CAN Primer: Creating Your Own Network

ARM® Keil™ MDK™ toolkit ***featuring Serial Wire Viewer and ETM Trace***

V 3 Robert Boys bob.boys@arm.com

ARM KEIL Microcontroller Tools

Introduction:

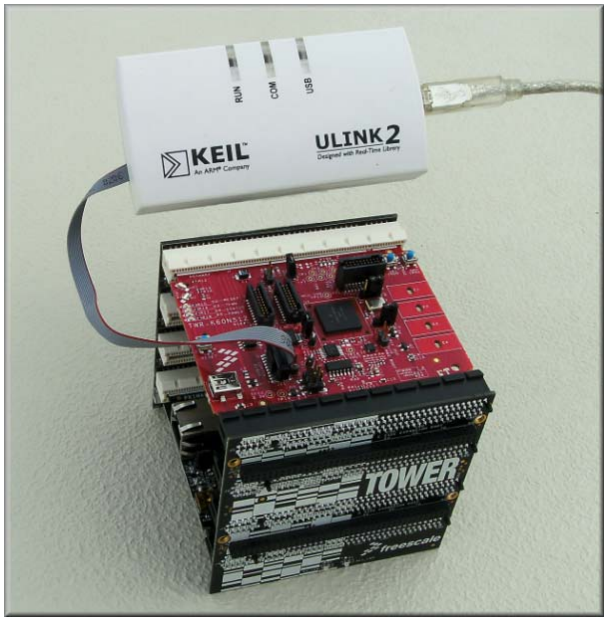
CAN is extensively used in automotive but it has found applications everywhere. There are many "application" layers available for CAN such as ISO 15765 (cars), J1939 (trucks), DeviceNET and CANopen (both are for factory automation) but it is very easy to develop your own protocol that will fit and simplify your needs. Modern CAN transceivers provide a stable and reliable CAN physical environment without the need for expensive coaxial cables. Nearly all of the mystery of CAN has dissipated over the years. There is plenty of example CAN software to help you develop your own network.

Many think CAN is just for automotive, but this is not true. CAN *has* become the standard for vehicle networks, but it has been adopted in most other fields. As you find out in these pages, there are no attributes in the Bosch CAN specification that are automotive related. It is completely generic. You can easily implement your own protocol on top of CAN.

A CAN controller is a sophisticated device. Nearly all the features of the CAN protocol described here are automatically handled by the controller with almost no intervention by the host processor. All you need to do in practice is to configure the controller by writing to its registers, write data to the controller and the controller then does all the housekeeping work to get your message on the bus.

The controller will read any frames it sees on the bus and hold them in a small FIFO memory. It will notify the host processor that this data is available which it then reads from the controller.

The controller also contains a hardware filter mechanism that can be programmed to ignore and discard those CAN frames you do not want passed to the processor. This saves on processor overhead.



Freescale K60 Tower Development System

Modern bus transceiver chips have made the physical CAN bus much less "finicky" and easier to construct and maintain.

The techniques discussed can be applied to many other microprocessors equipped with CAN controllers.

You can adapt CAN source code to any board with a CAN node. CAN controllers are quite similar. They all have to be able to communicate with each other easily and reliably.

Keil MDK: MDK supports Kinetic processors. There is no charge for the evaluation version: MDK-Lite™. You can use MDK-Lite to develop simple CAN examples.

See www.keil.com/freescale

See http://community.arm.com/groups/processors for more ARM processor information.

Keil Evaluation boards that support CAN:

How CAN works:

More Useful Information:

Main Features of CAN:

For the purposes of this article; we will assume a CAN network consists of the physical layer (the voltages and the wires), a frame consisting of an ID and a varying number of data bytes all with the following general attributes:

1. 11 or 29 bit ID and from zero to 8 data bytes. **TIP:** These attributes can be dynamically changed "on the fly".
2. Peer to Peer network. Every node can see all messages from all other nodes but it normally can't see its own.
3. Nodes are really easy to add. Just attach one to the network with two wires plus a ground.
4. Higher priority messages are sent first depending on the value of the ID. The lower ID has a higher priority.
5. Automatic retransmission of defective frames. A node will "bus-off" if it causes too many errors.
6. Speeds from approximately 10 Kbps to 1 Mbps. **TIP:** All nodes ***must*** operate at the same frequency.
7. The twisted differential pair provides excellent noise immunity and some decent bus fault protection.
8. The CAN system will work with the ground connection at different DC levels. **TIP:** Or no ground at all.

The Ground:

This is a contentious issue. A CAN system, especially in vehicles, sometimes must endure large ground loops or corrosion that can compromise signal integrity. CAN is designed using its differential pair to ignore ground voltage differences of many volts. The differential pair also cancels out incoming common mode interference and cancels potential outgoing EMI.

This means that if the ground wire is cut or doesn't exist, as long as CAN-Hi and CAN-Lo are intact, the system will perform at high performance capabilities. CAN, depending on the transceiver chip, can handle various bus problems such as open or shorted lines. This capability is lost without the ground. Therefore, it is recommended to always include a ground in your system design. If the ground is made through a chassis connection or negative power supply rail, any shielded CAN cables must have the ground connected at one end only to minimize ground loop problems.

The CAN System Layout:

A CAN network consists of at least two nodes connected together with a twisted pair of wires as shown below. A ground wire can be included with the twisted pair or separately as part of the chassis. One twist per inch (or more) will suffice and the integrity of the ground is not important for normal operation as described above. As in any differential systems; the important signal is the voltage levels ***between*** the wire pair and not their values to ground or a voltage supply.

The maximum length of the network is dependent on the frequency, number of nodes and propagation speed of the wire. It is relatively easy to have a 20 node (or more), 500 Kbps system running 30 or 40 feet (or more).

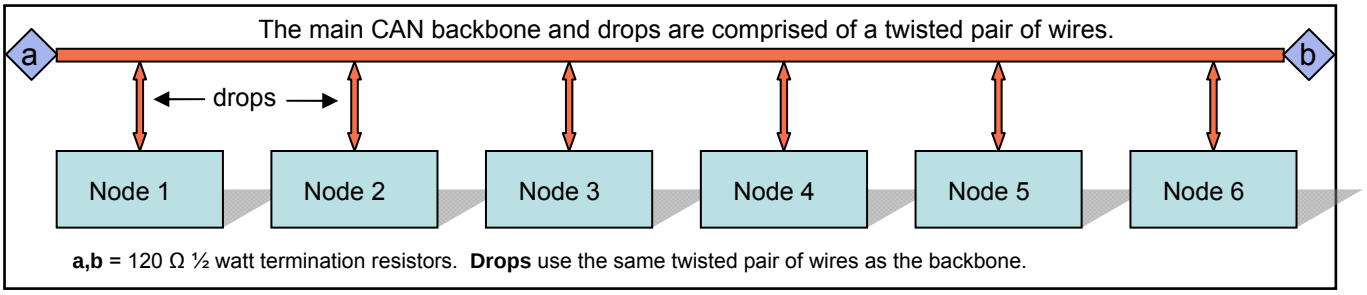
TIP: The drops should be less than 3 feet and randomly spaced to reduce standing waves. These issues all become more important at higher bus speeds i.e. 500Kbps and higher. CAN is completely described in ISO 11898.

Since the twisted pair is a transmission line, 120 ohm termination resistors are needed at both ends of the backbone. Do not put any resistors at the nodes unless a node is at the end of the backbone. Sometimes the resistors are not at the end of a backbone but very close and this seems to work. Resistors are often installed inside an end node chassis or module.

TIP: Your total resistance value as measured between the two twisted wires will therefore be 60 ohms. 10% is good enough.

CAN is a broadcast system. Any node can "broadcast" a message using a CAN frame on a bus that is in idle mode. Idle is at least 11 successive recessive bit times ("1" or ~0 volts CAN Hi to CAN Lo). Multiple controllers tend to start their messages at the same time. Every node will see this message. A "message" can be considered the same as a CAN frame until you need to use more than one frame to send a long message. In this case, you would use some sort of a multi-packet protocol.

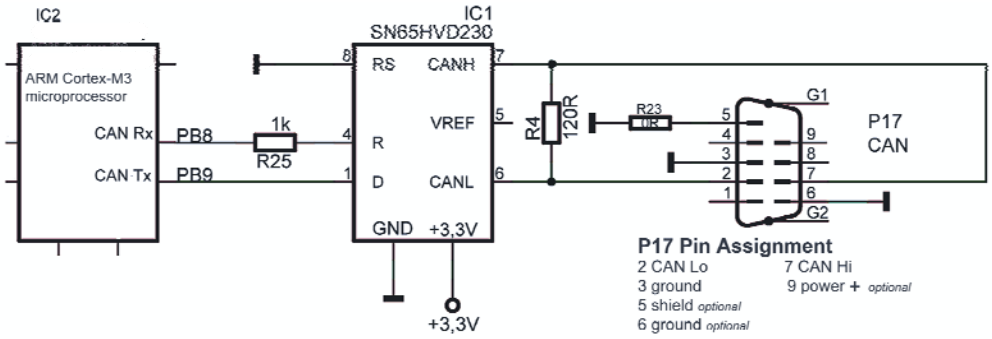
TIP: It is up to a receiving node if it must keep or ignore a frame. This can be handled in either your processor software and/or by configuring the CAN controller acceptance filters.



A Node Schematic:

This is the schematic diagram from a Keil evaluation board. IC1 is a Texas Instruments CAN transceiver which performs the conversion between the single-ended CAN controller CAN Tx and CAN Rx signals to the bi-directional differential pair of the CAN bus called CAN Hi and CAN Lo (High and Low). This schematic is complete. The processor CAN I/O is TTL, CMOS and 5 volt tolerant, all at the same time making it exceptionally easy to design the interface.

This transceiver IC1 connects to the Cortex-M3 microprocessor IC2 which contains an integral CAN controller via two pins: D (Driver input) and R (Receiver output). The corresponding nomenclature on the processor is CAN Rx and CAN Tx. CAN Tx connects to D. CAN Rx connects to R. It is that simple. Some processors have multiple CAN controllers. These are usually used in routers, gateways or to create more receiver FIFO memory for intentionally slowed down CPUs (for EMI reasons). For general use a node normally needs only one controller. If it had at least two, it could talk to itself.



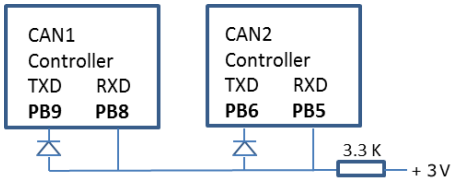
RS on IC1 (slope control) is used to adjust the rise and fall times of the output edges to limit EMI from the twisted pair.

Note R4, the 120 ohm termination resistor. This evaluation board is meant to be used with one other board as a small test network. If this board is used as a node, and is not at one of the ends, this resistor should be removed and external resistors used. P17 corresponds to a generally accepted standard for CAN on DB9 connectors. P17 Pin 7 is the CAN Hi bus line and pin 6 is CAN Lo. **TIP:** If the CAN Hi and CAN Lo wires are reversed, the network will not operate.

The Keil board has one CAN controller. Since there must always be two CAN nodes for a network, you need another board with a CAN node. Most CAN analyzers can act as the 2nd node. Many other boards, including those contain two CAN nodes.

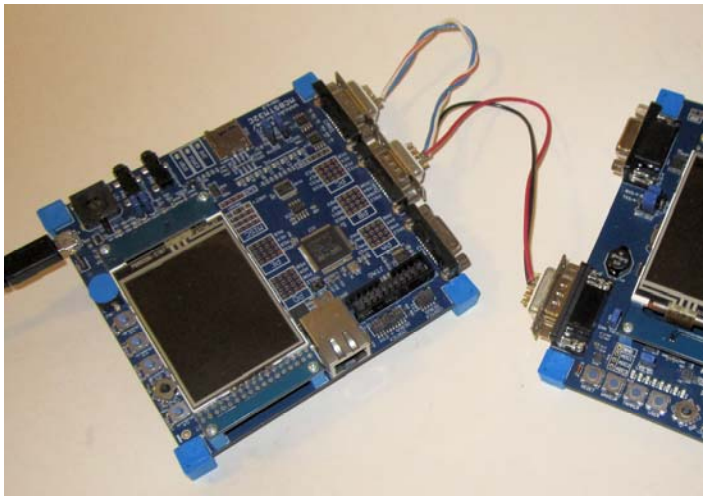
A Tiny Network without Transceiver ICs:

Sometimes you have a CAN equipped processor on a low cost board but it has no CAN transceiver chips. Here is a method that can be used to create a small experimental network with such a board. There will be no noise immunity and you might have to lower the speed….but many experimenters have made this work satisfactorily. Use a signal diode similar to 1N914 or 1N4148. Power supply diodes usually do not have a fast enough recovery time for CAN to function.



A three node CAN network. Uses two CAN nodes on left board and one on the right board. Termination is on the boards. On board CAN Transceiver chips are used. These boards can be connected to any Hi-Speed network as long as the bus frequencies are the same.

This network works perfectly even though the wires are a bit sloppy and are hardly a twisted pair. This is because of the network's small size and robustness of CAN in general: even at 500 Kbps.

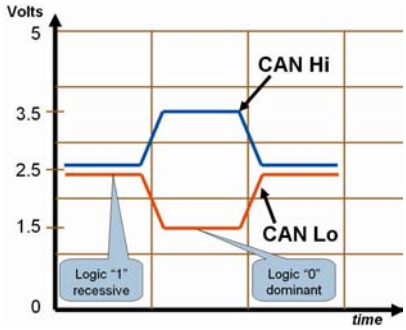


Physical Layer: *the wires and the voltages…*

There are three physical layers used in CAN: Hi-Speed, Fault Tolerant and Single Wire. Hi-Speed is the most common and is the only one we will use in this article. Fault Tolerant offers more robustness as its name implies and is used more often in European autos. Single Wire is used by General Motors and a few others as a low speed body network.

Hi-Speed in cars has a speed of 500 Kbps, trucks are 250 Kbps. CANopen runs up to 1 Mbps. Fault Tolerant is usually 125 Kbps and GM Single Wire is normally 33.33 Kbps. **TIP:** 1 Mbps in a large system is difficult to handle. 500Kbps is easier to use and maintain and will present fewer design problems. In general, the longer the physical wires and more nodes, the frequency should be lowered to maintain stability and reduce bus errors. You do not need to use these exact frequencies in your own nodes that will not be connected to other nodes and therefore do not require bus speed compatibility.

To change from one layer to the other requires only the transceiver chip need be exchanged and probably the speed changed. These three flavors of CAN cannot be physically connected to each other as the voltage levels are different. You need to use a router or gateway to join different CAN networks together. Any CAN controller will properly service all three flavors of CAN with the appropriate transceiver.



The Hi-speed CAN physical layer is merely a twisted pair of wires with a 120 ohm termination resistor at each end and twisted wire drops to the individual CAN nodes. You can connect your node's transceiver chip directly to the bus. It is possible to implement isolation techniques using appropriate devices.

CAN Hi voltage with respect to ground changes between 2.5 to 4 volts nominal. CAN Lo changes from 2.5 to 1 volt. Therefore the difference between the two is either 0 volts (is logical "1") or 2 volts (is logical "0").

0 volts is known as the "recessive" state and 2 volts is the "dominant" state.

These two signals, CAN Hi and CAN Lo, are 180 degrees out of phase as indicated in this diagram. Bus idle is when the CAN Hi and CAN Lo voltage difference is near zero (Recessive) for at least 11 successive bit times.

A 2 node CAN cable assembly:

Two wires and two 120 Ω resistors.

A ground connection is preferred. It facilitates certain bus defects such as open or shorted lines

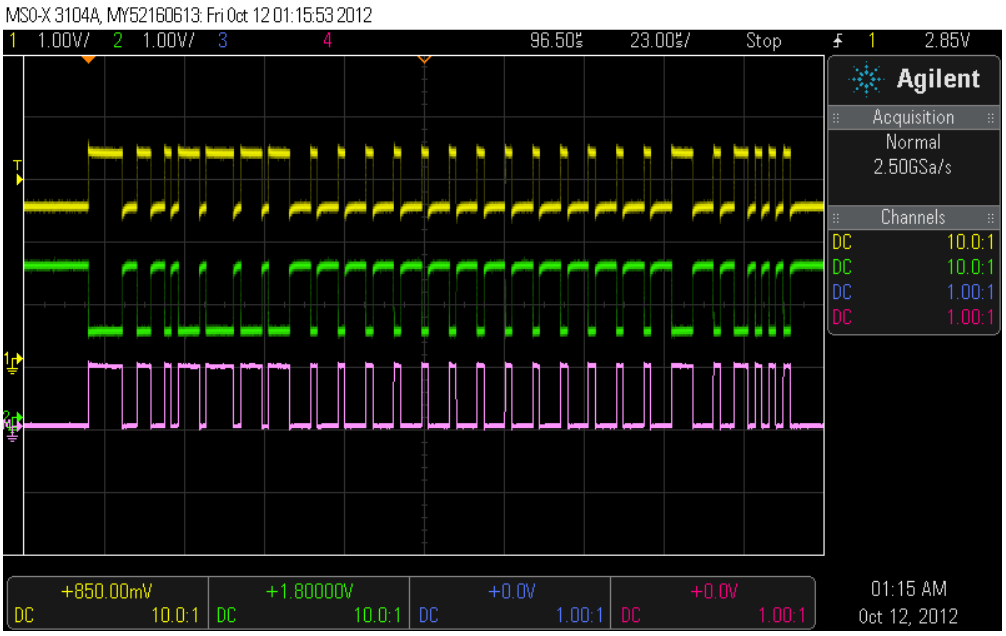
A CAN frame:

The top two traces are CAN_Hi and CAN_Lo respectively. Note they are 180 degrees out of phase. These are the differential signals.

The bottom trace is the algebraic sum of the top two.

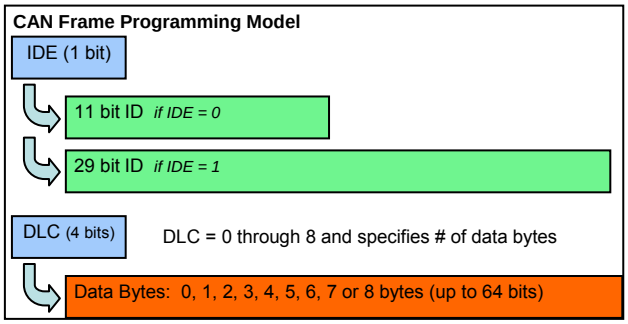
This is from the Keil CAN example program using transceiver chips.

TIP: It is a good idea to view the waveforms in your CAN network to make sure they are not distorted. The robustness of CAN will tend to cover up some design defects.



The CAN Frame:

The CAN frame has many fields but we can simplify this to a Programming Model as shown. These fields are accessed by your software through the CAN controller registers. The configuration registers are not included here.



- **IDE:** Identifier Extension: 1 bit - specifies if the ID field to be transmitted is 11 or 29 bits:
  If IDE = 0, then the ID is 11 bits.
  If IDE = 1, then the ID is 29 bits.
- **ID:** Identifier: 11 or 29 bits as set by the IDE field. This part of the CAN frame sets the priority.
- **DLC:** Data Length Code: 4 bits - specifies number of data bytes in the frame from 0 through 8.
- **Data Bytes:** 0 through 8 bytes.

TIP: A CAN frame with an ID field of either 11 or 29 bits ***and*** with zero data bytes is valid and useful.

TIP: You tell the transmitter size of ID and number of data bytes. The receiver tells you these values when it gets a frame.

ID: Identifier: 11 or 29 bits

The Identifier can be used for any purpose. It is often used as a node address or to identify requests and responses. CAN does not specify any mandatory ID values. 11 bit is often called Standard CAN and 29 bit is often called Extended CAN.

1. If two or more CAN messages are put on the bus at the same time; the one with the highest priority (the lowest value) ID will immediately get through. The others will be delayed and will be resent at the next bus idle time.
2. An ID of 0 has the highest priority and will always get through. When the first 11 bits of an 11 and 29 bit ID are the same, the 11 bit ID has priority over the 29 bit ID. This is because the IDE bit = 0 for 11 bits and wins arbitration.
3. You can have any mixture of 11 and 29 bit IDs on the bus. The controller can easily sort this out.
4. Messages tend to start transmitting at the same time. A CAN node is not allowed to start transmitting a frame in the middle of another node's frame. This will cause a bus error. Can controllers will not make this mistake.
5. CAN controllers can be configured to pass only certain received messages to its host processor. Choose your ID values carefully to take advantage of this if needed. This will reduce the workload of a node's CPU.
   This is why some systems use 29 bit IDs even though they do not need that many addresses. This facilitates grouping of IDs for easier filtering since acceptance filters usually do not usually have a very fine granularity.
6. You can use the ID for data, node addressing, commands and request/response sequences. Commercial protocols use any of these in practice. You can create your own method if that best suits your purpose.

TIP: Make sure two nodes will ***never*** send the same ID value at the same time. It is illegal but possible to do this. If two messages sent at the same time are completely identical, they will be seen on the bus as one. If the data bytes are different, a bus error will result and the frames will be resent continuously and havoc is created on the bus for a short time until a bus-off.

Data Bytes:

You can select from 0 to 8 data bytes using the 4 bit DLC field. A valid DLC is returned when a frame is received.

1. You can have any number of data bytes in frames on the CAN bus. The controller can easily sort this out.
2. If you always use only one number of data bytes, your software will be much simpler to write and debug.
3. The data bytes can contain anything. It is not prioritized like the ID is. CAN does not specify mandatory data.
4. Protocols such as J1939 specify these for data as well as control bits for multi-frame transmission schemes.

Remote Frames:

These are not used much anymore but are worth mentioning. A remote frame is a quick method of getting a response from another node(s). It is a request for data. The requesting node sends out a shortened CAN frame with only a user specified ID number and the number of data bytes it expects to receive (the DLC is set). No data field is sent. The responding node(s) sees this frame, recognizes that it has the desired information and sends back a standard CAN frame with the same ID, DLC and with data bytes attached. All of this (except that the response node recognizes the ID and DLC) is implemented in the CAN controller hardware. Everything else must be configured by the user software.

Other Bit Fields: Only the ACK bit will be mentioned in this document:

ACK: This is a one bit field in the CAN frame created by the transmitting node but set by all the ***other*** nodes.

TIP: The number 1 reason people can't get their CAN node working is you need at least two nodes to work. When a node puts a message on the bus, it will wait for the ACK bit to be asserted by any other node that has seen the message and determined it to be valid. If so, the transmitting node finishes the message and goes into the idle state or sends its next message. If not, it will immediately resend the message forever until the ACK is asserted or the controller is RESET. This transmitting node will never go into bus-off mode while re-transmitting this message. Note that a standard CAN test tool will usually act as a second node by providing the ACK signal unless its controller has the ACK feature disabled.

TIP: This presents an excellent opportunity to provide an easy test situation to confirm you are sending out CAN frames. It won't tell you the frequency, ID or data bytes values, but it will tell you if you are putting out something.

1. Connect a termination resistor to your single CAN node with a transceiver connected to the CAN controller. If you do not have a transceiver connected, join the TXD and RXD of the controller together. No resister is then needed.
2. Do not connect any other node or test tool. Just one node running by itself.
3. Connect an oscilloscope hot lead to CAN Hi and ground to CAN Lo. The scope ground must be isolated from the CAN ground. You do not need a high speed scope – almost any will suffice. You can also connect a scope to the CAN controller output pin and ground for a very clean signal.
4. Configure your CAN controller and write the IDE, ID, DLC and any data bytes into the appropriate registers.
5. A CAN frame will now be continuously displayed on the scope. RESET the processor to start over.

TIP: You can measure the CAN frequency with the method described at the bottom of this page.

Bit Stuffing:

The CAN protocol states that when there are 5 consecutive bits of the same polarity, one bit of opposite polarity will be inserted to prevent sync loss. These bits make the CAN frame longer and are very common. These bits are inserted and removed automatically by the CAN controller and are only visible when an oscilloscope is attached to the bus.

TIP: When bits are added (or not) to the CAN frame as various messages are sent on the bus, the changing frame length will look like jitter on the bus. It is not jitter of course: CAN just works this way. Just something else to be aware of.

Bus Loading:

Many CAN networks work on a bus loading from 15 to 35 % and this is increasing in some applications. A higher bus loading can cause lower priority messages to be delayed but these messages will still usually get through in a timely fashion. It is quite difficult to achieve 100% bus loading but one can come close. System performance does not drop greatly at high bus loading. Recall the message with the highest priority (lowest ID #) goes straight through with no time delay.

TIP: It is possible to get very high bus loads in a short period of time in a CAN network. CAN does not automatically space out messages. It is possible to get a series of back-to-back messages that will equal nearly 100 % bus loading. You should be prepared for this. One solution is to select only those messages needed by a node by programming its acceptance filter. Another is to have your software space out the messages. This transitory problem is quite hard to detect and diagnose.

TIP: It is possible that on a highly loaded bus, low priority messages will never get transmitted or be delayed by higher ones.

Bus Speed:

Bus speed in a system is a balancing act between things such as propagation delays (from bus length) and EMI emissions versus desired data throughput. Run your network as fast as possible for stable operation and with enough throughput. Do not run it much faster than it needs to be, but make some room for later expansion. Choose standard bus speeds.

TIP: If your network is not stable: make sure you have two good termination resistors at each end of the network. Try slowing the CAN speed down to see if this helps. Resistors can be 120 ohm ½ watt and their value is not critical. Try adjusting the BTR0 and BTR1 timing registers to obtain the most stable operating parameters.

TIP: **How to determine the bus frequency of a CAN signal:** This is the best and sometimes only way to determine this.

1. Connect an oscilloscope to CAN Hi or CAN Lo and ground. You do not need a high speed scope – almost any will suffice. You can also connect a scope to the CAN controller Tx output pin and ground for a very clean signal.
2. Display a trace. You might need a storage scope to see just one trace due to the non-repetitive nature of CAN.
3. Pick the smallest width signal pulse and measure its time period in seconds as accurately as you can.
4. Invert this value (divide into 1) and you have the CAN speed in bits per second. i.e. 2 µsec = 500 Kbps.

Bus Errors:

Recall we said that all nodes (including the transmitting node) check each CAN frame for errors. If an error is detected, here is what happens:

1. All the nodes will signify this fact by driving the bus to logical 0 (dominant state) for at least 6 CAN bits.
2. This violates the Bit Stuffing rule (never > than 5 bits the same polarity) so every node sees this as an error.
3. This so called "Error Frame" signals to all nodes a serious error has occurred if they don't already know it.
4. The transmitting bus abandons the current frame and adds 4 to its 8 bit TEC register. (Transmit Error Counter)
5. IF this TEC equals 0xFF, the transmitting node goes BUS OFF and takes itself off the bus. (TEC normally = 0)
6. IF not, it attempts to retransmit its message which has to go through the priority process again with other messages.
7. All other nodes abandon reading the current frame, and add 4 to their own REC register. (Receive Error Counter)
8. Any nodes that have messages queued up will transmit them now. All others start listening to the bus.
9. Hopefully, this time the message will be broadcast and received error free. Each time a frame is transmitted and/or received successfully, the corresponding TEC and REC registers are decremented (usually by only 1)

Super TIP: Error Counters ? These are two 8 bit registers in every CAN controller and you can read these with your software. This is a good idea because it gives some indication of general bus health and stability. In a good CAN network, TEC and REC will equal 0. If it starts having higher values, something has happened to your network. The usual suspect is bad hardware. The problem is usually in either the wires, the transceiver chip or the termination resistors.

TIP: Don't forget that if something happens to the integrity of your twisted pair, such as CAN Lo disconnected; it might still work but with greatly reduced noise immunity (that is what differential signals do best). If your network is in a very noisy environment, there might be more transient bus errors. This is very tricky to debug without knowledge of the REC and TEC contents. Read TEC and REC with your software and report it to your diagnostic routines.

In a general sense, TEC represents a given node's errors and REC indicates the other nodes' errors.

Bus Off: As mentioned, if a transmitting node detects it has put too many bad frames on the bus, it will disconnect itself. It will assume that there is something very wrong with itself. To get back on the bus depends on how you configure the controller. This can require a controller RESET or a certain number of good frames received. See your controller docs.

BUS Faults:

This is different (sort of) from a bus error. We normally think of a bus fault as something that has happened to the "wires" or the output transistors of the transceiver chip. Not all bus faults will result in a bus error. A bus error can be thought as the CAN controllers' reaction to a bus fault such as noise, a faulty node or other errors.

What happens if one of the twisted pair opens or is shorted out ? CAN has automatic mechanisms for this. Not all transceiver chips implement all of them. You can usually short CAN Lo to ground (ISO 11898 says you can short Hi also) or open one CAN line. The ground needs to be connected for this case. You can't short both Hi and Lo together (Fault Tolerant will work) or open both up. You can cut the ground or have a large ground loop present and CAN will still work.

Serious bus faults may be reported as a bus error as described above. At least one node must try to transmit a frame in a bus fault condition to trigger a bus error. A bus in idle mode can't trigger a bus error. When the bus fault is removed, in many systems the network will come back to life if so configured. CAN has excellent noise immunity because of the twisted pair that are 180 degrees out of phase. The common mode noise is cancelled out and the differential CAN signal is not affected.

The Ground: Strictly speaking, the ground is not needed for CAN operation if the twisted pair is intact. This is readily shown with simple experiments. One experiment showed a small network still worked properly with two nodes having a 40 volts DC ground difference ! However, it is a good idea to include a good ground in your system design. Some bus faults need the ground to allow the transceiver to compensate. This is good engineering practice. Watch out for ground loops !

For a practical demonstration of BUS faults: see the section on getting a real system working.

TIP: How can you create a Bus Error for testing ? ***Easy:*** have a node send a message at the wrong frequency. When this frame tries to get on the bus this is certain to create a bus error condition. Some CAN controllers can send a one-shot frame. This is useful if the ACK bit does not get set by other nodes and you do not want this frame being sent forever.

Bonus TIPS: **Here some items NOT part of the CAN specification but might prove helpful in your system:**

1) Transmitting data sets greater than 8 bytes:

Clearly, transmitting a data set greater than 8 bytes will require multiple frames and this will require some planning. Such schemes can become very complicated as they have to deal with a wide-ranging set of contingencies. If you can focus on a narrow requirement set, design of a simpler protocol is possible.

Most current schemes use the first data byte to contain the number of total data bytes to follow plus a counter to help determine which data byte is which. The ID usually identifies the node plus whether it is a request or response message. If you want to use an existing protocol see ISO 15765. This is what automobiles use. OBDII diagnostics on vehicles also use this protocol. OBDII is an example where one message can be comprised of many CAN frames. Diagnostics are common.

2) Periodic, Request/Response and Command Frames:

Periodic: This technique sends a frame out periodically – several times a second is usual. This frame will contain data that any node can use if it wants to and is identified by its ID. Examples are speed, position, pressure and events. Messages that are lost (usually because the processor fails to empty the controller input queue fast enough) are replaced quickly.

Request/Response: A node sends out a frame requesting certain specified information. Any other nodes that have the requested information then put it on the bus. The ID identifies the Request and the Response frames by changing one bit of the ID. An example is that ID 0x248 is a Request frame and 0x648 is its Response frame. The Request frame data bytes indicate what information is requested. The Response frame will contain the requested information or an error indication.

Command: A frame commanding some event to be performed. The ID usually contains the address of the commanded node and the data bytes the actual command(s). An ID signifying a broadcast message will be sent to all nodes. Technically all nodes can see all messages anyhow: but this can allow a message to get past a filter or from being ignored by a node. Sometimes an Acknowledge frame will be returned. Note: This is not the ACK bit.

TIP: You might want to consider a blend of these types of traffic depending on your system's needs.

3) Time-outs:

Automotive CAN networks use time-outs and this concept is easily and effectively transferred to systems in other fields. A time-out occurs when a node fails to respond to a request in a timely fashion. Time-outs are handled completely by software and not by the CAN specification. A time-out is helpful to recover from problems with the network such as severe bus errors, catastrophic bus faults, faulty nodes, intermittent connections or a user abort.

The result is usually a limp-home mode where a node will attempt to run itself without information from the rest of the network. In some cases, a punitive limp-home mode is entered that forces the user to perform repairs. Another result is to revert back to normal operation. This is common when a user stops making inputs for a long time. You do not want a system to sit in a configuration mode forever. You must control your processes.

A time-out consequence can be a system RESET or less likely, a shutdown. In any case, notification to an operator is a very good idea. In extreme circumstances, such as a remote system not accessible: a mode to download new firmware is effective.

A good example is if the vehicle transmission fails and proper gear shifting becomes impossible. In this case, the module will go into limp-home mode and the transmission might be put into one gear such as second to allow the vehicle to still be driven. This can be for safety reasons or to prevent further damage to the power train. Another example of a time-out is when the setting of the clock by a user is started but stopped midway. After the time-out, the clock will revert to its normal operating mode.

Another good example is the MARS Rovers. If communication stopped for a certain period of time: it will be assumed that a catastrophic event has occurred. Hopefully this will be a software bug and not a hardware failure. The Rover will go into a special mode where it listens for a software update or if really bad, a complete firmware replacement to be sent from Earth.

Heart-beats and Address Claiming: The other side to a time-out is a heart beat. Periodic messages can be sent out to determine that all nodes are on the bus and active. CANopen uses such heart-beats. J1939 has a software mechanism where each node can declare itself to be on the bus and be recognized by the other nodes. This is called "Address Claiming" and occurs during the system startup. None of these mechanisms are provided by the CAN specification but rather by your software or a suitable specification or protocol.

Sequence of Transmitting Data on the CAN Bus:

1. You give the transmitter the ID, the size of the ID (IDE), the number of data bytes (DLC) and the data if any.
2. You then set a bit to tell the transmitter in the CAN controller to send this frame.
3. Any node(s), seeing the bus idle for the required minimum time, can start sending a CAN frame.
4. All other nodes start receiving it except those also starting to transmit a message at the same time.
5. If any other node starts transmitting: the arbitration process starts – the node with the highest priority (lower ID value) continues and lesser priority nodes stop sending and immediately turn into receivers and receive the priority message. **Note:** The losing node "knows" the beginning ID of the other message. CAN arbitration is non-destructive.
6. At this point, only one node is transmitting a message and no other will start during this time else a bus error happens.
7. When the transmitting node has completed sending its message, it waits one bit time for the 1 bit ACK field to be pulled to a logic 0 by any other node (normally all of them) to signify the frame was received without errors.
8. If this happens, the transmitting node assumes the message reached its recipient, sends the end-of-frame bits and goes into receive mode or starts to send its next message if it has one. The receiving nodes pass the received message to their host processors for processing unless the acceptance filtering prevents this action.
9. At this time, any node can start sending any message or the bus goes into the idle state if none do. Go to 1.
10. If # 7 does not happen (ACK bit not set) then the transmitting retransmits the message at the earliest time allowed. If the ACK bit is never set, the transmitting node will send its initial message forever.

Transmitting Notes:

- **How do I transmit my message ?** Easy – you create the CAN frame you want to send by loading up the IDE, ID, DLC and any data byte registers in the CAN controller and then, in most controllers, you set a bit that triggers sending the frame as soon as legally possible. After this, the controller takes care of sending all frame bits. Unless the controller signals otherwise to the processor, you can assume the message was sent.
- **How does a node know when it should transmit a message ?** The CAN controller continuously monitors the bus. When it sees > = required number of idle bits (11 after ACK bit), it starts transmitting. It is quite possible for other node(s) to start transmitting at the same time. Arbitration decides which message is actually transmitted.
- **What if there is an error ?** All nodes, including the transmitting node, monitor the bus for any errors. If an error condition is detected – a node or nodes signal to the other nodes there is an error by holding the bus at logical 0 for at least 6 bus cycles. At this point, all nodes note this error event and take appropriate action. The message being sent (and now aborted) will be resent but only for a certain number of times. See Bus Errors on page 8.
- **What if no node wants or uses the message ?** Nothing. The ACK bit only says that the CAN frame was transmitted without errors and at least one node saw this frame error free. Remember the transmitting frame can't ACK itself. CAN does not provide any acknowledgment mechanism that a frame was used or not by its intended recipient. If needed, you will have to provide this in your software as many systems do.
  TIP: In a periodic system, if a node misses a message, it doesn't matter much as a copy frame will be along shortly.
- **How do I add a node ?** Just attach CAN-Hi and CAN-Lo signals to the existing network. If your CAN controller is not initialized, nothing will happen. If it is initialized to the correct frequency, it will start listening to the bus and set the ACK bit if appropriate. Until your software reads messages out of the buffer, messages will be lost.
- **What happens if I "hot plug" a node on the bus ?** Usually any disturbances will be taken care of by the CAN error detection schemes. A message that is sufficiently corrupted will result in a bus error and subsequent retransmission. If the TEC register is high or the disturbances are of a long duration, a bus-off might result.

Arbitration Notes:

1. Arbitration is performed bit-wise. That is, bit by bit on the ID (11 or 29 bit) and the IDE bit. No other bits are used.
2. The node that wins arbitration is not slowed or delayed by this process. CAN arbitration is non-destructive.
3. The losing nodes, if there are any, will attempt to retransmit at the next idle bus time.
4. CAN is not deterministic. This means you are usually never sure when a CAN message will appear on the bus.
5. If you need determinism, try Time Triggered CAN. TTCAN is described in ISO 11898-4. It is a software layer that sits on top of regular CAN. It places frames into specified time slots.

Sequence of Receiving data from the CAN Bus:

1. All nodes except those currently transmitting frames and those in bus-off mode are in listening mode.
2. A CAN frame is sent using the procedure as described previously: Sequence of Transmitting Data on the CAN Bus:
3. This sent frame is received by all listening nodes. If deemed to be a valid CAN message with no errors – the ACK bit is set by all listeners. In CAN terminology, this set to the "dominant" state as opposed to the recessive state.
4. The frame is sent through the controller's acceptance filter mechanism if it is enabled. If this frame is rejected: it is discarded. If accepted, it is sent to the controller FIFO memory. If the FIFO is full, the oldest frame is lost.
5. The host processor is alerted to the fact a valid frame is ready to be read from the FIFO. This is done either by an interrupt or a bit set in a controller register. This frame must be read as soon as possible.
6. You do not tell the receiver what the ID size is or the number of data bytes to be received. When the receiver gets a valid frame, it deciphers this information and provides you with the appropriate IDE and DLC register values.
7. The host processor decides what to do with this message as determined by your software.

TIP: You must decide whether to use polling or interrupts to alert the host processor a frame is available. Polling is where the host processor "polls" or continuously tests the bit mentioned in # 5. Polling runs the risk of losing or "dropping" a frame but is sometimes easier to implement and debug. Using interrupts is the recommended method and causes the CPU to jump to a handler to read the frame from the controller. Make sure your processor can handle 100% bus load bursts.

Receiving Notes:

1. **What happens if a message is "dropped" ?**

   This can cause some problems as CAN itself does not have a mechanism for acknowledging a CAN frame. If you want this, you must add it to your software. In the case of Periodic Messages, it doesn't normally matter much as a replacement message will be along shortly. This appears to be designed into CAN to handle dropped messages.

2. **How fast do I have to read the FIFO to not drop messages ?**

   It depends on the CAN speed, frame size, and bus loading. It is a good idea to read these frames as soon as possible since once a frame is dropped, it cannot be recovered or automatically resent by the transmitting node. It is gone forever unless you provide a suitable mechanism in your software to have it resent.

   It is possible to have a burst of CAN traffic approaching 100% bus loading when the controller dumps all its data on the bus. Your system must be prepared for this event. The CAN specification does not space out frames.

3. **How do the CAN controllers stay in sync when there is only the bus idle voltage (0) and no transistions:**

   The CAN controller depends on its internal clock to stay as close to the design frequency as possible. Upon receipt of the start bit, an internal counter starts counting the "time quanta" (TQ) that further divide the bit time. The number of TQs in a bit period is set in the controller by the user. The controller will automatically add or subtract TQs to adjust its effective operation frequency.

   CAN bit time transitions are used to calculate the correct number of TQs needed to keep the receivers in sync.

   See the data sheet for your CAN controller for details.

4. **I have a high bus loading factor. How can I reduce the pressure on my CAN controller ?**

   There are many ways and here are several:

   - Use the acceptance filters to ignore any messages your processor does not need. Ignored messages will never be sent to the processor. They are discarded very quickly by the CAN controller.
   - Use more than one CAN controller in your processor to receive CAN frames. Set the acceptance filters to divide the messages by ID (or even 1st data byte) to each of the controllers. Your processor must still have the ability to process the messages. For transmit: use mailboxes to alternatively transmit the CAN frames.
   - Space the messages sent by the nodes.
   - Use sub-nodes. Sometimes using a different protocol than CAN is appropriate.
   - Use a faster ARM processor or increase the bus speed if practical. Or use a combination of these tactics.

5. **Where do I get the values for the timing registers BTR0 and BTR1 ?**

   The controller data sheet will provide the formulae to configure these timing registers. Suggested values for various bus frequencies are usually provided. It is especially important to get the sampling point correctly set.

CAN FD: CAN with Flexible Data Rate:

CAN FD is a new extension to the standard CAN 2.0 protocol. For more information search the internet for the files can_fd_spec.pdf and can_fd.pdf . CAN FD was created by Robert Bosch GmbH.

The CAN frequency and bit overhead added to information carrying bits (ID + data bytes) (also called payload) are limitations to the effective maximum data rate transmission. CAN FD is one solution to this.

In a CAN system, once the frequency is chosen and implemented, it is difficult to change. Changes are easier if you know exactly what nodes are in a system such as in a passenger vehicle. In systems where the nodes can be supplied from different manufacturers depending on customer options or added by after-market users, the change problem is usually a problem.

This is certainly true for SAE J1939. J1939 is used in heavy duty trucks, buses, marine and in construction and farm equipment. It uses CAN at 250 Kbps. This speed is not fast enough for larger, more bus intensive J1939 systems. CAN FD might be a good solution as systems can slowly migrate from CAN to CAN FD.

CAN FD provides:

1. Up to 64 bytes of data bytes. Remember regular CAN has from 0 to 8 data bytes.
2. It is possible to increase the bus speed during the transmission of the DLC, data and CRC fields and before ACK bit. This time period occurs just after arbitration is complete.

Features of CAN FD:

1. A CAN FD controller will be different than regular CAN. It can also transmit and receive regular CAN frames.
2. Uses the same physical layer as regular CAN.
3. Can use the same transceiver although specialized ones might be made available.
4. CAN FD can be used for specific applications such as programming, large data transfers or general use.
5. When CAN FD is transmitting, regular CAN must be in standby. CAN and FD collisions will result in Bus errors.
6. Regular CAN uses bits 0000 through 1000 of the 4 bits of DLC (Data Length Code). CAN FD adds 1001 through 1111 to extend the data field.
7. Reserve bit R0 (in 11 bit) or R1 (in 29 bit) are used to signify the frame is CAN FD. This is the EDL bit.
8. Three new bits are added: They are added just before the DLC field.
    a. EDL: (Extended Data Length): specifies a CAN or CAN FD frame (EDL was called R0 or R1).
    b. BRS: (Bit Rate Switch): switches the bit rate after arbitration and before ACK bit.
    c. ESI: (Error State Indicator): denotes if the node is in error-active or error-passive mode.

Selection of Data Bytes:

The DLC has 4 bits: b0000 through b1000 are used by regular CAN and CAN FD to signify from 0 to 8 data bytes.

CAN FD uses b1001 through b1111. Each of these bits represents not one byte as in regular CAN, but map into a table. This makes a total of up 64 data bytes. See this table to the right: With this scheme it is not possible to select some byte numbers.

DLC	Number of Data Bytes
1001	12
1010	16
1011	20
1100	24
1101	32
1110	48
1111	64

The rest of the story- The Details:

There are many details covered in the Bosch and other documents. CAN FD controllers are now available. For details visit http://can-newsletter.org/ and search for **FD TechDay**.

Other useful network protocols:

Here are some other protocols that might prove useful to complement CAN:

FlexRay: High speed dual channel Time Triggered network. You can use one or two channels. It is used for redundancy in safety critical applications. Search on the internet.

LIN: Single wire network using a common UART. Nearly any controller can be used to implement LIN. LIN is a very low cost network. It is often used as a sub-network to CAN. See www.lin-subbus.org

Safe-by-Wire: A very reliable network used to activate vehicle airbags. Search the web for safebywire.pdf.

Ethernet: It is very common to use CAN for a small, local network (such as on one machine) and then use a gateway to ethernet for the long distances or to connect to the other machines. It can handle the traffic load of multiple CAN networks.

Wireless: WiFi, ZigBee, Bluetooth and NFC (Near Field Communication) are all useful in small networks attached to CAN.

CAN Controllers and their Errata Sheets:

CAN controllers are very sophisticated modules. Many times someone is experiencing trouble getting something to work or has an unexpected crash or result and they desperately search their code for the error causing this. Sometimes the answer lies in the errata sheet and not in your software. This document lists all known deviant behaviour from that claimed in the device datasheet. Some CAN controllers have bugs and you should find out what they are.

Note that technical support staff statistics show that many errors are in the user software code so check this carefully.

You should get the latest errata sheets and read them. You can potentially save an enormous amount of time. Sometimes the weirdest problems are caused by these defects. Then you have to be prepared for the day these bugs get fixed and show up in silicon on your board. Most issues will be in the controllers and not the rather simpler transceivers.

TIP: There are several Internet CAN newsgroups and mailing lists that can help you with your network. Remember that not all people on these groups are experts and there is some risk of getting poor information. Fortunately, these self-proclaimed experts are in the minority. See http://tech.groups.yahoo.com/group/CANbus/ and www.canlist.org.

For CANopen and other information see: www.can-cia.org/ Most CANopen docs are free. Most other documents are not.

Test Tools:

The biggest problem in getting your first CAN network running is that in order to see some messages, you have to have both a receiving node and a transmitting node properly working *at the same time*. This can be quite an onerous task. There are two ways to help here. One is to use a working node such as an evaluation board with some proven CAN examples provided. You can attempt to receive these known good CAN frames with your node.

Second, you can purchase a CAN test tool. This is the best idea. These provide both sending and receiving capabilities and usually (optionally) act as a CAN node. There are two types: simple low cost devices that provide basic creating and displaying bus traffic and those offering advanced capabilities such as translation that can save some serious cash and time.

Typical sources for inexpensive tools are SYS TEC http://www.systec-electronic.com/ and PEAK www.peak-system.com/. Also see www.kvaser.com/. SYS TEK, PEAK and Kvaser are also sold on the USA by www.phytools.com. There are many other companies that sell these types of inexpensive yet useful tools. Search the Internet to locate them.

Oscilloscopes are quite useful in making sure the CAN waveforms are not distorted. This can disclose the causes of some very strange network behavior. For a combination CAN analyzer and oscilloscope see www.phytec.com/pcan-diag.html or search the web for Phytec PCAN-Diag. Standard scopes also work well and of these, memory scopes work best. CAN aware scopes can provide the most useful testing abilities available.

If you are developing a more capable and powerful CAN system, you might want to consider a CAN analyzer. These offer very advanced features such as triggering, filtering and best of all; a database where your ID and data bytes are displayed in words rather than raw hex numbers. This will save a lot of time and make for a better, more reliable product. Normally, you can construct your own database to convert numbers embedded in the CAN frames to your own custom descriptive words.

Typical suppliers are Dearborn Group www.dgtech.com, National Instruments www.ni.com, Intrepid www.intrepidcs.com and Vector CANalyzer www.vector.com. Do not be afraid to use an automotive type device even if your application is something else. CAN is CAN no matter where it is used and no matter what anybody says. Everything else sits on top of CAN. Even so, it would be good to check if an analyzer is sufficiently adaptable for your needs.

As with all tools, buy the best analyzer you can afford ! *You are rarely disappointed with fine products…only cheap ones…*

CAN Documents:

CAN documents are available for ISO (ISO 11898) and SAE (J1939). They are not free. www.iso.org and www.sae.org.

The original Bosch 2.0 document is free: Search the web for can2spec.pdf.

TIP: If you have more than one CAN controller in your processor you can operate these as parallel receivers. Divide the messages up with the Acceptance Filters. This will help capture all the messages on a very busy bus minimizing or eliminating data frame losses. Each CAN controller will handle its share of the messages. This effectively multiplies the number of FIFO buffer memories which is an excellent method of capturing all the CAN frames

How to determine the frequency of a CAN signal:

This is the best and sometimes only way to determine this.

1. Connect an oscilloscope to CAN Hi or CAN Lo and ground. You do not need a high speed scope – almost any will suffice. You can also connect a scope to the CAN controller Tx output pin and ground for a very clean signal.
2. Display a trace. You might need a storage scope to see just one trace due to the non-repetitive nature of CAN.
3. Pick the smallest width signal pulse and measure its time period in seconds as accurately as you can.
4. Invert this value (divide into 1) and you have the CAN speed in bits per second. i.e. 2 µsec = 500 Kbps.

Four Newbie CAN Mistakes you can avoid:

1. You need at least two CAN nodes in order to get past the first initial message. ***This is a very common mistake.***
2. There is a bug in CAN that was turned into a feature. What happens is some noise on the bus causes a transmitter to think its first message was corrupted and it sends out a copy. However, the other nodes think the first message was valid and accept it and also the second copy. Therefore, they see two identical and valid messages. Therefore do not increment or toggle values or states. Send the actual value you want to nodes to receive. This situation occurs rarely but with millions of CAN frames and millions of networks, it does happen and it can be a problem.
3. The CAN controller will add (or not) a bit to the bitstream to ensure there are never more than five unchanging bits. This changes the CAN message length. When such messages are viewed on an oscilloscope – they look like jitter. It is not: this is how CAN works. Do not chase problems in your network on a false assumption you have jitter.
4. Avoid doing tricky and complicated things like changing the CAN frequency. Just keep it simple and stable.
5. *One more tip just for luck:* Do not try and design your own CAN controller. There are secrets that will stop you.

Keil RTX RTOS:

The Keil RTX RTOS is now available free with a BSD type license. Ports are available for Keil MDK, IAR and GCC. See www.keil.com/demo/eval/rtx.htm for more information. See http://community.arm.com/groups/processors for general information.

Other CAN Labs with experiments:

www.keil.com/appnotes/docs/apnt_247.asp

www.keil.com/appnotes/docs/apnt_236.asp

Notes on MDK:

1. **MDK-Freescale is available for $745.** Call Keil Sales for details.
2. A full feature Keil RTOS called RTX is included with MDK.
3. The two RTX Kernel Awareness windows are updated live.
4. **MQX:** An MQX port for MDK is available including Kernel Awareness windows. See www.keil.com/freescale.
5. **Processor Expert** compatible. For more information see www.keil.com/appnotes/files/apnt_235_pe2uv.pdf.

How can I learn more about CAN ?

Easy ! With a hardware board you can generate and receive real CAN messages and connect to other nodes or a CAN test analyzer. You can use the Cortex-M Serial Wire Viewer to see the CAN messages and interrupts displayed in real time. You can compile these examples with the evaluation version of the software.

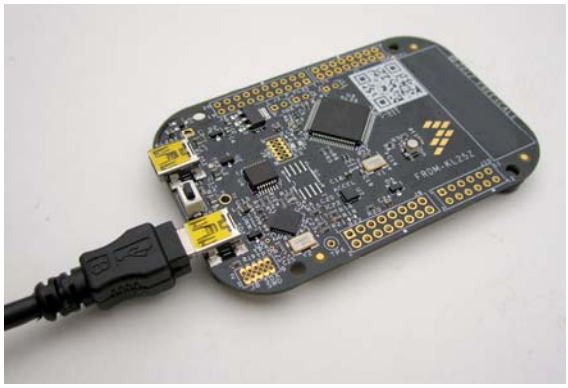
Keil completely supports the new Cortex-M0 and Cortex-M0+ processors as well as the Cortex-M3 and M4. DS-5 supports ST Cortex-A9 processors such as the iMX6. See www.arm.com/ds5. You now know how CAN works and are familiar with the Keil software and will have no problem getting a real CAN system operating.

Keil RTX™ RTOS now comes with a BSD type license. Source is provided. See www.arm.com/cmsis.

Keil MDK for Freescale:

1. **MDK-Freescale is available for $745.** See the last page.
2. A full feature Keil RTOS called RTX is included with MDK.
3. The two RTX Kernel Awareness windows are updated live.
4. **MQX:** An MQX port for MDK is available including Kernel Awareness windows. See www.keil.com/freescale.
5. **Processor Expert** compatible. For more information see www.keil.com/appnotes/files/apnt_235_pe2uv.pdf.
6. Choice of adapters: ULINK2™, ULINK-ME™, ULINK*pro*™, Segger J-Link (version 6 or later) and P&E OSJTAG.

Keil Tutorials for Freescale Boards:	**www.keil.com/freescale**
K60D100M Tower:	www.keil.com/appnotes/docs/apnt_249.asp
K60N512	www.keil.com/appnotes/docs/apnt_234.asp
KL25Z Freedom	www.keil.com/appnotes/docs/apnt_232.asp
KL20D50 Freedom	www.keil.com/appnotes/docs/apnt_243.asp
Freescale CUP Contest:	www.keil.com/appnotes/docs/apnt_257.asp



Document Resources: See www.keil.com/freescale

Books:

1. ***NEW!*** **Getting Started MDK 5:** Obtain this free book here: www.keil.com/mdk5/.
2. There is a good selection of books available on ARM processors. A good list of books on ARM processors is found at www.arm.com/university by selecting "Teaching Resources". You can also select ARM Related Books but make sure to also select the "Books suited for Academia" tab to see the full selection.
3. µVision contains a window titled Books. Many documents including data sheets are located there.
4. **A list of resources is located at:** www.arm.com/products/processors/cortex-m/index.php
   Click on the Resources tab. Or search for "Cortex-M3" on www.arm.com and click on the Resources tab.
5. The Definitive Guide to the ARM Cortex-M0/M0+ by Joseph Yiu. Search the web for retailers.
6. The Definitive Guide to the ARM Cortex-M3/M4 by Joseph Yiu. Search the web for retailers.
7. Embedded Systems: Introduction to Arm Cortex-M Microcontrollers (3 volumes) by Jonathan Valvano.

Application Notes:

1. ***NEW!*** ARM Compiler Qualification Kit: Compiler Safety Certification: www.keil.com/pr/article/1262.htm
2. **Processor Expert** compatible www.keil.com/appnotes/files/apnt_235_pe2uv.pdf.
8. Using Cortex-M3 and Cortex-M4 Fault Exceptions www.keil.com/appnotes/files/apnt209.pdf
9. Segger emWin GUIBuilder with µVision™ www.keil.com/appnotes/files/apnt_234.pdf
10. Porting mbed Project to Keil MDK™ www.keil.com/appnotes/docs/apnt_207.asp
11. MDK-ARM™ Compiler Optimizations www.keil.com/appnotes/docs/apnt_202.asp
12. Using µVision with CodeSourcery GNU www.keil.com/appnotes/docs/apnt_199.asp
13. RTX CMSIS-RTOS in MDK 5 C:\Keil_v5\ARM\Pack\ARM\CMSIS\4.1.0\CMSIS_RTX
14. Download RTX CMSIS-RTX www.keil.com/demo/eval/rtx.htm ***and*** www.arm.com/cmsis
15. Barrier Instructions http://infocenter.arm.com/help/topic/com.arm.doc.dai0321a/index.html
16. Lazy Stacking on the Cortex-M4: www.arm.com and search for DAI0298A
17. Cortex Debug Connectors: www.keil.com/coresight/coresight-connectors
18. Sending ITM printf to external Windows applications: www.keil.com/appnotes/docs/apnt_240.asp

Keil Tutorials for Freescale Boards: www.keil.com/freescale

K60D100M Tower:	www.keil.com/appnotes/docs/apnt_249.asp
K60N512	www.keil.com/appnotes/docs/apnt_234.asp
KL25Z Freedom:	www.keil.com/appnotes/docs/apnt_232.asp
KL20D50 Freedom:	www.keil.com/appnotes/docs/apnt_243.asp
Freescale CUP Contest:	www.keil.com/appnotes/docs/apnt_257.asp

ARM Community Forums: www.keil.com/forum and http://community.arm.com/groups/tools/content

Infineon Community Forum: www.infineonforums.com/

ARM University program: www.arm.com/university. Email: university@arm.com

ARM Accredited Engineer Program: www.arm.com/aae

mbed™: http://mbed.org

For comments or corrections on this document please email bob.boys@arm.com

For more information on the ARM CMSIS standard: www.arm.com/cmsis

How can trace help me find problems ?

Trace, either SWV or ETM, adds significant power to debugging efforts. Tells you where the program has been, how it got there, how long it took, when did the interrupts fire and all about data reads and writes.

- With RTOS and interrupt driven events – many programs are now asynchronous. Trace helps sort this out and provides for the dynamic analysis of running code.
- Putting test or printf code in your project sometimes changes or erases the problem. Trace is non-intrusive.
- Trace can often find nasty problems very quickly. Weeks or months can be replaced by minutes. *Really !*
- ….especially where the bug occurs a long time before the consequences are seen.
- Or where the state of the system disappears with a change in scope(s).
- Plus – you don't have to stop the program t see the trace. This is crucial to some applications.
- No trace availability is responsible for unsolved bugs – some of these problems are too hard to find without it.
- Pointer problems. Is your pointer really reading or writing what you think it is ? Where is it pointing to ?
- Illegal instructions and data aborts (such as misaligned writes).
- Code overwrites – writes to Flash, unexpected writes to peripheral registers.
- Profile Analyzer. Where is the CPU spending its time ? Where should I start to optimize my program ?
- Code Coverage. Can be a certification requirement. Was this instruction executed ?

Serial Wire Viewer and ETM Trace Summary:

Serial Wire Viewer can see:

- Global variables.
- Static variables.
- Structures.
- Peripheral registers – just read or write to them.
- Can't see local variables. (just make them global or static).
- Can't see DMA transfers – DMA bypasses CPU and CoreSight by definition.

Serial Wire Viewer displays in various ways:

- PC Samples.
- Data reads and writes.
- Exception and interrupt events.
- CPU counters.
- Timestamps for these.

These are the types of problems that can be found with a quality ETM trace:

- Pointer problems.
- Illegal instructions and data aborts (such as misaligned writes).
- Code overwrites – writes to Flash, unexpected writes to peripheral registers (SFRs), a corrupted stack.
  How did I get here ?
- Out of bounds data. Uninitialized variables and arrays.
- Stack overflows. What causes the stack to grow bigger than it should ?
- Runaway programs: your program has gone off into the weeds and you need to know what instruction caused this. Is very tough to find these problems without a trace. ETM trace is best for this.
- Communication protocol and timing issues. System timing problems.

Keil Products:

Keil Microcontroller Development Kit (MDK-ARM™) for Kinetis processors:

- MDK-Lite™ (Evaluation version) 32K Code and Data Limit - $0
- **MDK-Freescale™ For all Kinetis Cortex-M3/4, 1 year renewable term license**
- ***NEW !!*** MDK-ARM-CM™ (for Cortex-M series processors only – unlimited code limit)
- MDK-Standard™
- MDK-Professional (Includes Flash File, TCP/IP and USB driver libraries)

USB-JTAG adapter (for Flash programming too)

- ULINK2 - *(ULINK2 and ME - SWV only – no ETM)*
- ULINK-ME – sold only with a board by Keil or OEM.
- ULINK*pro* – Cortex-M*x* SWV & ETM trace

A Keil ULINK must be purchased or you can use the OS-JTAG on the Kinetis Tower board. For Serial Wire Viewer (SWV), a ULINK2, ULINK-ME or a J-Link is needed. For ETM support, a ULINK*pro* is needed.
OS-JTAG does not support either SWV or ETM debug technology.

Note: USA prices. Contact sales.intl@keil.com for pricing in other countries.

Call Keil Sales for more details on current pricing. All products are available.

For the ARM University program: go to www.arm.com/university. Email: university@arm.com

All products include Technical Support for 1 year. This can be renewed.

For the entire Keil catalog see www.keil.com, contact Keil Sales or your local distributor.

For Linux, Android, Bare Metal or other OS support for Cortex-A processors: see www.arm.com/ds5

For more information:

Keil products can be purchased directly from ARM or through various distributors.

Keil Direct Sales: In the USA: sales.us@keil.com or 800-348-8051. **Outside the USA:** sales.intl@keil.com

Keil Distributors: See www.keil.com/distis/

Keil Technical Support in USA: support.us@keil.com or 800-348-8051. **Outside the US:** support.intl@keil.com.

For comments or corrections please email bob.boys@arm.com.

For more information regarding Keil support of Freescale processors, see www.keil.com/freescale

For more information about ARM processors and products: http://community.arm.com/groups/processors

CMSIS COMPLIANT
ARM® Cortex® Microcontroller Software Interface Standard

Cortex™
Intelligent Processors by ARM®

KEIL™ Tools by ARM