Notes - Unit 12

INTRODUCTION TO CONTROLLER AREA NETWORK (CAN)

- The Controller Area Network (CAN) protocol was developed in the mid-1980s by Bosch GmbH, to provide a robust, cost-effective communications bus for automotive applications.
- We cover the version 2.0b of the CAN standard, as described in "*R. Bosch. CAN specification, v. 2.0, Stuttgart, 1991*".

OVERVIEW OF CAN

- The CAN specification, as developed by R. Bosch GmbH covers only the *Physical* and *Data Link* Layers.

PHYSICAL LAYER:
- In the Bosch CAN 2.0b standard, the description is limited to the definition of the bit timing, bit encoding, and synchronization.
- The Bosch CAN 2.0b standard does not specify the physical transmission medium, the acceptable (current/voltage) signal levels, the connectors, and other characteristics of the driver/receiver stages and the physical wiring. The system designer can choose from multiple available media technologies including twister pair, single wire, optical fiber, radio frequency, infrared, etc.

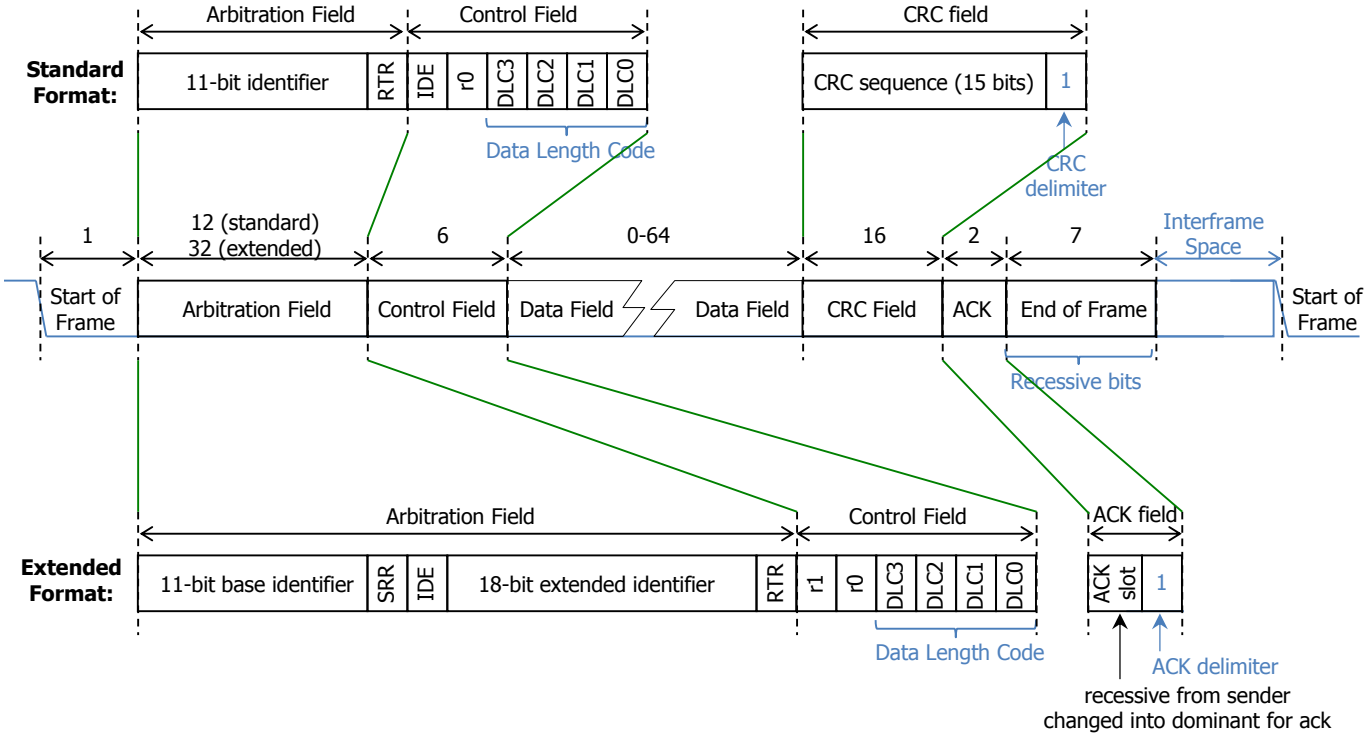
DATA LINK LAYER:
It consists of the Logical Link Control (LLC) and the Medium Access Control (MAC) sub-layers.
- **LLC Sub-layer**: It provides all the services for the transmission of a stream of bits from a source to a destination. In particular, it defines: message acceptance filtering, overload notification, and error recovery management.
- **MAC Sub-layer**: It represents the kernel of the CAN protocol. The MAC sub-layer is responsible for message framing, arbitration, acknowledgement, error detection, and signaling. For the purposes of fault containment and additional reliability, the MAC operations are supervised by a controller entity monitoring the error status and limiting the operations of a node if a possible permanent failure is detected.

GENERAL CHARACTERISTICS:
- The CAN protocol is optimized for systems that need to transmit and receive relatively small amounts of information.
- **Single channel**: The bus consists of a single bidirectional channel that carries bits. From this data, resynchronization information can be derived. The channel implementation is not specified, it can be: single wire (plus ground), two differential wires, optical fibers, etc.
- **Bus values:** The bus can have one of two complementary values: dominant (0) and recessive (1). During simultaneous transmission of dominant and recessive bits, the resulting bus value will be dominant.
- **Information Routing**: A node does not make use of any information about the system configuration (e.g.: node addresses). This has several important consequences:
  - ✓ System flexibility: Nodes may be added to the CAN network without requiring any change in the software or hardware of any node or the Application Layer.
  - ✓ Message routing: The content of a message is described by an ***identifier***. The ***identifier*** does not indicate the destination of the message, but rather the description of the meaning of the data, so that all nodes in the network are able to decide by Message Filtering whether the data is to be acted upon by them or not.
  - ✓ Multicast: As a consequence of the concept of Message filtering, any number of nodes may receive and act simultaneously upon the same message.
  - ✓ Data consistency: Within a CAN network, a message is guaranteed to be accepted simultaneously either by all nodes or by no node.
- **Arbitration**: Whenever the bus is free, any node may start to transmit a message. If two or more nodes start transmitting messages at the same time, the bus access conflict is resolved by bit-wise arbitration using the ***identifier***. The mechanism or arbitration guarantees that neither information nor time is lost. During arbitration, every transmitter compares the level of the bit transmitted with the level that is monitored on the bus. If these levels are equal, the node may continue to send. When a recessive bit (1) is sent, but a dominant bit (0) is monitored, the node has lost arbitration and must withdraw without sending any further bits.
- **Multi-master**: When the bus is free any node may start to transmit a message. The node with the message of highest priority (determined by the ***identifier***) to be transmitted gains bus access.
- **Error detection**: To detect errors, the following measures have been taken:
  - ✓ Monitoring: Each transmitter compares the bit levels detected on the bus with the bit levels being transmitted.
  - ✓ Cyclic Redundancy Check (CRC): For every message, CRC is calculated and the checksum is appended to the message.
  - ✓ *Bit-Stuffing*: CAN is an asynchronous protocol (clock information embedded in the message rather than transmitted as a separate signal). A message including a long sequence of identical bits could cause a synchronization problem. Thus, the CAN protocol uses bit stuffing: every 5 identical bits, a complemented bit is included.
- **Fault confinement**: CAN nodes are able to distinguish between short disturbances and permanent failures. Defective nodes are switched off.

CAN MESSAGE FORMAT

- This is specified by the Data Link Layer.
- In CAN, there are 4 different types of frame, according to their content and function:
  - ✓ **Data Frames**: They contain data information from a source to possibly multiple receiver
  - ✓ **Remote Frames**: They are used to request transmission of a corresponding (with the same identifier) Data Frame
  - ✓ **Error Frames**: They are transmitted whenever a node on the network detects an error.
  - ✓ **Overload Frames**: They are used for flow control, to request an additional time delay before the transmission of a Data Frame or Remote Frame.
- Data frames and remote frames are separated from preceding frames by an interframe space. Applications do not need to send or handle error and overload frames.

DATA FRAME



- **Start of Frame Field**: Single dominant bit that marks the beginning of the Data Frame (or Remote Frame). A node is allowed to start transmission only when the bus is idle. All nodes have to synchronize to the edge of the node that starts transmission first.
- **Arbitration Field**: The format is different for the Standard Format and Extended Format frames.
  - ✓ Standard Format: The Arbitration Field consists of the 11 bit identifier followed by the RTR Bit.
  - ✓ Extended Format: The Arbitration Field consists of the 29 bit identifier, divided into Base identifier (11 bits) and extender identifier (18 bits). It also includes the SRR Bit, the IDE bit, and the RTR bit.
  - ✓ The ***identifier*** bits are transmitted with the MSB first. The most significant 7 bits cannot all be recessive (1).
  - ✓ RTR (Remote Transmission Request) Bit: It is dominant (0) in Data Frames. For a Remote Frame, it is recessive (1).
  - ✓ SRR (Substitute Remote Request) Bit: It is a recessive (1) bit. SRR occupies the position of RTR in the Standard Format. As a result, collisions between a Standard Frame and an Extended Frame (assuming Base IDs to be identical) are resolved in such a way that the standard frame prevails over the extended frame (this is because RTR=0 and SRR=1).
  - ✓ IDE (Identifier Extension) Bit: It belongs to the arbitration bit for extended format and to the control field for the standard format. The IDE is a dominant (0) bit for the standard format and a recessive (1) bit for the extended format.
- **Control Field**: The Standard Format includes the IDE bit (dominant), the r0 bit (dominant), and the Data Length Code (4 bits). The Extended Format includes r_1 bit (dominant), r_0 bit (dominant), and the Data Length Code (4 bits).
  - ✓ Data Length: It specifies the number of bytes (from 0: 0000 to 8: 1000) contained in the Data Field.
- **Data Field**: This is the actual data. It may contain 0 to 8 bytes, where each byte is transferred with the MSB first.
- **CRC Field**: Cyclic Redundancy Check. A 15-bit CRC check value is appended followed by a recessive bit (delimiter).

A $k-bit$ message can be represented as: $m_{k-1}m_{k-2}\dots m_1m_0$. The message consists of the de-stuffed Start-of-Frame, Arbitration, Control, and Data (if present) fields. The associated polynomial $M(x)$ (of order $k-1$) is given by:

$$M(x) = m_{k-1}X^{k-1} + m_{k-2}X^{k-2} + \cdots + m_1X + m_0$$

An $n-bit$ CRC value $r_{n-1}r_{n-2}\dots r_1r_0$ is appended to the message, resulting in: $m_{k-1}m_{k-2}\dots m_1m_0r_{n-1}r_{n-2}\dots r_1r_0$

The CRC value has an associated polynomial $R(x) = r_{n-1}X^{n-1} + r_{n-2}X^{n-2} + \cdots + r_1X + r_0$. $R(x)$ can be obtained as the remainder of the modulo-2 division of the following polynomials:

$$\frac{X^n M(x)}{G(x)}$$

- ✓ $G(x)$: Generator polynomial of order n with non-zero highest and lowest-order coefficients. It is of order n because the quotient must be of lower or equal order than $M(x)$. $g_n g_{n-1} \ldots g_1 g_0$. $G(x) = g_n X^n + g_{n-1}X^{n-1} + \cdots + g_1 X + g_0$.
- ✓ $X^n M(x)$ can be seen as the associated polynomial of the message to which n zeros were appended.

- ✓ CAN uses CRC-15-CAN, where $n = 15$ and $G(x) = X^{15} + X^{14} + X^{10} + X^8 + X^7 + X^4 + X^3 + 1$. A k-bit CAN message includes a CRC-15 check value. This error correcting code is generated by the transmitter and sent to the receiver. The receiver generates the CRC-15 value again and compares it against the received CRC-15. If they do not match, a CRC error will be issued. Note that the same 15-bit value will result from different messages and as such, there might be cases in which the transmitted and received messages differ but both generate the same CRC-15 value. Nevertheless, this scheme is very robust. The selection of the particular $G(x)$ is beyond the scope of this course.
- ✓ Ethernet uses CRC-32, where $n = 32$ and $G(x) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^4 + X^2 + X + 1$. Also, the simple even parity bit is a trivial CRC case, called CRC-1 where $n = 1$ and $G(x) = X + 1$.

Example: $m = 11100110,\ M(x) = X^7 + X^6 + X^5 + X^2 + X,\ G(x) = X^4 + X^3 + 1$

$\rightarrow n = 4$. The remainder $R(x)$ is such that: $\frac{X^n M(x)}{G(x)} = Q(x) + \frac{R(x)}{G(x)}$, where $order\big(R(x)\big) < order\ (G(x))$

$$\frac{X^4(X^7 + X^6 + X^5 + X^2 + X)}{X^4 + X^3 + 1} = \frac{X^{11} + X^{10} + X^9 + X^6 + X^5}{X^4 + X^3 + 1} = X^7 + X^5 + X^4 + X^2 + X + \frac{X^2 + X}{X^4 + X^3 + 1}$$

Thus: $R(x) = X^2 + X$, and its CRC-4 is 0110.

- **ACK Field**: It consists of 2 bits. The ACK slot records acknowledgements from receivers. The other bit is the delimiter (recessive bit). The acknowledgement is recorded in the ACK slot by letting the receiver overwrite the recessive bit (1) sent by the transmitter with a dominant bit (0). This is possible since the CAN bus is a wired AND channel connecting all nodes.
- **End of Frame Field**: Each data frame and remote frame is delimited by a sequence of 7 recessive (1) bits.

REMOTE FRAME

A Remote frame is used to request the transmission of a message with a given identifier from a remote node:

- The identifier is used to indicate the identifier of the requested message.
- The data field is empty (0 bytes).
- The Data Length Code field indicates the data length of the requested message (not the transmitted one).
- The RTR bit is set to recessive (1).

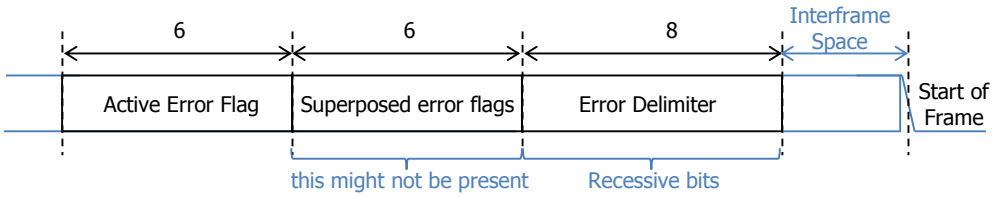
ERROR FRAME

There are five different, non-mutually exclusive, error types:

- Bit Error: A node sending a bit is also monitoring it. A bit error has to be detected during that time, when the value monitored is different from the transmitted bit value. The exceptions are i) the recessive bits sent during the stuffed bitstreams of the arbitration process, ii) the ACK slot (a recessive is changed into a dominant), and iii) a transmitter that sends a passive-error flag and detects a dominant bit.
- Stuff Error: It is detected if 6 consecutive bits are identical in the message fields subject to stuffing.
- CRC Error: The receiver computes the CRC of the message and it differs from the one sent by the transmitter.
- Form Error: When a particular field contains one or more illegal bits.
- Acknowledgment Error: This is detected by the transmitter if the recessive bit transmitted (on the ACK slot) is not modified by the Receiver to a dominant bit.

The Error Frame consists of a superposition of error flags, transmitted from different nodes, possibly at different times, followed by an Error Delimiter field (8 recessive bits). The CAN node destroys the message by sending the Error Frame so that the transmitter can resend the message.

- Error active node: This node signals an error condition by transmitting an active-error flag (6 consecutive dominant bits).
- Error passive node: This node signals an error condition by transmitting a passive error flag (6 consecutive recessive bits).



A node transmits an error flag (6 bits) immediately after it detects an error. The error flag violates normal transmission rules (bit stuffing or destroys the form of ACK field or End-of-Frame field) and so it is detectable by other nodes, who can concurrently transmit their own error frames (which can superpose in time). The total length of this sequence varies 6 (only the initial error flag to 12 bits (other superposed error flags).
After the transmission of an Error Flag each node sends recessive bits and monitors the bus until it detects a recessive bit. Afterwards, it starts transmitting 7 more recessive bits. The node can then attempt transmission of regular CAN frames.
A CAN node keeps track of how many times it received or transmitted error frames using error counters. When these error counters reach certain limits the CAN node is first partially disabled (set as error-passive node) and then totally disabled. This avoid the possibility that a broken node is disabling the bus by constantly generating error frames. There are specific rules about how the error counters get decremented or incremented (see CAN specification).

OVERLOAD FRAME
It contains two fields: Overload flag and Overload delimiter (8 recessive bits).
There are three kinds of Overload condition which lead to the transmission of an Overload flag:
- The internal conditions of the receiver are such that it requires a delay of the next Data Frame (or Remote Frame).
- On detection of a dominant bit (0) during the Intermission of the Interframe Space.
- If a CAN node samples a dominant bit (0) at the 8^{th} bit (last bit) of an Error Delimiter of Overload Delimiter.

At most, two Overload frames may be generated to delay the next Data Frame or Remote Frame.

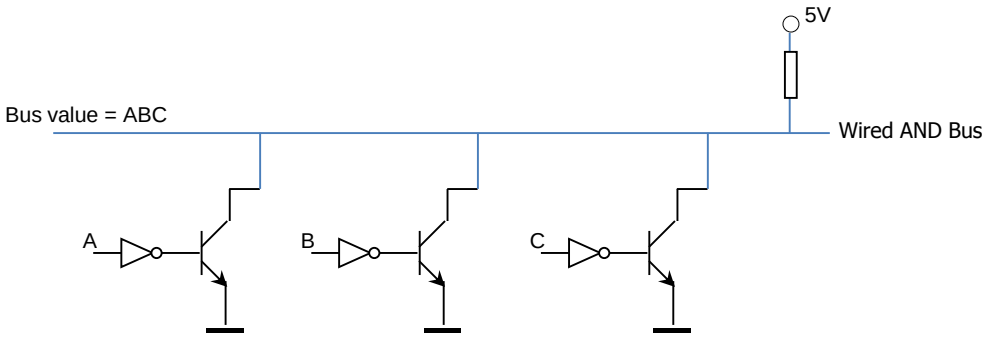
INTERFRAME SPACE
- Data frames and remote frames are separated from preceding frames (any frame type) by the Interframe space.
- Overload frames and error frames are not preceded by an interframe space
- Multiple overload frames are not separated by an interframe space.

- Interframe space: Intermission and Bus Idle. This is for nodes that are not error-passive or have been receivers of a previous message.
- Interframe space: Intermission, Suspend Transmission, and Bus Idle. This is for error-passive node that have been the transmitter of the previous message.

- Intermission: 3 recessive bits (0). During intermission, no node is allowed to start transmission of a Data Frame or Remote frame. The only action permitted is signaling of an Overload condition.
- Bus Idle: This might be of arbitrary length. The bus is recognized to be free, and any node having something to transmit can access the bus. A message, pending during the transmission of another message, is started in the first bit following Intermission. The detection of a dominant bit (0) on the bus is interpreted as Start of Frame.
- Suspend Transmission: After an error-passive node has transmitted a frame, it sends 8 recessive bits following intermission, before starting to transmit a further message or recognizing the bus to be idle.

BIT STUFFING
Whenever a transmitter detects 5 consecutive identical bits, it inserts a complemented bit. This is applied only to the start-of-frame field, arbitration control, data and CRC field.

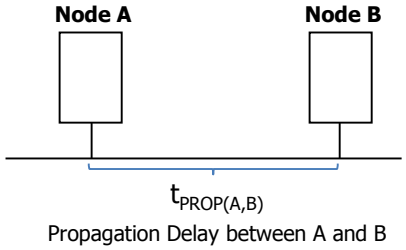
BUS ARBITRATION
- The CAN arbitration protocol is both priority-based and non-preemptive (a message being transmitted cannot be preempted by higher priority messages that were queued after the transmission has started). The CAN bus is essentially a wired AND channel connecting all nodes.
- If a node wishing to transmit finds the bus in an idle state, it starts an arbitration phase by issuing a start-of-frame bit. At this point, each node with a message to be transmitted can start racing to grant access of the bus by serially transmitting the ***identifier*** bits of the message in the arbitration field, starting from the MSB. Collisions among identifier bits are resolved by logical AND: if a node notices that its identifier bits have not been changed, it realizes that it is the winner of the contention and it is granted access for transmitting the rest of the message while the other nodes switch to listening mode. If however, if one bit is changed when the node reads it back, it means that there is a higher priority (dominant bit) node and thus the node message is withdrawn. The identifier specifies the priority: the lower the value, the higher the priority.



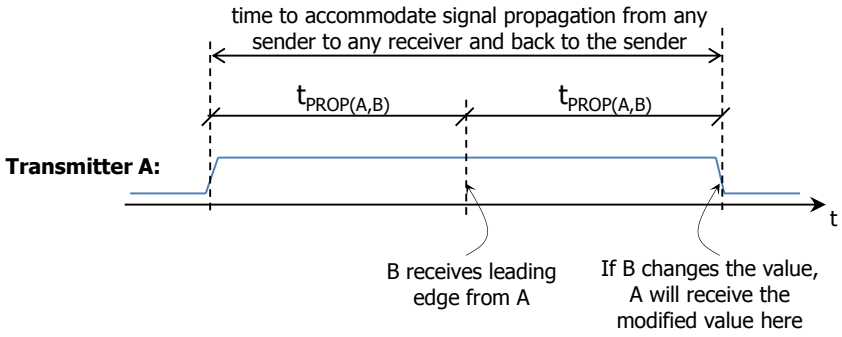
CAN BIT TIMING

This is defined by the Physical Layer.

- Nodes are requested to be synchronized on the bit edges so that every node agrees on the value of the bit currently transmitted on the bus. To do so, each node implements a *synchronization protocol* that keeps the receiver bit rate aligned with the actual rate of the transmitted bits. The synchronization protocol uses transition edges to resynchronize nodes. Long sequences without bit transitions should be avoided. This is why the protocol employs 'bit stuffing': every 5 consecutive identical bits, a complemented bit is included.
- Signal type: Non Return to Zero (NRZ) encoding. Bits are encoded as "recessive" (typically 0), and "dominant" (typically 1).
- The bit time includes a propagation delay segment. This considers the signal propagation on the bus and the signal delays caused by the transmitting and receiving nodes. It should be long enough to accommodate signal propagation from any sender to any receiver and back to the sender. In practice, this is determined by the two nodes within the system that are farthest apart from each other:



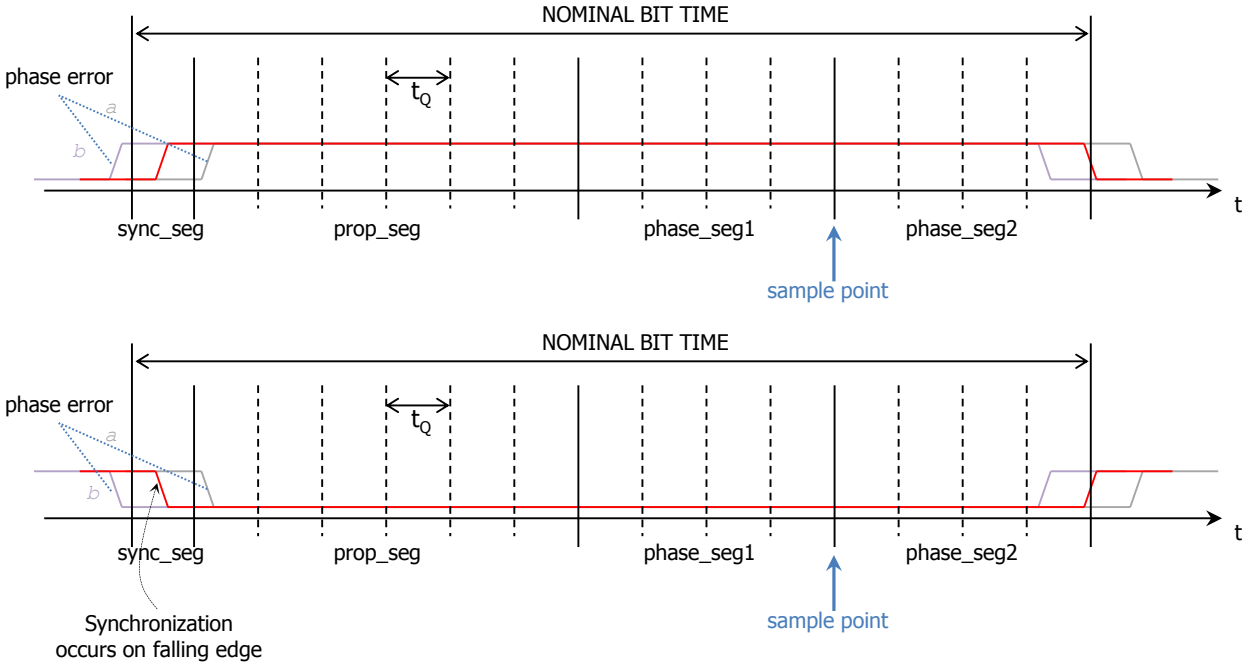
Propagation Delay between A and B



Notice that we consider the case when the receiver changes the value of a bit (like the ACK slot).

- Time Quanta: Units in which the Bit time can also be expressed.
  $f_{CAN} = \frac{E-clock}{M}$, M: Pre-scale factor
  $T_{CAN} = Time\ quantum = t_Q = M \times T_{E-clock}$
  $T_{E-clock}$: Also known as the minimum time quantum

- In practice, we must allow for more time than the propagation delay segment. As a result, the bit time includes four non-overlapping time segments (including the propagation delay segment), as depicted in the figure below.



- The nominal Bit Time results in: $T_{NBT} = t_{SYNC\_SEG} + t_{PROP\_SEG} + t_{PHASE\_SEG1} + t_{PHASE\_SEG2}$
- The nominal Bit Rate is defined to be the number of bits transmitted per second:

$$Nominal\ Bitrate = f_{NBT} = \frac{1}{T_{NBT}},$$

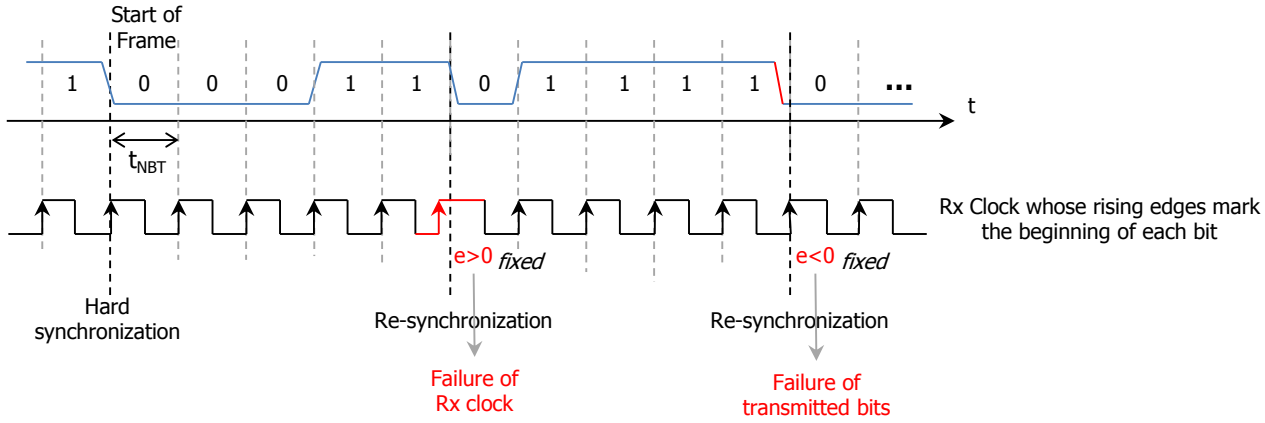
- S*ample point*: It is the point in time at which the bus level is read and interpreted as the value of that respective bit. The sample point is located at the end of *phase_seg1*.
- *Information processing time*: It is the time required to convert the electrical state of the bus, as read at the *sample point,* in the corresponding bit value. It is less than or equal to $2 \times t_Q$. In the HCS12 CAN module, it is fixed at $2 \times t_Q$.

  - ✓ Synchronization segment (*sync_seg*): This is a reference interval, used for synchronization purposes. The leading edge of a bit is expected to lie within this segment. It is 1 time quantum long.
  - ✓ Propagation segment (*prop_seg*): This part of the bit time is used to compensate for the (physical) propagation delays within the network. It is twice the sum of the signals propagation time on the bus line and on the transmitting and receiving nodes. $t_{PROP\_SEG} = 2 \times t_{PROP(A,B)}$, $t_{PROP(A,B)} = t_{BUS} + t_{Tx} + t_{Rx}$. t_{PROP_SEG} is programmable from 1 to 8 time quanta long.
  - ✓ Phase segment 1 (*phase_seg1*): It is used to compensate for a phase error in the position of the bit edge. A delay in the bit edge is compensated by increasing the length of *phase_seg1*. In the HCS12 CAN module, *phase_seg1* is programmable from 1 to 8 time quanta long.
  - ✓ Phase segment 2 (*phase_seg2*): It is used to compensate for a phase error in the position of the bit edge. An anticipation in the bit edge is compensated by decreasing the length of *phase_seg2*. In the HCS12 CAN module, *phase_seg2* = max(*phase_seg1, information processing time*). This is between 2 and 8 quanta long.

CAN SYNCHRONIZATION

- The beginning of each received bit must occur during each node's *sync_seg* segment. This is achieved by synchronization, which is required because of phase errors between nodes. Two types of synchronization exist; notice that this only happens during a recessive (1) to dominant (0) bit transition.
- **Hard synchronization:** It takes place at the beginning of the frame, when the start of frame bit changes the state of the bus from recessive (1) to dominant (0). Upon detection of the edge, the bit time is restarted at the end of the *sync_seg* segment. Thus, the edge of the start bit lies within the *sync_seg* segment of the restarted bit time.
- **Re-synchronization:** It is subsequently performed during the remainder of the message frame whenever a change of bit value from recessive (1) to dominant (0) occurs outside of the expected *sync_seg* segment. The phase segments are lengthened (*phase_seg1*) or shortened (*phase_seg2*).
  There are three possibilities for the occurrence of the incoming recessive-to-dominant edge. In each case, there is a phase error 'e' defined as the position of the edge relative to *sync_seg* measured in time quanta. The sign is defined by:
  - ✓ e >0: After the *sync_seg* segment, but before the sample point: This is case 'a' in the previous figure and it is interpreted as a late edge. The node will attempt re-synchronization by increasing the duration of *phase_seg1* by the number of time quanta by which the edge was late, up the Re-synchronization jump width limit.
  - ✓ e < 0: After the sample point but before the *sync_seg* segment of next bit: This is case 'b' in the previous figure and it is interpreted as an early bit. The node will attempt re-synchronization by decreasing the duration of *phase_seg2* by the number of time quanta by which the edge was early, up to the Re-synchronization jump width limit.
  - ✓ e = 0: Within the *sync_seg* segment of the current bit time: Here, there is no synchronization error. This is the red wave in the previous figure.

  Re-synchronization Jump Width (RJW): It is the maximum amount of time quanta by which we can increase *phase_seg1* and decrease *phase_seg2*. This is programmable and given by: $RJW \in [1, \dots, \min(4, \text{phase\_seg1})]$



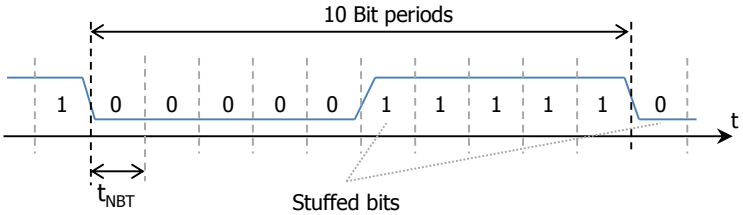
Setting CAN timing parameters

- T_{NBT}: Nominal Bit Time (units of time). NBT: Nominal Bit Time (in time quanta).
- The nominal bit rate of a CAN Network is uniform throughout the network and given by: $f_{NBT} = {}^{1}/_{T_{NBT}}$

$$t_{PROP\_SEG} = 2 \times t_{PROP(A,B)}, \qquad t_{PROP(A,B)} = t_{BUS} + t_{Tx} + t_{Rx}$$

- $prop\_seg = \left\lceil \frac{t_{PROP\_SEG}}{t_Q} \right\rceil$. $prop\_seg$ is defined in terms of time quanta (t_Q) that must be allocated to this segment.

- In the worst-case scenario, we wait for 10 bit periods for re-synchronization. This happens if 5 dominant bits (which follow a recessive bit) are followed by 5 recessive bits and then another dominant bit. This represents the worst-case condition for the accumulation of phase error.



  This must be compensated for by resynchronization and the time must be less than the programmed RJW width time (t_{RJW}). Note that $t_{RJW} \le min(t_{PHASE\_SEG1}, t_{PHASE\_SEG2})$. The accumulated error is due to tolerance in the CAN clock. The CAN clock has a period t_Q.

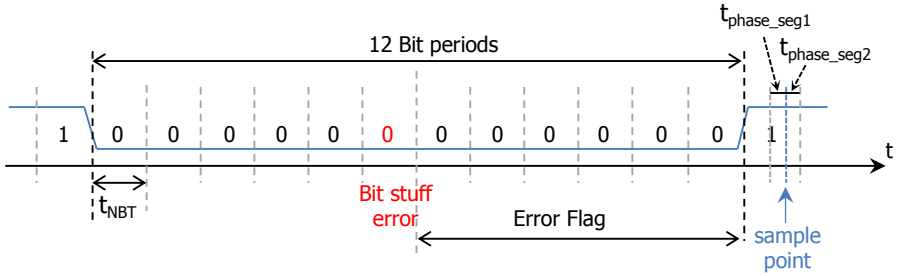
$$2 \times \Delta f \times 10 \times t_{NBT} < t_{RJW}$$

  Δf is the largest frequency variation (in percentage) of all CAN nodes in the network (we consider both transmitter and receiver, that's why Δf is multiplied by 2).

- The error flag from an error-active node consists of 6 dominant bits. There can be up to 6 dominant bits before the error flag (e.g. a bit stuffing error). This would make 12 dominant bits. After the error flag, there could be an extra six dominant bits (from the superposed error flags of other nodes) or an error delimiter (8 recessive bits). To detect this, we need to correctly sample the 13th bit after the last resynchronization. The accumulated error during all that time is equal to 12 bit times plus the time to sample the 13th bit: $13 \times t_{NBT} - t_{PHASE\_SEG2}$. Due to tolerance of the CAN system clock, the maximum accumulated error must be smaller than both t_{PHASE_SEG1} *and* t_{PHASE_SEG2}, otherwise we could not correct it with t_{RJW}:

$$2 \times \Delta f \times (13 \times t_{NBT} - t_{PHASE\_SEG2}) < min(t_{PHASE\_SEG1}, t_{PHASE\_SEG2})$$

  The figure below depicts a case with a bit stuffing error followed by an error flag, subsequently followed by the error delimiter.



Procedure for determining the optimum bit timing parameters:
These parameters must satisfy the requirements for proper bit sampling.

1. Determining the minimum $t_{PROP\_SEG} = 2 \times (t_{BUS} + t_{Tx} + t_{Rx})$.
2. Choose a CAN clock frequency, i.e., the pre-sale factor M. $f_{CAN} = \frac{E-clock}{M}$, $t_Q = M \times T_{E-clock}$
   Note that t_Q is picked such that the Nominal Bit Time is between 8 and 25 time quanta.
3. Calculate $prop\_seg$. Note that it cannot be greater than 8, otherwise we need to pick another pre-sale factor M
4. Determine $phase\_seg1$ *and* $phase\_seg2$. If $phase\_seg1 + phase\_seg2 < 3$, we need to pick a different pre-sale factor M. We prefer to assign the same value to both $phase\_seg1$ *and* $phase\_seg2$, so if the summation is odd, we subtract 1 and add it back to $prop\_seg$. The exception is when $phase\_seg1 + phase\_seg2 = 3$; here we assign $phase_{seg1} = 1,\ phase\_seg2 = 2$.
5. Determine $RJW \in [1, \dots, \min(4, \text{phase\_seg1})]$. We usually pick $RJW = \min(4, \text{phase\_seg1})$
6. Calculate the required oscillator tolerance Δf.

Example:
Calculate the CAN bit time and the segments for the following system constraints:
E-clock= 24 MHz Bit rate = 100 kbps
Bus length = 25 m Bus propagation delay = $5 \times 10^{-9}\ ^{S}/_{m}$
Transmitter (MCP2551 Transceiver) plus receiver propagation delay = 150 ns at 85 ºC

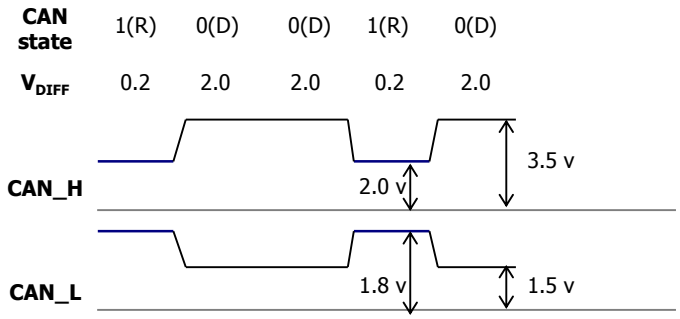
- $Bit\ Time = t_{NBT} = \frac{1}{100\ kbps} = 10us$
- $Bus\ delay = 25m \times (5 \times 10^{-9}\ ^{S}/_{m}) = 125ns \rightarrow t_{PROP\_SEG} = 2 \times (125 + 150) = 550ns$
- Pre-scaler: let's start with $M = 12$: $t_Q = 12 \times \frac{1}{24MHz} = 500ns$. Then: $NBT = \frac{10us}{500ns} = 20$
  $\rightarrow prop\_seg = \left\lceil \frac{t_{PROP\_SEG}}{t_Q} \right\rceil = \left\lceil \frac{550}{500} \right\rceil = 2. \rightarrow sync\_seg + prop\_seg + phase\_seg1 + phase\_seg2 = Bit\ Time = 20$
  $\rightarrow phase\_seg1 + phase\_seg2 = 20 - 2 - 1 = 17 > 16\ (maximum\ possible\ sum\ of\ phase\_seg1\ and\ phase\_seg2)$
- We then need a larger pre-scaler. Let's pick $M = 24$: $t_Q = 24 \times \frac{1}{24MHz} = 1us$. Then: $NBT = \frac{10us}{1us} = 10$
  $\rightarrow prop\_seg = \left\lceil \frac{t_{PROP\_SEG}}{t_Q} \right\rceil = \left\lceil \frac{550}{1000} \right\rceil = 1. \rightarrow sync\_seg + prop\_seg + phase\_seg1 + phase\_seg2 = Bit\ Time = 10$
  $\rightarrow phase\_seg1 + phase\_seg2 = 10 - 1 - 1 = 8. \quad \rightarrow phase\_seg2 = phase\_seg2 = 4$

- Then $RJW = \min(4, phase\_seg1) = 4$, $t_{RJW} = RJW \times t_Q$
- $2 \times \Delta f \times 10 \times t_{NBT} < t_{RJW}$: $2 \times \Delta f \times 10 \times 10us < 4 \times 1us \rightarrow \Delta f < \frac{4}{200} \rightarrow \Delta f < 2\%$
- $2 \times \Delta f \times \left(13 \times t_{NBT} - t_{PHASE\_SEG2}\right) < min\left(t_{PHASE\_SEG1}, t_{PHASE\_SEG2}\right)$

  $2 \times \Delta f \times (13 \times 10us - 4 \times 1us) < 4 \times 1us \rightarrow \Delta f < \frac{2}{126} \rightarrow \Delta f < 1.59\%$
- Thus, $\Delta f < 1.59\%$
- In summary: $M = 24,\ NBT = 10, sync\_seg = 1,\ prop\_seg1 = 1, phase\_seg1 = 4, phase\_seg2 = 4, RJW = 4,\ \Delta f = 1.59\%$

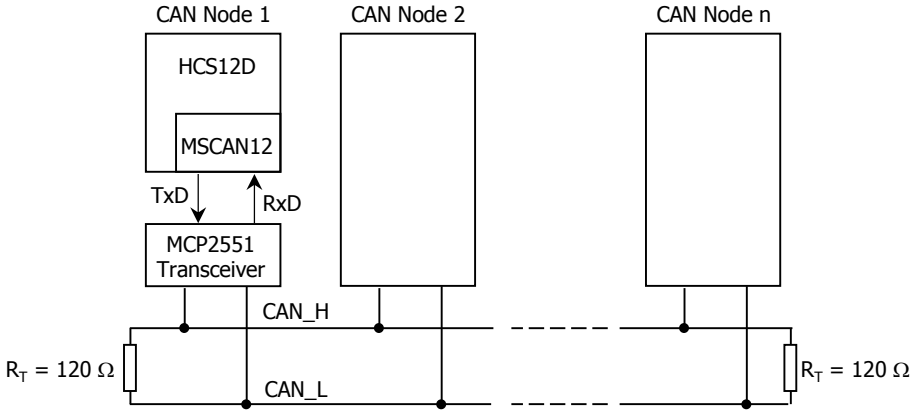
PHYSICAL CAN BUS CONNECTION

- The medium for transmission is not specified in the CAN Protocol. A typical CAN bus system is setup such as differential data transmission is used. Each node connects to the bus using two terminals: CAN_H and CAN_L, which provide differential receive and transmit capabilities. However, note that there is only one signal that is being transmitted or received over the CAN_H and CAN_L terminals.
  The two states of dominant (logic 0) and recessive (logic 1) are represented by the CAN_H and CAN_L voltage levels. This signaling method is fundamental both to the node arbitration and inherent prioritization of messages with lower IDs.

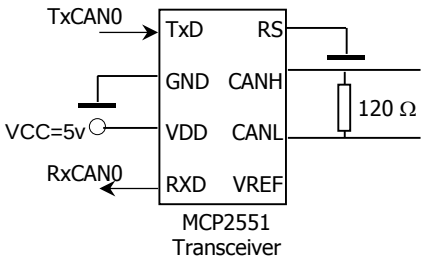


  $V_{DIFF} = V_{CAN\_H} - V_{CAN\_L}$
  Recessive state (logic 1): $V_{DIFF} \leq 0.5v$. Here, the driver outputs on the nodes must be in high impedance.
  Dominant state (logic 0): $V_{DIFF} \geq 0.9v$. Here, when CAN_H is high (3.5V for example) and CAN_L is low (1.5 V for example).

- **Dragon12-Light Board**: The HCS12D CAN module generates digital signals TxD (output) and RxD (input). The MCP2551 transceiver converts these digital signals to differential terminals CAN_H and CAN_L. The resistor R_T is a terminating resistor (120Ω) and it is included in the Board, which means that the MCU must be located on either end of the CAN bus line.



On Dragon12-Light Board:



OVERVIEW OF THE HCS12 CAN MODULE

The HCS12DG256 device inside the Dragon12-Light Board includes:

- 2 CAN modules, called MSCAN12V2 modules: CAN0 and CAN1. They implement the CAN 2.0A/B protocol.
- CAN0: The signals RxCAN0 and TxCAN0 are connected to PM0 and PM1 respectively.
- CAN1: The signals RxCAN1 and TxCAN1 are connected to PM2 and PM3 respectively.

MSCAN REGISTER ORGANIZATION

Each MSCAN module (n=0,1) occupies 64 bytes of I/O space:

- 12 Control Registers: `CANnCTL0`, `CANnCTL1`, `CANnBRT0`, `CANnBTR1`, `CANnRFLG`, `CANnRIER`, `CANnTFLG`, `CANnTIER`, `CANnTARQ`, `CANnTAAK`, `CANnTBSEL`, `CANnIDAC`.
- 2 Error Counters: `CANnRXERR`, `CANnTXERR`.
- 16 Identifier Filter Registers: `CANnIDAR0-7`, `CANnIDMR0-7`
- Receive buffer (16 bytes): `CANnRXFG`
- Transmit buffer (16 bytes): `CANnTXFG`
- **Outline of Receive and Transmit Buffers**: Identifier Registers 0-3, Data Segment Registers 0-7, Data Length Register, Transmit buffer priority register (only for transmit buffer), time stamp high byte, time stamp low byte.

Dragon12-Light Board: Only CAN0 is connected to the MCP2551 transceiver.

- MSCAN Clock Source: It depends on bit 6 (CLKSRC) of `CANnCTL1`. If CLKSRC=1, the MSCAN clock source is E-clock (24 MHz in Dragon12-Light Board). If CLKSRC=0, the MSCAN clock source is the oscillator clock (8 MHz in Dragon-12 Light Board). So, if CLKSRC=1, then: $f_{CAN} = \frac{E-clock}{M}$, M: Pre-scale factor. $T_{CAN} = Time\ quantum = t_Q = \frac{M}{E-clock}$

- Recall the formula: $T_{NBT} = t_Q \times (SYNC\_SEG + PROP\_SEG + PHASE\_SEG1 + PHASE\_SEG2)$
  In the MSCAN Module, the formula, while the same, uses different terms (and the parameters are specified in time quanta): $T_{NBT} = t_Q \times (1 + TimeSeg1 + TimeSeg2), \quad TimeSeg1 = PROP\_SEG + PHASE\_SEG1, TimeSeg2 = PHASE\_SEG2$
  `CANnBTR1`: Configures TimeSeg1 and TimeSeg2. It also configures samples per bit (1 or 3).
  `CANnBTR0`: Configures Synchronization Jump Width (RJW), and the Pre-scale Factor M (6 bits)

Interrupt Capability: Wake-up, Error Interrupts (receiver overrun, error, warning, bus-off), Receive, and Transmit.

MSCAN INITIALIZATION:
- **Out of Reset**:
  - ✓ Enable CAN module by setting CANE bit of CANnCTL1 to 1.
  - ✓ Request to enter Initialization Mode by setting INITRQ bit of CANnCTL0 to 1.
  - ✓ Wait until Initialization Mode is entered by waiting until the INITAK bit of CANnCTL1 is 1.
  - ✓ Configure CAN parameters: Write to configuration registers (CANnCTL1, CANnBTR0, CANnBTR1, CANnIDAC, CANnIDAR0-7, CANnIDMR0-7) in Initialization Mode (both INTRQ and INITAK bits are '1').
  - ✓ Clear INITRQ bit of CANnCTL0 to leave Initialization Mode and enter Normal Mode.
- **Normal Mode:**
  - ✓ Make sure MSCAN transmission queue is empty and bring the module into sleep mode by asserting the SLPRQ bit (CANnCTL0 register) and waiting for the SLPAK bit (CANnCTL1 register) to be '1'.
  - ✓ Enter the Initialization Mode
  - ✓ Configure CAN parameters: Write to the Configuration Registers in Initialization Mode.
  - ✓ Clear INITRQ bit of CANnCTL0 to leave the Initialization Mode and enter Normal Mode.

Example: Configure MSCAN0 after reset:
- Enable Wake-up
- Disable time-stamping
- Select Bus Clock (E-clock=24 MHz) as the clock source to the MSCAN.
- Disable loopback mode, disable listen-only mode
- One sample per bit
- Accept messages with extended identifiers that start with T1 and P1 (use two 32 bit acceptance filters).
- $M = 24,\ NBT = 10, sync\_seg = 1,\ prop\_seg1 = 1, phase\_seg1 = 4, phase\_seg2 = 4, RJW = 4,\ \Delta f = 1.59\%$

```
CAN0CTL1 |= 0x80; // Enable CAN, required after reset, disable loopback mode, disable listen-only mode
CAN0CTL0 |= 0x01; // Request to Enter Initialization Mode
while (!(CAN0CTL1&0x01)); // wait until Initialization Mode is entered

/* Configure CAN Parameters */
CAN0CTL1 = 0xC4; // Enable CAN0, select Bus Clock as MSCAN clock source
CAN0BTR0 = 0xD7; // Set RJW (or SJW) to 4, set pre-scaler to 24
CAN0BTR1 = 0xB4; // Set phase_seg2=4, phase_seg1=4, prop_seg=1. Set 1 sample per bit

/* Set acceptance identifier T1 . 'T' = 0x54, '1': 0x31. Bits 4 and 3 of IDAR1 are "11" (SRR,IDE) */
CAN0IDAR0 = 0x54; CAN0IDAR1 = 0x3C; CAN0IDAR2 = 0x40; CAN0IDAR3 = 0x00;

/* Set acceptance mask for T1. '0': match corresponding acceptance code register and ID bits, '1': ignore */
CAN0IDMR0 = 0x00; CAN0IDMR1 = 0x00; CAN0IDMR2 = 0x3F; CAN0IDMR3 = 0xFF;

/* Set acceptance identifier P1. 'P' = 0x50, '1': 0x31. Bits 4 and 3 of IDAR5 are "11" (SRR,IDE) */
CAN0IDAR4 = 0x50; CAN0IDAR5 = 0x3C; CAN0IDAR6 = 0x40; CAN0IDAR7 = 0x00;

/* Set acceptance mask for P1. '0': match corresponding acceptance code register and ID bits, '1': ignore */
CAN0IDMR4 = 0x00; CAN0IDMR5 = 0x00; CAN0IDMR6 = 0x3F; CAN0IDMR7 = 0xFF;

CAN0IDAC = 0x00 // Select two 32-bit filter mode.
CAN0CTL0 = 0x25 // Stop clock on wait mode, enable wake up
CAN0CTL0 &= ~(0x01) // Exit initialization mode (clear INITRQ bit)
```