

Motorola Semiconductor Engineering Bulletin

Frequently Asked Questions and Answers M68HC05 Family MCAN Module

Contents

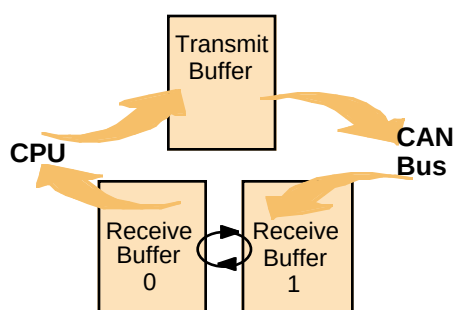
- Does the M68HC05 Family MCAN (Motorola controller area network) module use the BasicCAN or the FullCAN principle?
- How does the Ping-Pong principle of the two receive buffers work?
- Which bit time should be chosen for a network?
- Which resynchronization mode should be chosen?
- How is the HC05X microcontroller connected to an external CAN driver?
- Can the M68HC05 Family be used in a network which runs extended CAN frame format?
- How are the MCAN module registers initialized?
- When should and how should the single-line mode be used?
- What are the considerations when putting the MCAN module into sleep mode?
- After the MCAN module is put into sleep mode, why doesn't the oscillator stop?
- How should an overrun condition be handled?
- How should a bus error or bus off condition be handled?
- What does an MCAN interrupt handler look like?
- How is a message aborted?



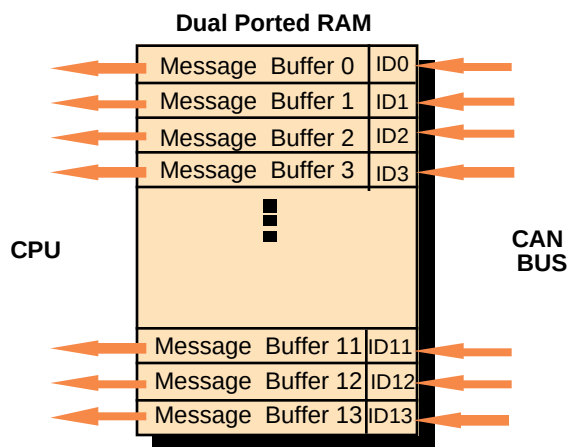
Does the M68HC05 Family MCAN module use the BasicCAN or the FullCAN principle?

The MCAN module's message buffers are implemented as a BasicCAN structure. Three buffers are implemented in RAM — one transmit buffer and two receive buffers — and each is accessible by either the CPU or the CAN bus at the same time. Because the module has two receive buffers, its software can read a received message while another message is being received from the CAN bus. (Refer to [How does the Ping-Pong principle of the two receive buffers work?](#)) A BasicCAN structure offers the advantage of having no fixed link between a receive buffer and a message identifier. Instead, a programmable 11-bit identifier filter defines which messages are allowed to be received.

Basic CAN



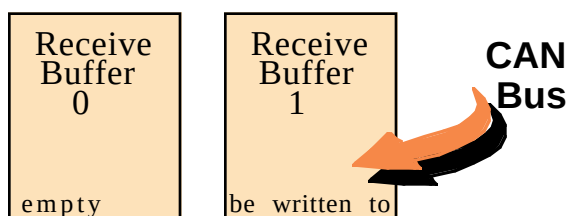
Full CAN



How does the Ping-Pong principle of the two receive buffers work?

As described, the MCAN module offers two receive buffers which can be accessed by the CPU or the CAN bus. Both buffers alternate with each other to receive data from the bus. This gives the CPU time to read a message buffer while a new message is being received at the same time. The illustrations demonstrate the steps involved during a message reception.

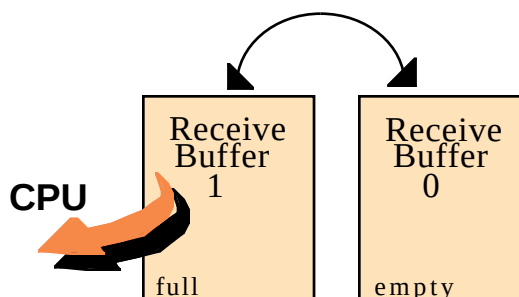
- Both receive buffers were empty and released to allow reuse by the CAN interface. A new message arrives and will be written into receive buffer 1.



During the time that the message is received, the relevant status bits have these states:

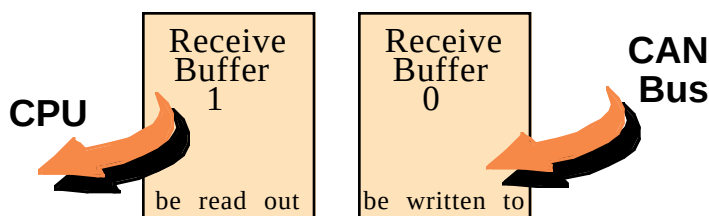
Receive status (RS):	1
Receive buffer status (RBS):	0
Receive interrupt flag (RIF):	0
Release receive buffer (RRB):	0

- As soon as receive message buffer 1 is full, this will be indicated by the flags RBS and RIF, and control of this receive buffer is given to the CPU. If the receive interrupt has been enabled, the interrupt routine starts to read and release this message buffer.



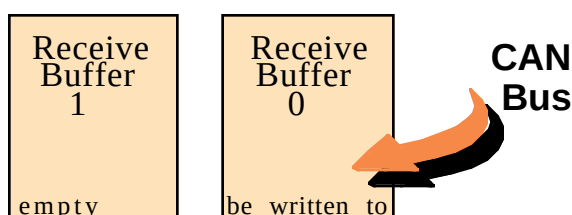
Receive status (RS):	0
Receive buffer status (RBS):	1
Receive interrupt flag (RIF):	1
Release receive buffer (RRB):	0

- At the same time that the last message is being read, a new message may be received from the CAN bus and written into the second message buffer (receive buffer 0). Receive status (RS) becomes active again. The receive interrupt flag has been cleared by reading the status register at the beginning of the interrupt routine. (Also, refer to [What does an MCAN interrupt handler look like?](#))



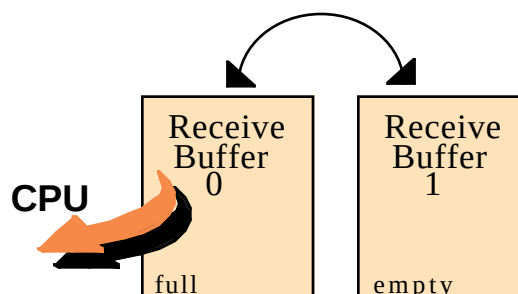
Receive status (RS):	1
Receive buffer status (RBS):	1
Receive interrupt flag (RIF):	0
Release receive buffer (RRB):	0

- After reading all the message bytes in receive buffer 1, the CPU releases this buffer by setting the RRB bit in the MCAN command register. The buffer can be reused by the CAN interface now. At the same time, the receive buffer status bit is cleared, showing that receive buffer 1, which is still attached to the CPU, is empty again.



Receive status (RS):	1
Receive Buffer status (RBS):	0
Receive Interrupt Flag (RIF):	0
Release Receive Buffer (RRB):	1

- The new message is fully stored into buffer 0 and the control of that buffer is given to the CPU. As receiving stopped, the receive status bit is cleared. As in step 2, the two bits RBS and RIF indicate that the buffer 0, which is currently attached to the CPU, is full. If receive interrupt has been enabled, the interrupt routine starts.



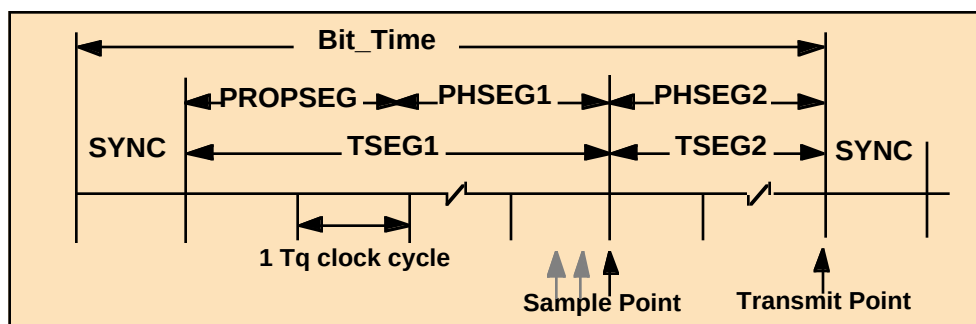
Receive status (RS):	0
Receive buffer status (RBS):	1
Receive interrupt flag (RIF):	1
Release receive buffer (RRB):	1

Which bit time should be chosen for a network?

To define bit time and the length of its time segments, some rules have to be followed. The CAN specification defines a bit time consisting of four segments. These segments are:

- Synchronization segment (SYNC)
- Propagation segment (PROP)
- Phase segment 1 (PS1)
- Phase segment 2 (PS2)

Each segment length must be set up by the CAN timing registers CBT0 and CBT1. While SYNC represents the synchronization segment, TSEG1 is a summation of the propagation segment and phase segment 1, and TSEG2 represents phase segment 2. The segment length is set up by defining the number of time quanta ($T_q = t_{SCL}$) per segment. Time quanta is the smallest time unit of the MCAN module and is derived from a programmable prescaler.



The setup of the time segments has to fulfill these rules:

Length of Time Segments

$$\begin{aligned}
 \text{SYNC} &= 1 T_q \\
 \text{PROP} &= 1, \dots, 8 T_q \\
 \text{PS1} &= 1, \dots, 8 T_q \\
 \text{PS2} &= \max(\text{PS1}, 2) \\
 \text{SJW} &= 1, \dots, \min(4, \text{PS1}) \\
 \text{with } f_N &= \text{Synchronization Jump Width}
 \end{aligned}$$

In some special cases during a CAN communication, there could be a long sequence of bits (28 to 30 bits) where no signal transition from recessive to dominant occurs. There is no possibility for resynchronization during this time, and the phase shift should not exceed the synchronization jump width. This worst case condition used to be considered within the oscillator tolerance calculation of the original CAN protocol. But this allows only clock tolerances below 0.5 percent. To allow the use of ceramic resonators, the CAN specification has been modified so that larger phase shifts of up to one bit time are allowed for long sequences of equal bits.

The two formulas of the enhanced CAN protocol imply all worst case conditions which have to be considered:

Clock Tolerance Rules

Requirement 1: Sample correctly the first bit after sending an active error flag (localizing bus errors)

$$(2 * df) * (13 * BT - PS2) < \min(PS1, PS2)$$

Requirement 2: Correct synchronization in stuffed part of the bit stream

$$(2 * df) * 10 * BT < SJW$$

with f_N = nominal CAN clock frequency

f = actual CAN clock frequency

df = relative clock difference: $(df = |f - f_N| / f_N)$

The two formulas show that the system's required clock accuracy depends on the bit time definition. The smaller the bit time and the larger the synchronization jump width, the larger the allowed clock tolerance is.

To understand the above rules better, this example shows how to set up the CAN bit time in a system.

Example

The following system parameters are assumed:

CAN bus frequency	500 kBaud → 2 μs bit time
Bus driver delay	50 ns
Receiver circuit delay	30 ns
Bus line delay (40 m)	220 ns
Total propagation delay	600 ns

First, the correct oscillator frequency has to be chosen. The oscillator frequency influences the length of the bit segments PS1 and PS2 and with that the synchronization jump width. As the formula above shows, the higher the synchronization jump width and the phase segments the larger is the maximum allowed clock tolerance. The smallest possible crystal/resonator frequency, which fulfills the propagation delay requirement is 6 MHz. Larger frequencies as 8 MHz or 16 MHz show better values for the maximum allowed clock tolerance due to larger SJW, PS1 or PS2 values. In resonator based systems the maximum allowed clock tolerance should be as large as possible. A 16 MHz clock showed the highest tolerance values and were chosen for this example.

When selecting a crystal frequency for the microcontroller, the internal bus frequency has to be considered as well. With higher external crystal frequencies a higher clock divide ratio may be needed, which could result in different internal bus frequencies. An oscillator frequency of 16 MHz with a clock divide ratio of 8 gives an internal bus frequency of 2 MHz.

Oscillator frequency	16 MHz
----------------------	--------

With a 16-MHz oscillator frequency, the MCAN module is clocked with a frequency of 8 MHz. A prescaler value of 1 defines the number of time quanta per bit time.

$$Tq = t_{SCL} = \frac{2 * P}{f_{OSC}} = 125 \text{ ns}$$

Time quanta per bit time	16 Tq
Prescaler	1

The propagation segment has to be able to buffer the system’s signal delay time. The segment should have double the length of the delay between sender and receiver, which is 600 ns in this example. With a time of 125 q is specified as the propagation delay time to fulfill this requirement.

Propagation delay segment 5 Tq = 625 ns

The synchronization segment of 1 Tq and the propagation segment of 5 Tq within the total bit time of 16 Tq still leaves 10 Tq to be shared between phase segment 1 and phase segment 2. As the above rules show, phase segment 2 equals phase segment 1 if phase segment 1 is greater than 2 Tq. The length of 5 Tq for each segment fulfills this requirement.

Phase segment 1 5 Tq
Phase segment 2 5 Tq

Corresponding to the above rule, a phase segment length of 5 Tq allows the synchronization jump width of a maximum value of 4 Tq. At the same time, this gives the maximum possible clock tolerance for that bus frequency.

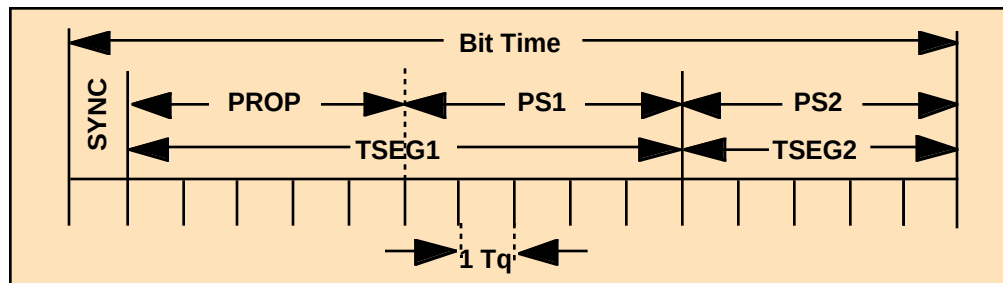
In this example, the MCAN module of the M68HC05 Family has to be set up with these values:

TSEG1 (PROP + PS1) 5 Tq + 5 Tq = 10 Tq
TSEG2 (PS2) 5 Tq
SJW 4 Tq

The corresponding bit timing registers of the MCAN module CBT0 and CBT1 would be specified as:

CBT0	1 1 0 0 0 0 0 0								=\$C0
	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0	
CBT1	0 1 0 0 1 0 0 1								=\$49
	SAMP	TSEG22	TSEG21	TSEG20	TSEG13	TSEG12	TSEG11	TSEG10	

After initializing the MCAN timing registers for the above values, the bit time of this example can be seen in the next picture.



Which resynchronization mode should be chosen?

The following phenomenon has to be considered when resynchronization on both edges is selected. In that mode, a transmitter will resynchronize on positive phase shifts if a dominant bit is sent. If the transceiver delay of that node is larger than the SYNC time quanta, the transmitting node receives its sent bit edge after the SYNC time quanta and then resynchronizes itself. This will result in a variable bit time/ baud rate on the bus. In a CAN network with a different bit timing definition in some CAN nodes (for instance, due to different used crystals), this could cause bus failures.

As described earlier, an enhanced CAN protocol was added, which allows phase shifts of a whole bit time after long sequences of equal bits. With these modifications to the original CAN protocol, the resynchronization on recessive-to-dominant edges as well as dominant-to-recessive edges has absolutely no advantage over resynchronization on recessive-to-dominant edges only. The oscillator tolerance is the same for both ways of resynchronization.

Therefore, resynchronization is recommended only on recessive-to-dominant edges.

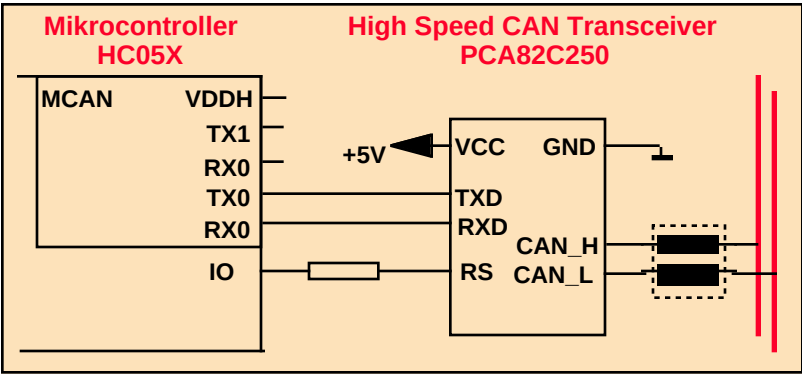
How is the HC05X microcontroller connected to an external CAN driver?

Most applications today implement an integrated physical bus interface device which is specified for a higher drive current and which includes short-circuit protection and slope control. Some newer interface circuits even integrate an automatic single-wire mode switch, which recognizes an interrupted or short-circuited bus line to GND/ V_{DD} and automatically switches to one functional bus line.

The appropriate physical interface for a system depends on the speed, which is needed for the CAN bus. There are low-speed physical interfaces (<125 kBaud) and high-speed physical interfaces (up to 1 Mbaud).

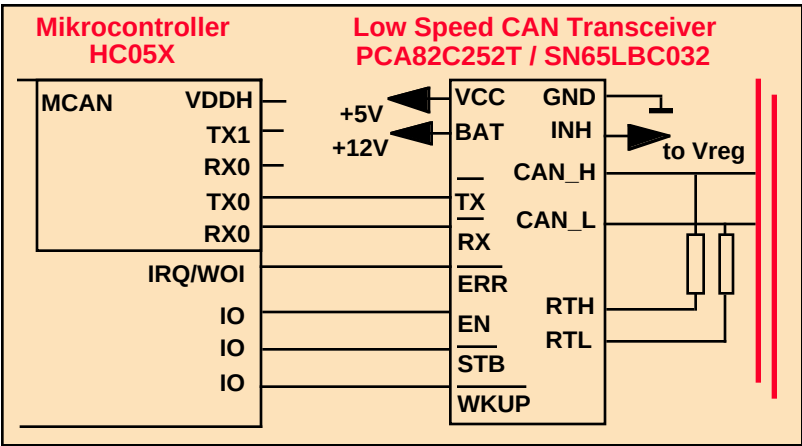
When using external transceiver devices, the internal comparator logic of the M68HC05 Family is not needed. Only one transmit and one receive line are connected between the microcontroller and the transceiver. The voltage reference pin, V_{DDH} , of the X-microcontroller can be left open, if internally the V_{DDH} reference is connected to one comparator input. This can be done by initializing the CCOM and COCNTRL register for single-line mode. (For more details, see [When should and how should the single-line mode be used?](#))

Two examples of the HC05X controller combined with an external transceiver are shown here. When using a Philips PCA82C250, only three pins need to be connected. Use one TX and one RX CAN line (either 0 or 1 can be chosen) in addition to one input/output pin, which controls the output slope of the bus signal and the standby mode of the transceiver. If the port pin is set to low, the output slope will be controlled via the current through the resistor. This is used in applications with lower CAN bus speed. By switching the port pin to high, the transceiver enters the low-power mode because no current passes through the pin.



The second example shows the use of the newer, more sophisticated, low-speed transceiver PCA82C252 or SN65LBC032. This device is connected to VBAT to detect a short circuit to the battery and ground. In case of bus failures, the device automatically switches to single-line mode, which reduces the HC05X microcontroller’s error handling software.

In addition to the CAN lines TX and RX, three input/output pins and one interrupt/wired or interrupt pin have to be spent to control the transceiver. The connections to EN and NSTB control the mode of operation of the transceiver (sleep, standby), and the input/output line to NWKUP can enable a wakeup request to the powered down transceiver. Feedback of any failure on the bus or of a wakeup is sent to the microcontroller via the NERR line, which triggers an interrupt.

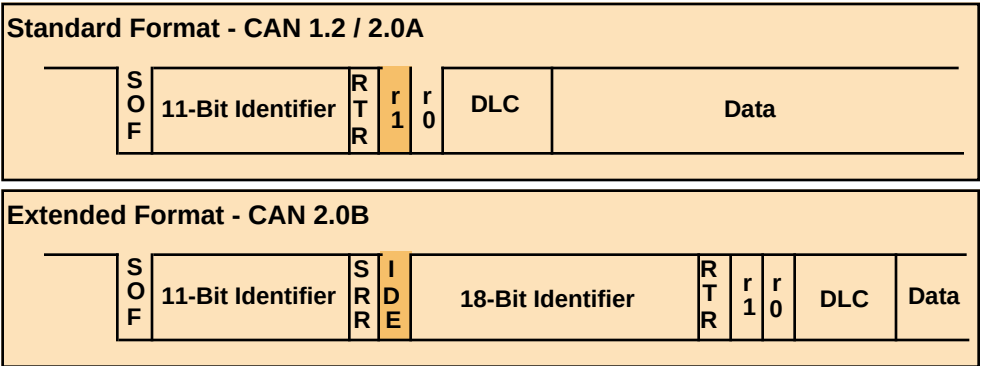


Can the M68HC05 Family be used in a network which runs extended CAN frame format?

The MCAN module has been designed according to CAN protocol 1.2 (standard CAN, 11-bit identifier) as defined by Robert BOSCH GmbH. The two reserved bits, r1 and r0, within the control field are sent as dominant bits and are not evaluated during reception.

Bit r1 of those reserved bits has been redefined within CAN standard 2.0 B (extended CAN, 29-bit identifier). It has been renamed to IDE bit (identifier extension) and distinguishes between extended CAN and standard CAN format.

Because the IDE bit is not evaluated in the MCAN module, an extended, 29-bit identifier frame would be treated as a standard frame, which would lead into a form error.



How are the MCAN module registers initialized?

The MCAN registers usually are initialized within a separate CAN initialization routine. For some registers, it is mandatory to be in the CAN reset state during initialization, which is entered by setting the reset request bit in the CAN control register.

These registers are:

- Acceptance code register (CACC)
- Acceptance mask register (CACM)
- Bit timing register 0/1 (CBT0, CBT1)
- Output control register (COCTRL)

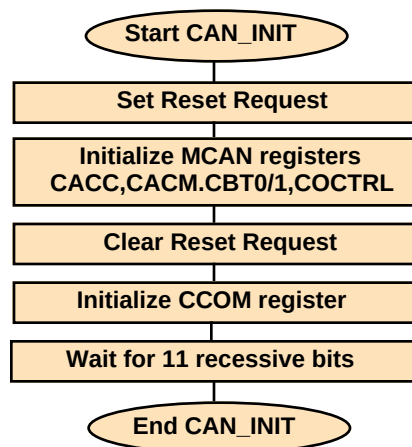
NOTE: In opposition to these registers, the CAN command register (CCOM) must not be initialized during the reset condition of the MCAN module. (The reset request bit in the CAN control register is set.) All bits of the CCOM register are forced to 0 during the CAN reset state. Initializing the register while CAN reset is entered would have no effect.

NOTE: The CAN command register (CCOM) must not be modified by using the read-modify-write instructions BSET/BCLR. This register is write only and returns the value \$FF when reading it. Thus, a BSET instruction would first read \$FF, then set a bit and write \$FF back to the CCOM register, which would result in a non-functional CAN module.

When setting the MCAN out of reset mode (clearing RR bit), remember that the MCAN module doesn't start normal operation immediately. In a normal situation, MCAN waits for 11 consecutive recessive bits on the CAN bus before starting normal operation. If the bus off state is active, MCAN waits for 128 occurrences of 11 recessive bits before starting normal operation.

Therefore, a wait loop after the reset sequence should be added to make sure that no MCAN action, such as a sleep mode request, is activated before MCAN re-enters normal operation mode.

This diagram shows the usual sequence of a CAN initialization routine.



The init routine usually should be executed only after power-on reset. Therefore, the bus off state does not need to be checked, and there is no need to wait for 128 times 11 consecutive recessive bits within the init routine. Usually, bus off failure is handled separately. (For more details, see [How should a bus error or bus off condition be handled?](#))

Here is an example of an MCAN initialization routine written in assembler language:

Engineering Bulletin

```

*
LDA    #RIE|OIE|EIE|RR    ;set up CAN control register:
STA    CCNTRL              ;enable Receive/Overrun/Error interrupts,
*                          ;only rec. to dom. edge for synchronization,
*                          ;reset request present,

LDA    #$FF
STA    CACM                ;accept all IDs-->no need to initialize CACC
*

LDA    #$58
STA    CBT1                ;Set up bus timing reg. 1
*                          125 kBaud -> 8 µs bit time
*                          with 4 MHz crystal
*                          for P=1: 1tSCL = 1Tq = 0.5 µs
*                          -->BT = 16 x tSCL
*                          choosen:tSEG1 = 9 x tSCL
*                          tSEG2 = 6 x tSCL
*                          tSYNC = 1 x tSCL

LDA    #$C0
STA    CBT0                ;Set up bus timing reg. 0
*                          P=1 --> tSCL = 0.5 µs
*                          tSJW = 4 x tSCL
*

LDA    #$FA
STA    COCNTRL             ;Set output control register
*                          Normal mode 1,
*                          OCTP0 = OCTN0 = 1
*                          OCTP1 = OCTN1 = 1
*                          --> drivers Tx0 and Tx1 push/pull
*                          -->both NTrans and PTrans enabled
*                          OCPOL0 = 0, OCPOL1 = 1,
*                          Tx0 Tx'ed normally, Tx1 inverted

LDA    CCNTRL
AND    #$FE                ;Set up CAN control reg.
STA    CCNTRL              ;set reset request absent
*

LDA    CCOMCPY             ;load a copy of CCOM register in RAM
ORA    #CMPSEL             ;setup sleep comparator for
STA    CCOM                ;two line mode
STA    CCOMCPY             ;save CCOM register (write only)
*

NOBSLDACSTAT              ;check receive/Transmit state for bus idle
AND    #$30                ;bus is idle, if TS and RS equals 0
BNE    NOBS                ;wait until bus is idle
BRS    WAIT88              ;if bus is idle, then count 11 consecutive bits
*                          ;which is 88 µs at 125 kbaud bus speed

RTS

```


When should and how should the single-line mode be used?

The usual CAN configuration uses two signal lines which transmit a differential signal. A differential signal allows redundancy so that the system is less sensitive to bus failures in a disturbed environment. The transmitter and receiver of the MCAN module also can be set in a single-line mode, where only one line is used to transmit the signal. This mode could be used due to these reasons:

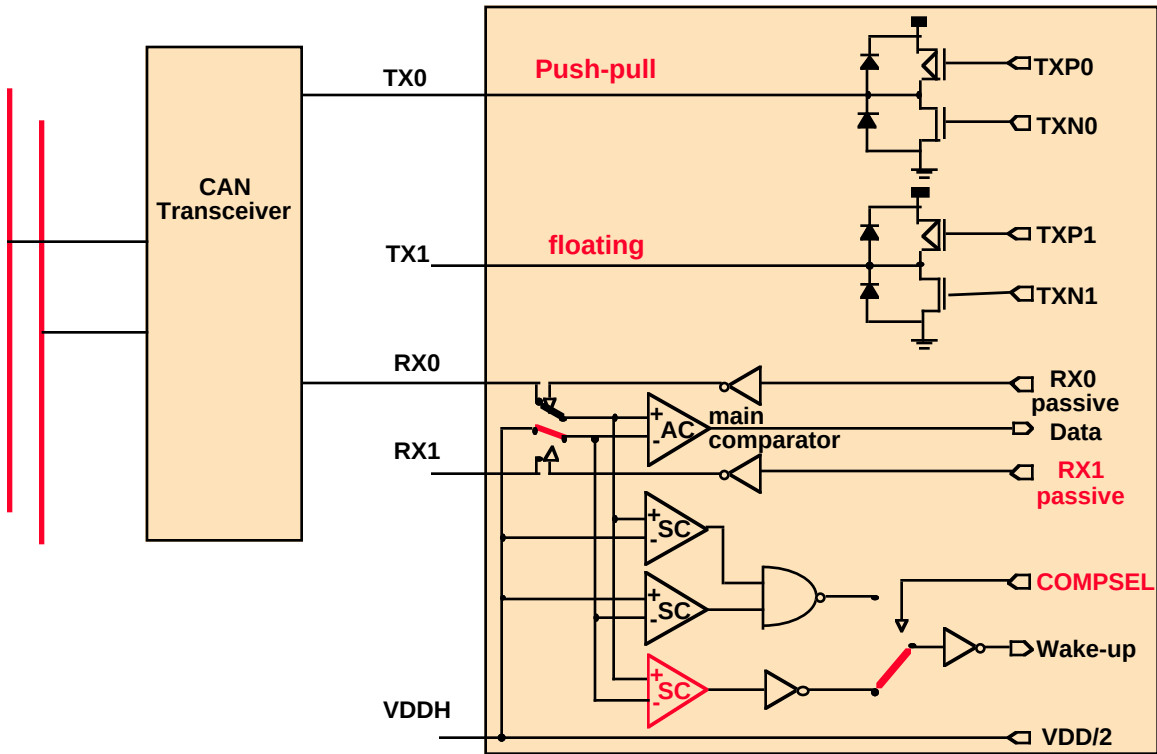
- The MCAN uses an external bus transceiver, which generates the differential signal itself.
- The environment is less critical, so that the system can renounce the redundancy of the two-line mode.
- Due to short circuit to ground or battery of one bus line or due to an interrupted bus line, the communication fails (error status active). One communication line is still operational.

One-line communication mode on the receiver side is entered by connecting one input of the internal receive comparator to a fixed reference voltage. The internal 2.5-V (V_{DDH}) reference should be used to save additional external connections. The switch to V_{DDH} is done in the CCOM register. Three bits have to be taken care of to use one-line mode: RX0, RX1, and COMPSEL.

On the transmitter side, only one transmit line should be enabled in single-line mode. The unused transmit line should be set into a floating state. This is set up by the appropriate value in the output control register.

The following pictures show the physical schematics of the MCAN physical interface in one-line mode (RX0/TX0 active) and the required register settings which have to be chosen in one-line mode:

Engineering Bulletin



One-line mode on TX0/RX0 line:

CCOM	0 1 0 0 0 0 0 0								=\$40
	RX0	RX1	Comp SEL	Sleep	COS	RRB	AT	TR	
COCNTRL	0 0 0 1 1 0 1 0								=\$1A
	OC TP1	OC TN1	OC POL1	OC TP0	OC TN0	OC POL0	OCM1	OCM0	

One-line mode on TX1/RX1 line:

CCOM	1 0 0 0 0 0 0 0								=\$80
	RX0	RX1	Comp SEL	Sleep	COS	RRB	AT	TR	
COCNTRL	1 1 0 0 0 0 1 0								=\$C2
	OC TP1	OC TN1	OC POL1	OC TP0	OC TN0	OC POL0	OCM1	OCM0	

The TX line polarity depends on the external physical interface. The register setup shown here uses positive polarity, as an example of the use of an external bus transceiver.

NOTE: The COMPSEL bit has to be cleared in one-line mode, which means that both main comparator inputs are compared to each other to recognize a wakeup condition on the bus. Since one comparator input remains connected to V_{DDH} by setting RX0/1, this configuration leads to the comparison of one single bus line with V_{DDH} (bottom sleep comparator SC in the schematics). If COMPSEL would be set in single-line mode, the two middle comparators would be used for wakeup recognition. In this case, one comparator would compare V_{DDH} against V_{DDH} , which is an unpredictable condition. When entering sleep mode this setup could immediately wake up the MCAN module.

NOTE: The CAN command register (CCOM) must not be modified by using the read-modify-write instructions BSET/BCLR. This register is write only and returns the value \$FF when reading it. Thus, a BSET instruction would first read \$FF, then set a bit and write \$FF back to the CCOM register, which would result in a non-functional CAN module.

What are the considerations when putting the MCAN module into sleep mode?

After requesting the MCAN sleep mode, the CPU usually checks if the sleep state really was entered. The MC68HC05X16 and the MC68HC(7)05X32 microcontrollers offer a sleep flag, which acknowledges the sleep request. This MCAN asleep flag (CAF) is located at bit 6 of the EEPROM/ECLK control register. The MC68HC(7)05X4 microcontroller also offers such a sleep mirror bit, which is located at bit 3 of the port configuration register (SLEEP).

After a successful sleep request, which is indicated by a set sleep flag, a stop instruction may be entered to power down the CPU. If the sleep flag is not set, a new message may be arrived, which can activate a receive interrupt in case of a valid, accepted message. After a certain waiting time, the CPU could try again to enter sleep mode.

The oscillator clock will stop only if the STOP instruction is executed after the sleep request. Otherwise, only the clock path of the MCAN module is stopped.

There are several reasons why the MCAN module could not enter sleep mode after setting the sleep request bit:

- The sleep request has been activated during message reception or transmission, which immediately causes a wakeup interrupt. Before setting the sleep request bit, always verify that no transmission is pending and that the MCAN module is in idle mode.
- The sleep request has been entered immediately after the CAN initialization routine. Perhaps not enough time went by between clearing the MCAN reset request and the sleep request so that the MCAN module could not re-enter normal operation mode. To wait for the occurrence of 11 consecutive recessive bits on the bus, a small wait loop should be included after clearing the reset request bit. Then the MCAN module starts normal operation and sleep can be requested. (For more details, see [How are the MCAN module registers initialized?](#))

After the MCAN module is put into sleep mode, why doesn't the oscillator stop?

- The CCOM register has been used wrongly. A read-modify-write operation such as BSET may be used to set the sleep bit. This register is write only and returns the value \$FF when reading it. Thus, a BSET instruction would first read \$FF, then set a bit and write \$FF back to the CCOM register, which would result in a non-functional CAN module. LDA/STA instructions should be used instead to set a bit.
- One-line mode was selected, but the comparator selection bit, COMPSEL, of the CCOM register was set. In one-line mode, COMPSEL should be cleared. (For more details, see [When should and how should the single-line mode be used?](#))

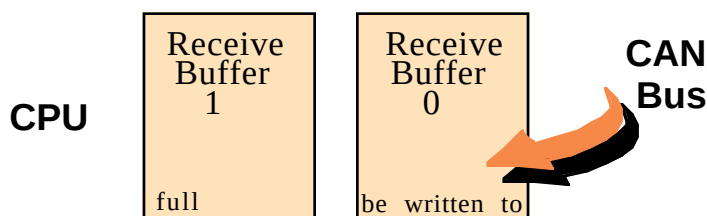
After the MCAN module is put into sleep mode, why doesn't the oscillator stop?

The oscillator stops only if the CPU executed a stop instruction and the CAN module was set into sleep mode. It remains in stop until either an external interrupt or a CAN interrupt occurs.

How should an overrun condition be handled?

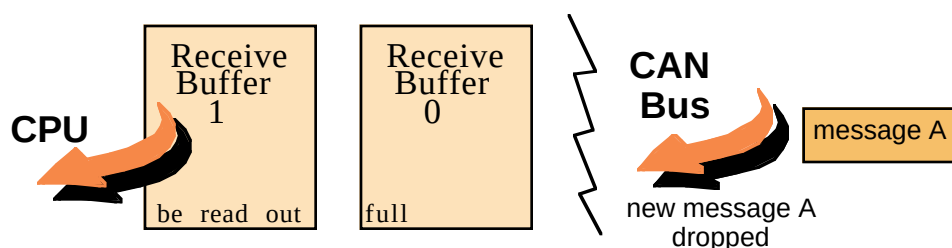
An overrun condition occurs if both receive buffers are full and a third message occurs. The following pictures illustrate such an overrun event. In this example, receive and overrun interrupts are enabled. Reading of the message buffers is controlled by the MCAN interrupt routine.

1. In this overrun example, the first received message is ready to be read by the CPU, while a second one is received from the bus and written into receive buffer 0. Due to other peripheral interrupts, such as a timer interrupt, the receive interrupt routine cannot start immediately to read out receive buffer 1.



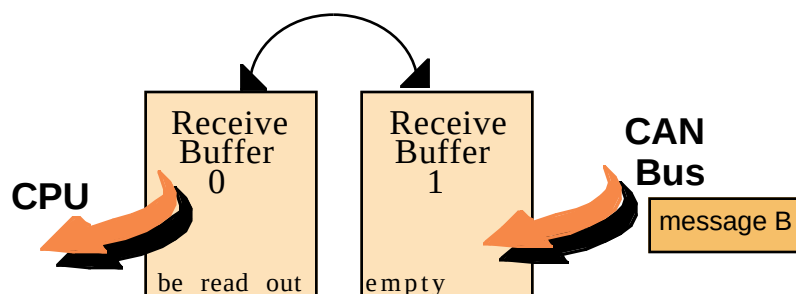
Receive status (RS):	1
Receive buffer status (RBS):	1
Receive interrupt flag (RIF):	1
Release receive buffer (RRB):	0

2. The receive interrupt routine starts and reads the message in receive buffer 1. Receive buffer 0 is now full, and a receive interrupt request for receive buffer 0 will be asserted. A third message arrives, which cannot be written into a buffer since the first message buffer is accessed by the CPU and the second one is full. The new message is dropped. This is indicated by the data overrun status bit (DO), which is set. If data overrun interrupt has been enabled in the MCAN control register, the MCAN interrupt line remains asserted, since it is still asserted by the receive interrupt.



Receive status (RS):	1
Receive buffer status (RBS):	1
Receive interrupt flag (RIF):	1
Release receive buffer (RRB):	0
Data overrun (DO):	1

- As soon as the CPU leaves the interrupt routine after reading receive buffer 1 and after releasing this buffer, the CPU will re-enter the MCAN interrupt routine immediately due to the latched receive and overrun interrupt requests. As receive buffer 1 is released, it can be accessed by a new message (B), which may arrive on the CAN bus. The RBS bit was cleared by releasing buffer 1, but it immediately was set again because the control of the buffer 0 is given to the CPU. The RRB bit gets cleared as soon as the controller starts to receive a new message (RS = 1).



Receive status (RS):	1
Receive buffer status (RBS):	1
Receive interrupt flag (RIF):	0
Release receive buffer (RRB):	0
Data overrun (DO):	1

Within the active MCAN interrupt routine, **all** MCAN interrupt flags have to be checked to start the corresponding interrupt subroutines. In this case, both the receive and the data overrun interrupt subroutines will be executed.

Within the data overrun subroutine the only action is to clear the data overrun status bit by setting the clear overrun status bit COS in the CCOM register.

How should a bus error or bus off condition be handled?

Error management can be defined for the lowest level of the OSI reference model, but also for the higher communication levels of this reference model. Only the low-level error management is discussed here.

The definition of the error management depends very much on the intelligence of the physical interface. When using a transceiver with integrated automatic single-line switch, the software for bus error handling gets reduced. In case of an active error, there's no need for a two-line/one-line mode switch by the software, as this is done in hardware by the transceiver itself. Within the error interrupt routine, the status line of the transceiver, which reports failure mode, could be evaluated and reported to the CAN management software.

When using a simple bus transceiver, no function in the physical interface device recognizes a short circuit or interruption of a bus line. Usually the error management software with this type of transceiver just tells the management software that a bus failure has occurred. Extra, costly hardware would be needed for bus failure sensing.

Another possible implementation is a discrete physical interface, where the two-line mode of the MCAN is used. With this configuration, an active error status could trigger a software algorithm within the MCAN interrupt routine to detect a faulty bus line.

Error status

The error status bit is set when the receive or transmit error counter reaches value 96. The MCAN interrupt handler should give an indication to the CAN management software that the bus may be heavily disturbed. To avoid a resultant bus off state (only produced by transmission), this may lead the management software to stop any further transmission, if possible.

When using the two-line mode with a discrete physical layer, the error status interrupt software could trigger an algorithm to detect possible bus line interruption or short circuits.

Bus off status

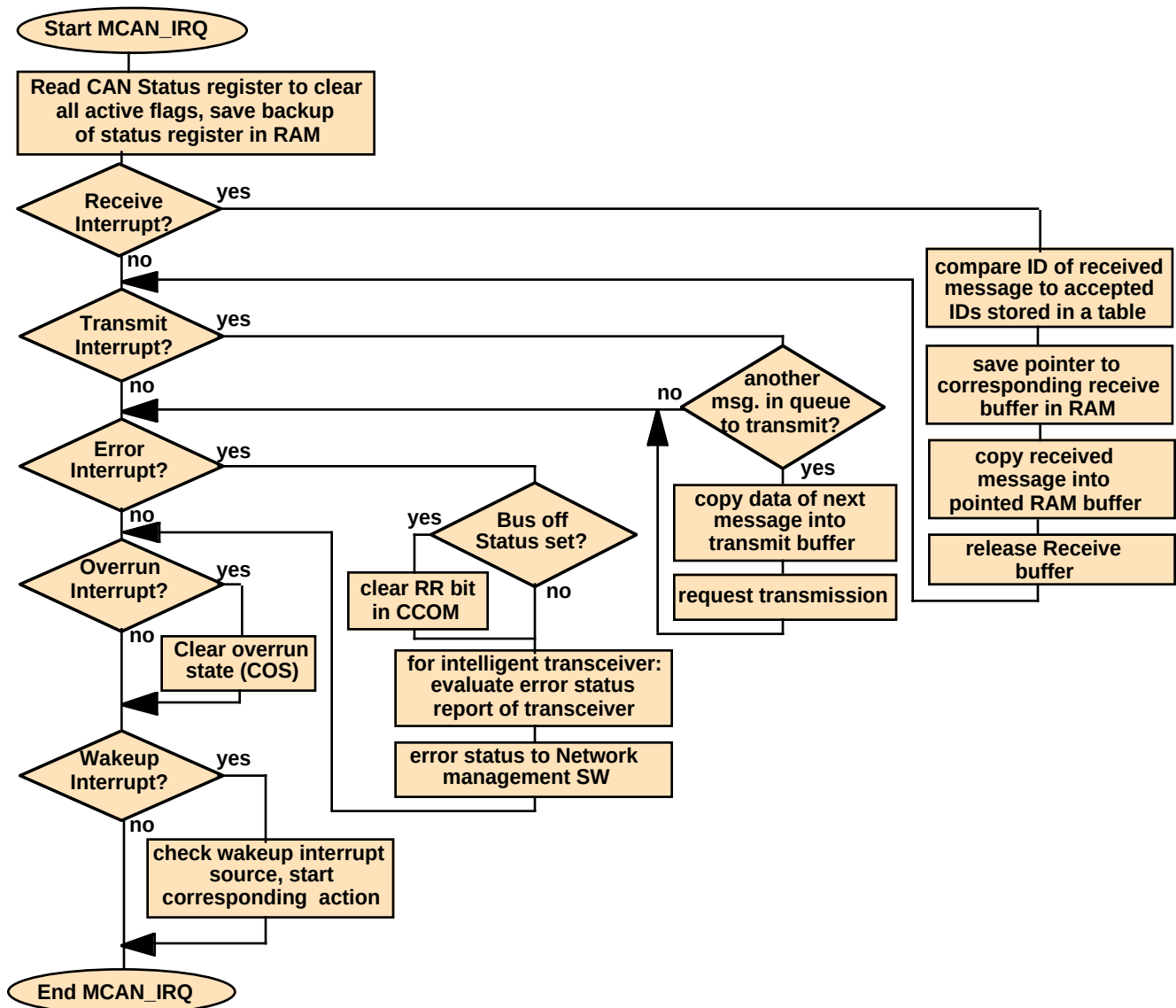
When the bus status flag in the status register CSTAT gets set, a bus off condition is signaled, which means the internal transmit error counter reached the value 255. This state can never be reached, if the node is in receive mode or if there is only one node on the bus.

Within the MCAN interrupt routine, the RR bit should be cleared and the management software should be made aware of a problem on the bus.

NOTE: The error status and bus off bits are level sensitive. But the error interrupt flag is not level sensitive. The interrupt flag gets set only when either bus off or error status bits change. If the error interrupt becomes active due to a set error status bit, the error interrupt flag gets set and will call the MCAN interrupt handler, when the error interrupt has been enabled. After clearing the error interrupt flag within the MCAN interrupt routine, it gets set again only if the bus off state gets active in addition to error status or if error status was cleared and then set again.

What does an MCAN interrupt handler look like?

This illustration is an example of the basic MCAN interrupt handler structure.



How is a message aborted?

In some cases, an urgent message gets activated, for example, by the network management software, but the transmit buffer is full. Then the loaded message has to be aborted to release the transmit buffer for the new message.

First the abort transmission bit (AT) in the CAN command register (CCOM) has to be set to request the abortion of a pending transmission. If the transmission is not already in progress, the buffer is released immediately, which sets the transmit buffer access bit. In the case of an enabled transmit interrupt, the transmit interrupt request gets asserted, which allows an orthogonal interrupt handler. Within the transmit interrupt routine the new message then will be copied automatically into the transmit buffer and requested for transmission.

If the transmission already is in progress when an abort request is being received, the abort request remains active and will be evaluated in the next interframe space. In case of lost arbitration on the bus, the abortion of the old message could then be executed as described above. In that case, the transmit buffer would not get released immediately after the abort request, which has to be considered in polling mode when no transmit interrupt is used. The transmit buffer access bit (TBA) in the CAN status register should be polled to indicate the released state before writing the new message into the buffer. Otherwise, the new data get lost without being signaled.

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Motorola data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and  are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

How to reach us:

USA/EUROPE/Locations Not Listed: Motorola Literature Distribution; P.O. Box 5405, Denver, Colorado 80217. 1-800-441-2447 or 303-675-2140

Mfax™: RMFAX0@email.sps.mot.com – TOUCHTONE 602-244-6609, US & Canada ONLY 1-800-774-1848

INTERNET: <http://motorola.com/sps>

JAPAN: Nippon Motorola Ltd. SPD, Strategic Planning Office 4-32-1, Nishi-Gotanda Shinagawa-ku, Tokyo 141, Japan. 81-3-5487-8488

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; 8B Tai Ping Industrial Park, 51 Ting Kok Road, Tai Po, N.T., Hong Kong. 852-26629298

Mfax is a trademark of Motorola, Inc.



MOTOROLA

© Motorola, Inc., 1997

EB181/D

**For More Information On This Product,
Go to: www.freescale.com**