bus off state the node can only enter Error Active. There are two methods to recover

The CAN node can change to configuration then back to Normal Bus mode… which clears out the transmit and receiver error counters

MICROCHIP WebSeminars

Fault Confinement

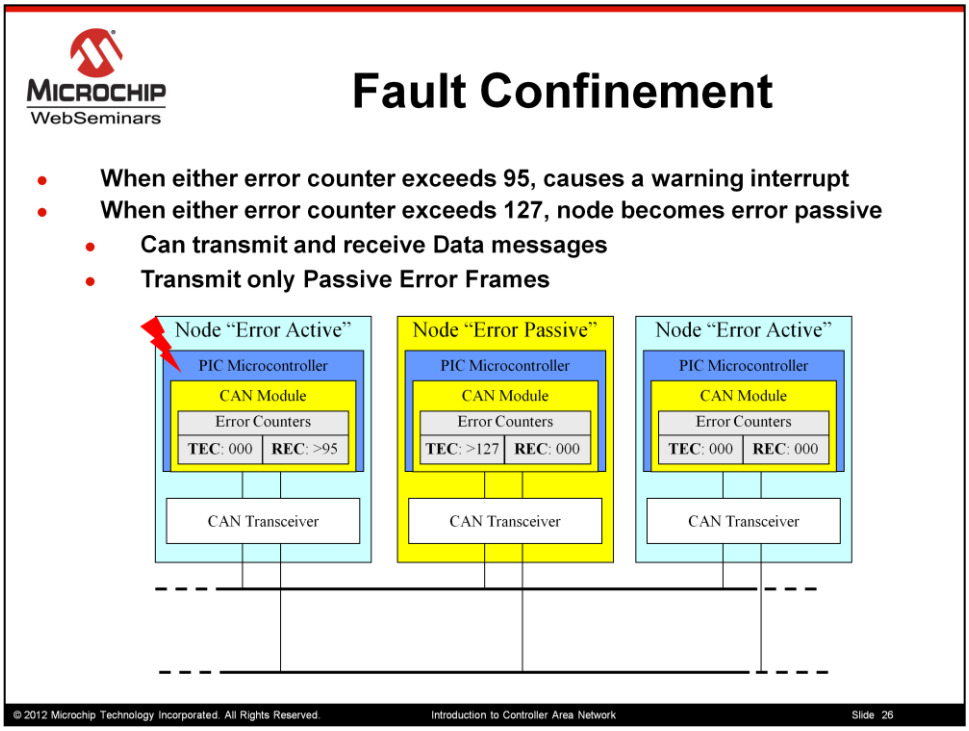
- Three Error States defined by CAN as a function of the error counters: Error Active, Error Passive, and Bus-Off
- Error Active is normal mode
  - Allowed to send messages and Active Error Frames



Now lets walk through a Fault confinement example.

In this example I have three nodes, that are all in the "Error Active" state which means they are able to interact with the bus normally

By transmitting data / remote / or error active frames

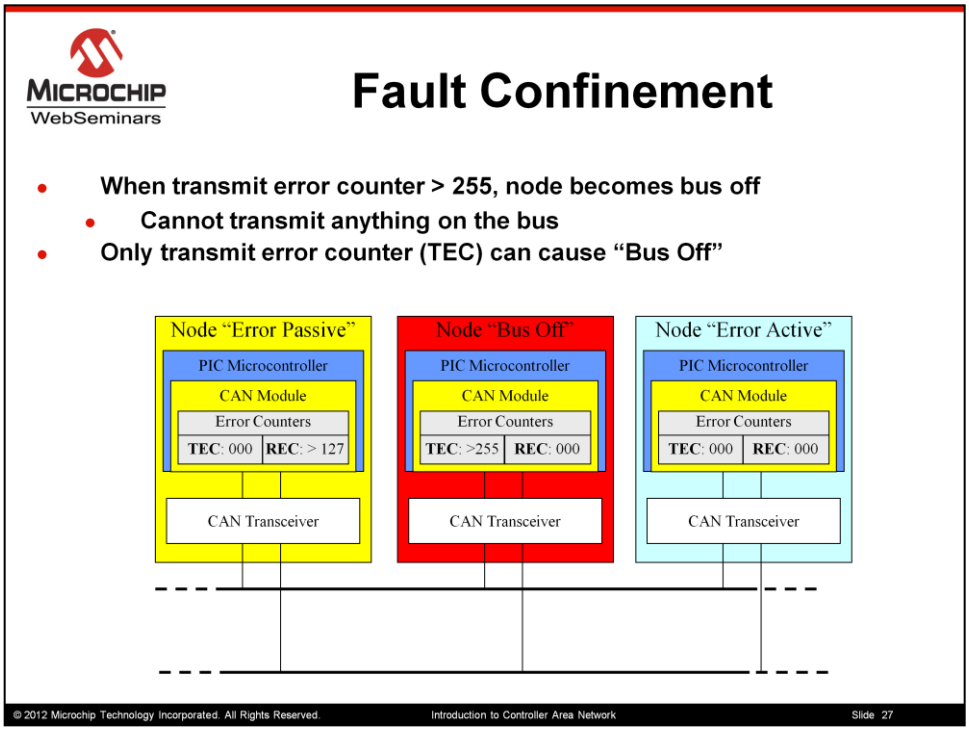


At some point in the future it possible for some node to detect error conditions and start transmitting out error frames; either there is noise in the environment, there is a short, or the node has become faulty in some way.

PICmicrocontrollers with the CAN module come with a internal "error counter warning" interrupt that will trigger when either error counters exceed 95. These feature gives the CAN node an early warning that the bus might be experiencing an issue.

We can see that the node on the left has received this "error counter warning interrupt".

Where as the node in the middle has entered into Error Passive state because it's transmit error counter has become greater then 127. At this point the node in the middle can still transmit data and remote frames, but is unable to transmit error active frames. Instead it will need to transmit error passive frames to limit it's disturbance of the CAN BUS



Now we can see that the Node on the left has entered Error Passive, and the node in the middle has entered "Bus Off"

The node in the middle now has removed it's self from the BUS and will no longer be able to transmit out CAN data or remote frames. Also this "Bus off" node can not transmit Active or Passive error frames.

By allowing this Node to drop off the BUS… the CAN protocol has effectively guaranteed that bandwidth will always be available for

critical messages to be transmitted by other nodes on the BUS.

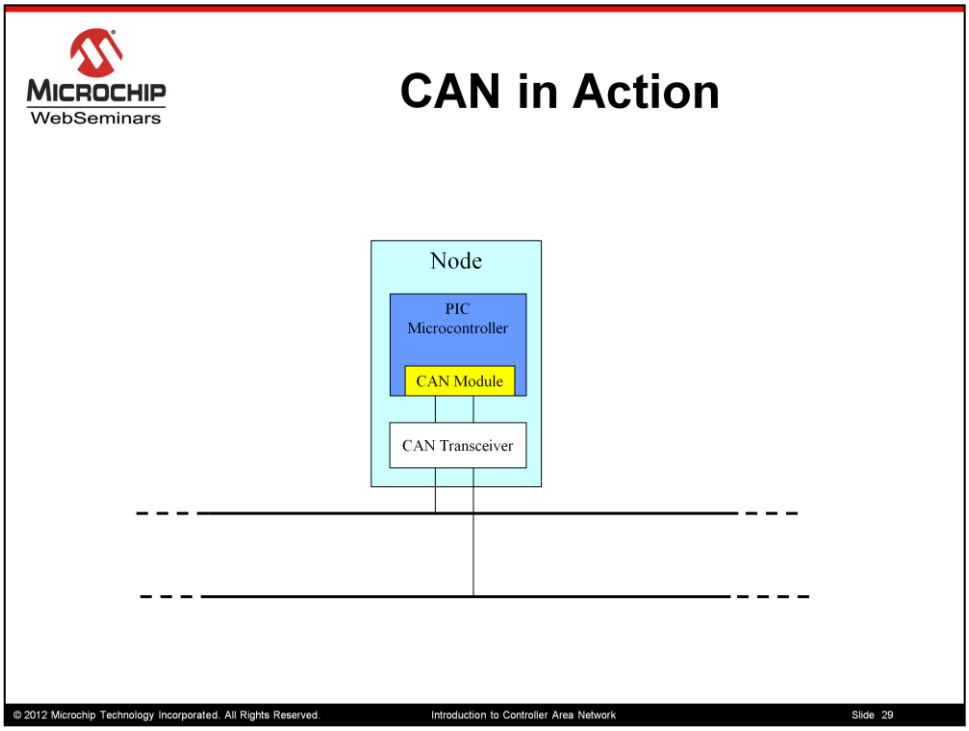
Higher Layer Protocol that implements Network Management / Diagnostics should be able to notified the user of such a condition to allow them the chance to service their product

Now you might be asking the question about why the node on the right didn't increment it's error counters the entire time through this example. There could be several legitimate reasons why; Maybe this node has been power cycled every time we decide to get a snapshot of the BUS, or ??? Maybe the node is in a "listen Only State" in which it only receives and doesn't transmit anything on the BUS

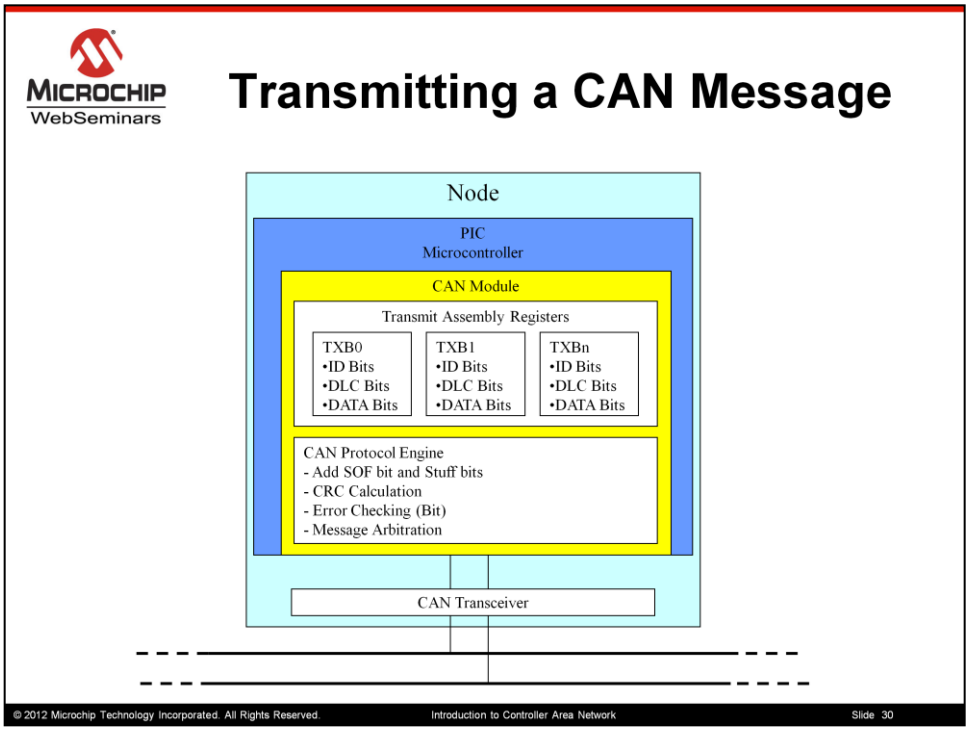
MICROCHIP
WebSeminars

Class Agenda

- What is CAN?
- CAN Protocol Overview
  - Key characteristics of CAN
  - Message Frames
  - Error Handling
- CAN in Action
  - Transmitting a CAN message
  - Receiving a CAN message
- Summary
- References



For the next couple of slides we will be using the Node with the CAN Module built into the PIC microcontroller as reference



Lets talk about the steps a CAN Node will need to go through to transmit out a CAN message

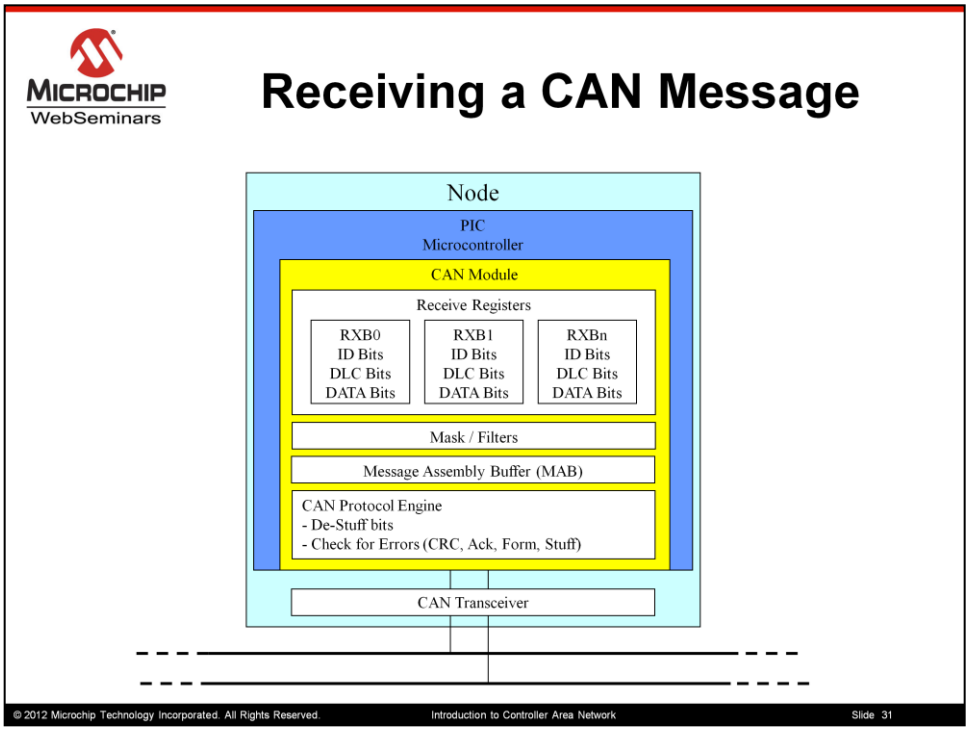
First the user software will need to confirm the can module is in configuration mode.
Then initialize the CAN module bitrate settings so that the module is using the same bitrate as all other nodes on the bus
Then the user should place the CAN module from configuration mode to normal mode

Second The user software will need load the TXB with the Message ID, DLC, and Data bit, now the user software will "set" the "transmit request bit"
This will lock that TX buffer from being written to by the user software…

At this point the CAN module takes over.
The message is assembled and stuff bits are added where needed
Next the CAN module senses the CAN bus looking for the next "quiet time

When there is quiet time the module will start transmitting the message while checking for error frames.
The CAN transceiver will translate the digital signals from the PICmicro to the transceiver to differential signals to go out onto the BUS
Actually the Physical layer bus signal depends on what transceiver is being used… differential was just mentioned because it is common.

If the CAN message is successfully sent onto the CAN bus… meaning there were NO error frames that forced a retransmit… the TX Buffer will unlocks for the next user software writes.



Lets talk about the steps a CAN Node will need to go through to receive CAN message from the CAN bus

First the user software will need to confirm the can module is in configuration mode.

then user software will need to initialize the CAN modules bitrate settings so that the module is using the same bitrate as all other nodes on the bus

The user software should also initialize the mask and filter settings as well.

Then the user should place the CAN module from configuration mode to normal mode

Next the CAN module will wait until another node on the CAN Bus transmits out a CAN message

it senses a CAN message on the BUS starting with the SOF

The CAN transceiver will translate the differential signal from the bus into digital signal and pass it along to the CAN module on the Microcontroller

The CAN module will de-stuffed the stuff bits and check for message Errors while it is loading the message into the MAB

If there are no Errors and the entire message has been loaded the message in the MAB the CAN module will check the received messages ID against the Mask and Filter settings

If the message ID is a match then the message will then get stored into the Receiver Buffer and at that point the PICmicrocontroller (ie the User's application) is notified

If the message DOES Not match the Mask and Filters the PICmicrocontroller would never get notified in other words precious microcontroller cycles would not be wasted on unwanted messages

Which is why it is important to setup and use the Mask and Filter settings correctly.

DLC = Data Length Code

MICROCHIP
WebSeminars

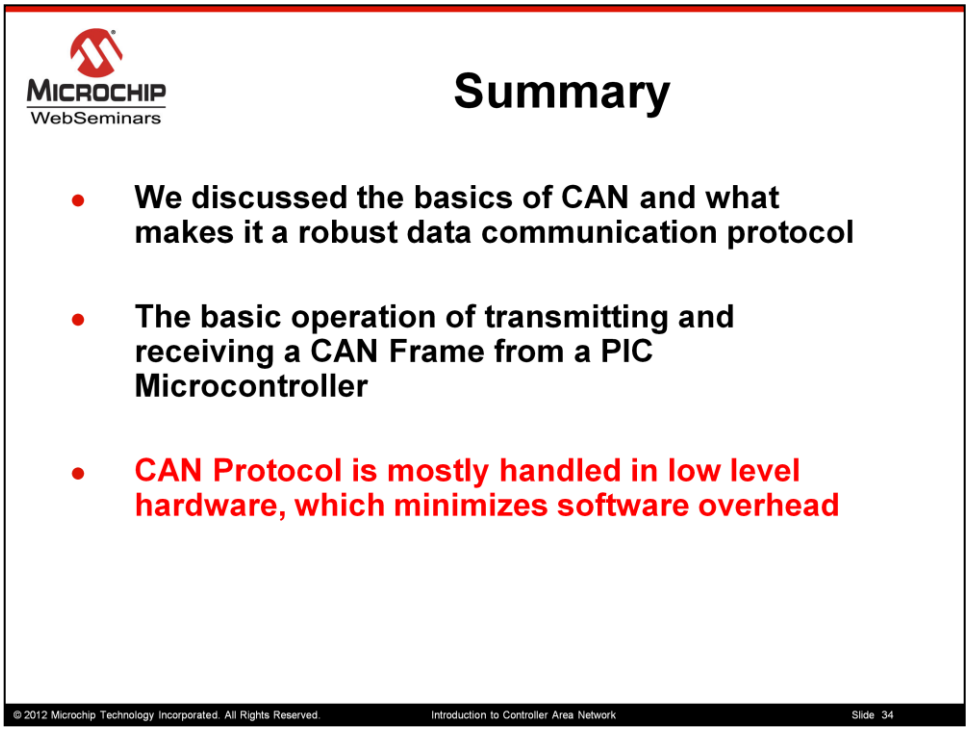
Transmitting and Receiving Is not as difficult as it looks!

- Most aspects of transmitting and receiving a CAN message is handled by the protocol engine in the CAN module
- Transmitting a message typically requires loading the Identifier, DLC, and Data, and then setting the "Transmit Request" bit.
- Receiving a message typically requires monitoring the "Receive Buffer Full" bit with polling or using interrupts to notify user application when a CAN message has been received

MICROCHIP
WebSeminars

Class Agenda

- CAN Protocol Overview
  - What is CAN?
  - Key characteristics of CAN
  - Message Frames
  - Error Handling
- CAN in Action
  - Transmitting a CAN message
  - Receiving a CAN message
- Summary
- References



So in summary

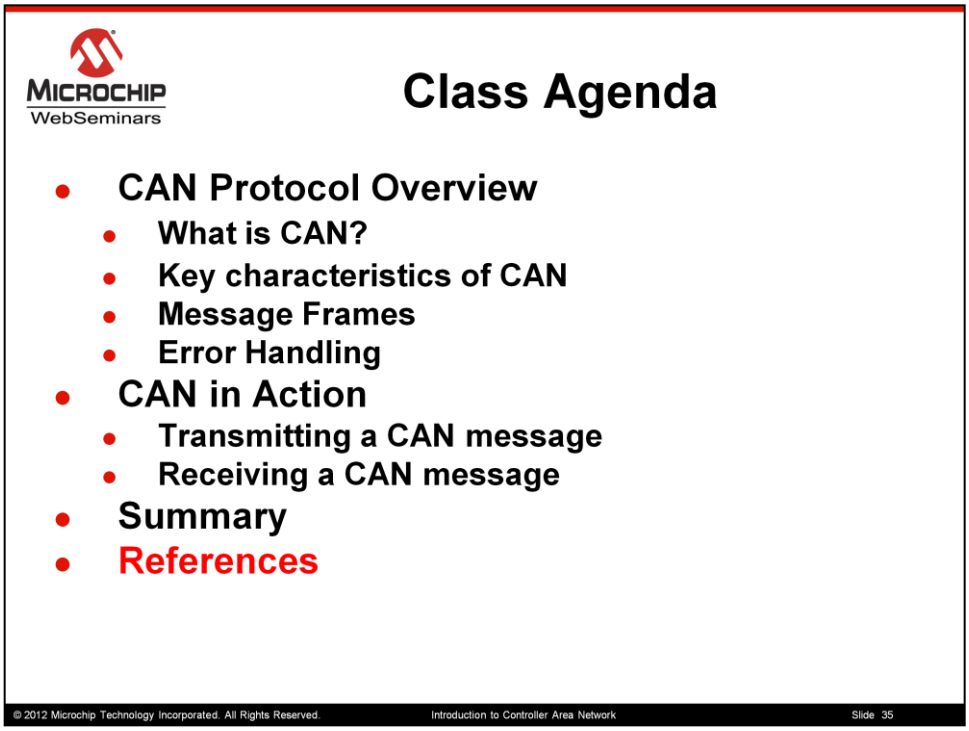
We discussed the basics of the CAN and what makes it a robust data communication protocol by covering topics like general protocol overview, message structure, arbitration, error detection and recovery

There is still more to learn about CAN that we didn't cover here in this web seminar. During our short time we didn't have the chance to cover bit timing, physical layers, or higher layer protocols in any detail

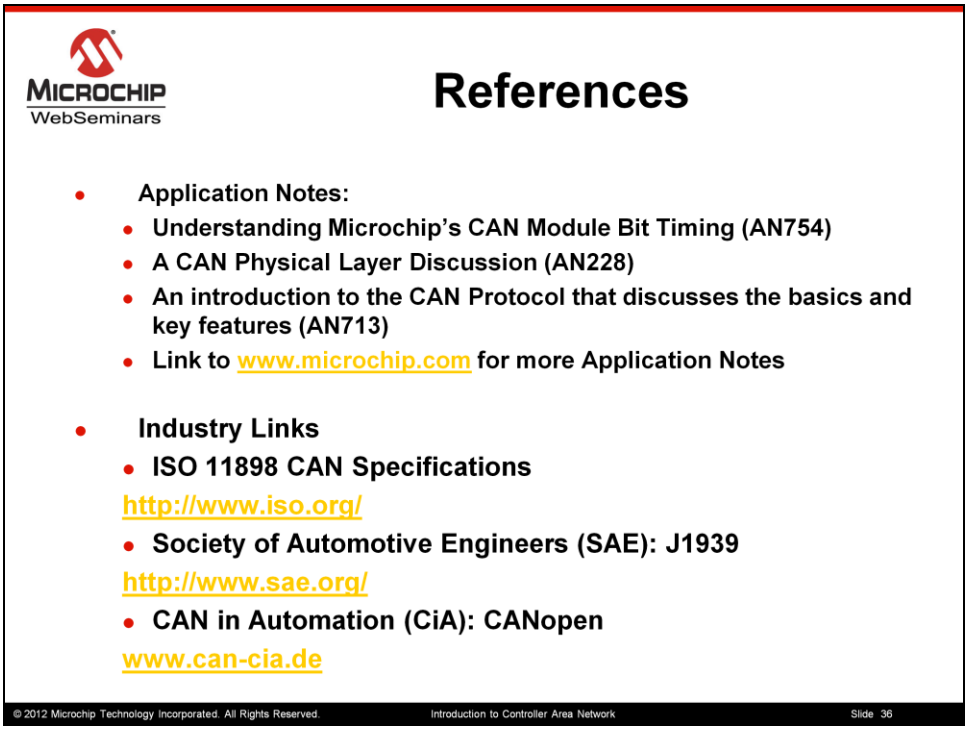
so for additional information I would recommend that you refer to the ISO-11898 specification while using the applications notes listed in the reference section to understanding it.

We also covered the basic operation for transmitting and receiving a CAN Frame

Keep in mind that the Most of the Protocol that ensures data integrity is handled the low level hardware CRC calculation, synchrotrons, checking for error conditions, etc,… so the software overhead is minimized



Here are some ref for the CAN protocol



As mentioned earlier you can refer to our application notes on our website to learn more about CAN

I would also recommend that you visit the following Industry Websites to continue your learning as well.