*Application Note*

# *MCAN Debug Guide*

*Aparna Patra, Karan Saxena, and Tarun Mukesh Puvvada*

**ABSTRACT**

The Modular Controller Area Network (MCAN) peripheral is a CAN Flexible Data-Rate (CAN FD) implementation on select devices within the Sitara™ MCUs, MPUs, Jacinto™ DRA8x, TDA4x Processors, and so forth. This application note describes the basic debug steps to check CAN hardware and software configuration. The objective is to help users who face a MCAN communication failure

Various TI devices do not have CAN Transceivers onboard. This requires external transceivers to be connected to form a complete CAN bus. All the CAN peripherals need a second CAN FD node for operation to verify the external data transmission. This requirement can be met by another device featuring CAN FD or any CAN bus analysis tool that is capable of working with both classic CAN and CAN FD protocols. Many USB-bus based tools are currently available. In addition to providing visibility to the bus traffic, these tools are also capable of generating frames and are an invaluable aid in debugging. An oscilloscope with built-in CAN FD triggering and decoding is essential for debugging. Throughout this document, the terms MCAN and CAN FD are used interchangeably. While *CAN FD* refers to the protocol, *MCAN* refers to the peripheral in the MCU that implements the protocol.

## Table of Contents



## Trademarks

# 1 Introduction

CAN is a serial communication protocol that was originally developed for automotive applications. Due to the robustness and reliability, CAN is pertinent to applications in diverse areas such as industrial equipment, medical electronics, trains, aircraft, and so forth. CAN protocol features sophisticated error detection and confinement mechanisms and has simple wiring at the physical level. The original CAN protocol standard is now referred to as *classical* CAN to distinguish from the more recent CAN FD standard. CAN Flexible Data Rate (CAN FD) is an enhancement to the classical CAN in terms of higher bit rates and the number of bytes transferred in one frame, thus increasing the effective throughput of communication. While classical CAN supports bit rates up to 1Mbps and a payload size of 8 bytes per frame, CAN FD supports bit-rates up to 5Mbps and a payload size of up to 64 bytes per frame.

# 2 MCAN Features

The following are the supported MCAN features:

- Conforms with CAN Protocol (ISO 11898-1:2015)
- Full CAN FD support (up to 64 data bytes)
- Up to 32 dedicated transmit buffers
- Configurable transmit FIFO, up to 32 elements
- Configurable transmit queue, up to 32 elements
- Configurable transmit Event FIFO, up to 32 elements
- Up to 64 dedicated receive buffers
- Two configurable receive FIFOs, up to 64 elements each
- Up to 128 filter elements
- Loop-back mode for self-test
- ECC check for Message RAM
- Clock stop and wakeup support
- Timestamp counter

# 3 MCAN Software Configuration

## 3.1 Filter Configuration

The MCAN module is capable to configure two types of acceptance filters - one for standard(11-bit ID) and one for extended identifiers(29-bit ID).

Acceptance filtering starts when complete Message ID is received. Acceptance filtering stops at the first matching enabled filter element or when the end of the filter list is reached. If a filter element matches, then the Rx Handler starts writing the received message data in portions of 32 bit to the matching Rx Buffer or Rx FIFO. If an error condition occurs (for example, CRC error), then this message is rejected. To receive message for a particular CAN ID: if the receptor CAN node is not configured for the same ID, then data gets rejected.

Configuration registers to check for filter elements are:

- Global Filter Configuration (MCAN_GFC) register
- Standard ID Filter Configuration (MCAN_SIDFC) register
- Extended ID Filter Configuration (MCAN_XIDFC) register
- Extended ID AND Mask (MCAN_XIDAM) register

## 3.2 Transmitter Delay Compensation

The MCAN module provides a delay compensation mechanism to compensate the transmitter delay. The compensation mechanism enables transmission with higher bit rates during the CAN FD data phase. The transmitter delay compensation is enabled by setting the MCAN_DBTP[23] TDC bit to 1.

Users do not need to measure the transceiver loop-back delay or refer to the transceiver documentation for the same. Users need to program the SSP position, which is the same as the Sampling Point (SP). In case of CAN FD, make sure the Transceiver Delay Compensation value is correctly programmed. The Transmission Delay Compensation Value (TDCV) used as the Secondary Sample Point (SSP) is selected automatically by the CAN controller. The controller measures the delay between when a dominant signal is driven on m_can_tx to when the corresponding edge appears on m_can_rx, sampling every tq. This measured value is updated in the TDCV field of the Protocol Status Register (PSR address=h1044) at each transmission of CAN FD frame, is used as reference, and is used during the data portion of CAN FD transmission to verify good transmission.

The SSP that is selected by the CAN controller can be restricted to a certain window using the Transmitter Delay Compensation Offset (TDCO) and Filter Window Length (TDCF). The offset provides extra amount of time **quanta** to be added to the delay measurement of the CAN controller. The filter window length defines a minimum valid delay measurement, making sure the SSP is greater than the value defined here. These values can be configured in the Transmitter Delay Compensation Register (TDCR address=h1048).

This SSP position is in the data phase the number of mtq between the start of the transmitted data bit and the secondary sample point. Valid values are 0 to 127 mtq.

## 3.3 MCAN Bit Timing Parameters

The bit rate and sampling point for the CAN nodes needs to be calculated and configured accordingly for seamless communication. The CAN nodes need to be configured for same bit rates, and timing parameters. Any mismatch can lead to repeated introduction of error frames into the bus.

**Bit rate (bits per second) = (CAN clock in Hz) / BRP / (1 + TSEG1 + TSEG2)**

**Sampling Point(%) = (1 + TSEG1) / (1 + TSEG1 + TSEG2)**

where:

CAN clock is functional clock of CAN module (80MHz by default)

BRP: Bit rate pre-scalar value

TSEG1, TSEG2: Time segments used to define sampling point for the bit.

TSEG1: Time before the sampling point = Prop_Seg + Phase_Seg1

TSEG2: Time after the sampling point = Phase_Seg2

Sampling point is the point of time at which the bus level is read and interpreted as the value at that respective time. Typical value of sampling point must be between 75-90%. Note that, if Classic CAN is used nominal bit rate = data bit rate, then all other parameters for nominal and data sections need to be the same.

**Note**

Refer to the FAQ for more details: [FAQ] TDA4VM: CAN: How is bit-rate calculated for CAN.

# 4 Debug Tips to Resolve MCAN Communication Issues

CAN controllers have an on-chip error detection mechanism. If any node detects an error at the time of transmission or reception of a message, then the discovering node transmits an error flag or error frame. This helps in detection of faulty CAN frames. The other nodes also detect the error caused by the error flag if the nodes failed to detected the original error. This helps in taking appropriate action in rejection of that message.

## 4.1 Debugging the MCAN Hardware

1. **Bus Termination**: The bus termination at the ends must be within 120 ohms only. Proper bus termination is crucial for maintaining signal integrity on the CAN bus. Make sure that the bus is correctly terminated at both ends (120-ohm resistors). Incorrect termination can lead to signal reflections and communication errors.
2. **Power supply**: The CAN transceiver must be powered up properly.

   For example, in case of TCAN 1042D, apart from other physical connections, 5V power supply must be provided for CAN transmission
3. **Checking physical connections**: If an oscilloscope is used, then make sure the triggered signal from oscilloscope, matches with the signal probed on the board.
4. **Acknowledgment error**: When a node transmits a frame, this node expects an acknowledgment (ACK) back. When there is no ACK received from the receptor node, the transmitter node keeps sending the frame forever, until a bus-off is reached.
5. **Transceiver**: To see the waveform until the ACK phase, a transceiver must be connected to the node. Without a transceiver, the node immediately goes into an error state.

## 4.2 Debugging using MCAN registers

### *4.2.1 MCAN Protocol Status Register*

The MCAN protocol status can be inferred from the PSR register.

When any CAN message transmission or reception fails, the controller updates LEC (Last Error Code) and DLEC (Data Phase Last Error Code) fields in status register (MCAN_PSR)

**LEC**: This field is updated when error occurs in arbitration phase in case of CAN FD message or during full message for classic CAN message.

**DLEC**: This field is updated when error occurs in data phase in case of CAN FD message.

LEC and DLEC fields have the following errors:

- **Stuff Error**
  - Node type: Rx
  - Description: More than five equal bits in a sequence have occurred in a part of a received message where this is not allowed.
  - Check if all the nodes on the bus have same bit-rate.
- **Form Error**
  - Node type: Rx
  - Description: A fixed format part of a received frame has the wrong format.
  - This can happen if the receiver node has drift in CAN functional clock momentarily or bus has interference.
- **Ack Error**
  - Node Type: Tx
  - Description: The message transmitted by the MCAN module was not acknowledged by another node.
  - CAN bus needs to have more than one node (apart from the transmitting node) up and running (not in sleep or power down state).
- **Bit1 Error and Bit0 Error**
  - Node Type: Tx
  - Description:

    - **Bit1 Error**: During the transmission of a message (with the exception of the arbitration field), the device wanted to send a recessive level (bit of logical value 1), but the monitored bus value was dominant.
    - **Bit0 Error**: During the transmission of a message (acknowledge bit, active error flag, or overload flag), the device wanted to send a dominant level (data or identifier bit logical value 0), but the monitored bus value was recessive.
  - Tx and Rx lines are sampled at Sampling Point (SP) and Secondary Sampling Point (SSP) respectively have different values. Position of SSP is configurable.
  - This can happen in following cases:
    - Check if tx and rx pins are configured properly. These pins need to be pulled high by default, that is, when the bus is idle (when no communication is going on).
    - Check if CAN transceiver is enabled.
    - For LEC:
      - If this configured, then SP is too large for given node.
      - Segment after SP (TSEG2) needs to be large enough to compensate for Transceiver Delay.
    - For DLEC:
      - Configured Transceiver Delay Compensation value is less or more than required.
- CRC Error
  - Node type: Rx
  - Description: The CRC check sum of a received message was incorrect. The CRC of an incoming message does not match with the CRC calculated from the received data.
  - This can happen if receiver node has drift in CAN functional clock momentarily or bus has interference.

### *4.2.2 MCAN Error Counter Register*

When the node sends an error frame or flag, this keeps the track of this status in a buffer register.

CAN controller is has two registers namely TEC and REC.

1. **Transmit Error Counter Register (TEC):** Which counts the number of transmission errors detected on the frames that the ECU sends. Transmit Error Counter, values between 0 and 255.
2. **Receive Error Counter Register (REC):** Which counts the number of reception errors detected on the frames that the ECU receives. Receive Error Counter, values between 0 and 127.

Whenever any node detects an error and sends an error frame, the node increases the counter value. If there is a successful message after an error flag, then the node decreases the counter value. The increase or decrease of TEC or REC counter value depends if the error happened in Transmitter ECU or receiver ECU. There are several rules governing how these counters are incremented or decremented.

A CAN node can be possibly in one of the three error states:

1. Error Active State: In this state both of the error counters are less than 128 (REC < 128 and TEC < 128). This takes part fully in bus communication and signals an error by the transmission of an active error frame. This consists of a sequence of 6 dominant bits followed by 8 recessive bits, all other nodes respond with the appropriate error flag, in response to the violation of the bit stuffing rule.
2. Error Passive State: A node goes into an error passive state if at least one of the error counters is greater than 127 ((TEC > 127 | REC > 127) and TEC <= 255). This still takes part in bus activities, but sends a passive error to the frame, and only on errors. Furthermore, an error passive node has to wait an additional time (Suspend Transmission Field, 8 recessive bits after Intermission Field) after transmission of a message, before the node can initiate a new data transfer.
3. Bus-Off State: If the Transmit Error Counter of the CAN controller exceeds 255 (TEC > 255), then the node goes into the bus off state. The node is disconnected from the bus (using internal logic) and does not take part in the bus activities anymore. To reconnect the protocol controller, a bus-off recovery sequence has to be executed. This usually involves the re-initialization and configuration of the CAN controller by the host system, after which the node waits for 128 * 11 recessive bit times before commencing further communication.

## 4.3 Understanding MCAN applications in TI SDKs

### *4.3.1 MCU PLUS SDK*

1. Message is transmitted and received back internally using internal loopback mode. When the received message id and the data matches with the transmitted one, then the example is completed. The MCAN application code is generally configured for an internal loopback test. Make sure the App_mcanConfig() has the correct macro passed, depending on external and internal loopback test being performed.
2. Make sure to allocate the correct instance (Eg. MCU_MCAN0, MCAN0) in the syscfg settings(if supported in the SDK), and make hardware connections to those instances.
3. The calculated MCAN timing parameters can be set in syscfg, and allocate the same timing parameters in the receiver end as well, for proper transfer of data.

### *4.3.2 Linux SDK*

1. Some EVMs do not have onboard CAN transceivers; there are no transceiver nodes in the respective DTS files. A device tree overlay needs to be applied on top of default dtb (that is, k3-am62x-sk-mcan.dtbo).
2. To check if the device is registered or not, use the command dmesg | grep can.
3. Refer to the respective SDK documentation to test MCAN from Linux prompt. If the external communication does not work, then test the internal loopback mode first, before proceeding further. To use loopback mode, use the command ip link set can0 type can bitrate 1000000 dbitrate 5000000 fd on loopback on.

### *4.3.3 MCAL SDK*

1. CanApp_Startup ()
    a. Builds interrupt list and registers ISR for enabled CAN instances
2. CanApp_PowerAndClkSrc ()
    a. Dummy function for now.
3. CanApp_PlatformInit ()
    a. Pinmux configuration required for internal loopback testing mode
        i. Write 0x08050007 to specific pad config control register for the MCAN transceiver pin. Further details can be found under Device Configuration > Control Module in TRM.
        ii. Confirm the CAN transceiver is enabled by performing a read and compare on the same pin.
    b. Enables CAN Transceiver connected to CAN instances. Please note that for internal loopback testing this is not applicable.
4. CanApp_LoopbackTest ()
    a. For each enabled instance, follow the steps below:
        i. Initialize CAN hardware
        ii. Setup PDU info to transmit
        iii. Set CAN controller mode to START
        iv. Enable loopback mode
        v. Trigger CAN transmission
        vi. Wait for transmission completion
        vii. Check for Tx and Rx confirmation success
        viii. Set CAN controller mode to STOP
        ix. Disable loopback mode
        x. Compare received message Id and Data with PDU Info
5. Checks for error status, stack corruption and prints result

#### 4.3.4 PDK

The CSL MCAN application operates as either transmitter or receiver CAN nodes. The CSL MCAN application also supports digital loopback mode. Two boards can be used to emulate two nodes with CAN network; with one board as transmitter and the other board as receiver. This also provides an external loopback test, where one MCAN instance is configured as transmitter and the other as receiver on the same board. In all modes, MCAN operates as CAN-FD with arbitration bitrate and data-phase bitrate set to 1Mbps and 5Mbps, respectively. MCAN is comprised of four tests.

1. Receiver Test: Receives 15 messages with varying payloads between 1 byte to 64 bytes.
2. Transmitter Test: Sends 15 messages with varying payloads between 1 byte to 64 bytes.
3. Internal Loopback Test: Sends and receives 15 messages with varying payloads between 1 byte to 64 bytes.
4. External Loopback Test: Sends and receives 15 messages with varying payloads between 1 byte to 64 bytes externally from 1 CAN instance to another.

### 4.4 Other Common Issues

- **Pin Mux Configuration**: Make sure the Pin Mux configuration is done as per the CAN instance usage and also check the pin mux registers are not getting over written else where to avoid failure.
- **Transceiver Configuration**: Make sure that users the appropriate transceiver hardware connected to the MCAN module. The transceiver is essential for interfacing with the physical CAN bus. Verify that the transceiver settings match the communication requirements (that is, baud rate, termination resistors).
- **Acceptance Filtering**: MCAN uses acceptance filters to determine which messages MCAN needs to process. If the acceptance filters are not configured correctly, then users can experience issues with message reception or transmission. Double-check the filter settings.
- **Frame Format**: MCAN supports both standard CAN and CAN FD frames. Make sure that users are using the correct frame format (11-bit standard identifier or 29-bit extended identifier) based on the application requirements.
- **Error Handling**: Monitor error counters (such as transmit error count and receive error count) to detect communication issues. Excessive errors can indicate issues with the bus or incorrect configuration.
- **Initialization Sequence**: Follow the proper initialization sequence when setting up the MCAN module. Incorrect initialization can lead to unexpected behavior.
- **Message Buffer Configuration**: Allocate MCAN message RAM correctly. Define the message buffers (TX FIFOs, RX FIFOs, and RX buffers) based on the application needs. Insufficient buffer space can cause dropped messages.
- **Loopback Mode**: Use loopback mode during testing to verify that the MCAN module is functioning correctly. In loopback mode, transmitted messages are received back internally, allowing users to validate the communication path.
- **Interrupt Handling**: Implement interrupt handlers for MCAN events (such as message reception, transmission, and error conditions). Proper interrupt handling makes sure of timely processing of messages. Check if the ISR is getting hit upon reception of a message. If this is getting hit when sending a message from the external CAN emulator, then check the interrupt status; this can be inferred from the MCAN_IR register.
- **Clock Configuration**: The clock configuration needs to match the settings as mentioned in SDK guide. CAN clock is functional clock of CAN module, which runs at 80MHz

## 5 Related FAQs

Here is a list of FAQs that are helpful for debugging MCAN issues:

- Texas Instruments, [FAQ] TDA4VM: How to enable CAN along with linux on MCU2_1 using Main Domain CAN0 instance, webpage
- Texas Instruments, [FAQ] SK-AM62X/SK-AM62A-LP: Testing MCAN External Loopback example using external transceiver TCAN1042D, webpage
- Texas Instruments, [FAQ] J7200XSOMXEVM: How do I enable CAN with Linux driver on J7200?, webpage
- Texas Instruments, [FAQ] TDA4VM: How do I enable CAN channels on Gateway/Ethernet Switch/Industrial (GESI) expansion card when using J721E EVM from Linux A72?, webpage
- Texas Instruments, [FAQ] TDA4AL-Q1: CAN Profiling Application failed to operate in Tx only or External Loopback and Polling mode or interrupt mode., webpage
- Texas Instruments, [FAQ] AM69: How can I assign interface ID to CAN modules, webpage
- Texas Instruments, [FAQ] DRA821: canTxstatus is busy even after Cancellation of Finished messages is set for respective Tx Buffer, webpage
- Texas Instruments, [FAQ] SK-TDA4VM: How do I use the CAN bus headers on SK-TDA4VM using Edge AI SDK?, webpage
- Texas Instruments, [FAQ] TDA4VM: MCUSW demo on CAN along side Linux on A72, webpage
- Texas Instruments, [FAQ] TDA4VM: Not able to receive messages for a particular CAN ID, how to debug?, webpage
- Texas Instruments, [FAQ] TDA4VM: CAN: How is bit-rate calculated for CAN, webpage
- Texas Instruments, [FAQ] TDA4VM: How can I use CAN on Linux, webpage

## 6 Summary

The CAN bus has become a key communication standard in the automotive industry, particularly for applications requiring numerous short messages with high reliability. The message-based (rather than address-based) architecture makes this an excellent choice for scenarios where data must be accessed by multiple locations, making sure of system-wide data consistency. One of the major advantages to CAN is fault confinement: malfunctioning nodes are automatically removed from the bus, preventing a single faulty node from disrupting the entire network and preserving bandwidth for critical messages. As vehicles become more connected and autonomous, the demand for efficient and secure communication protocols like CAN continues to rise, driving innovation and progress within the automotive sector.

## 7 References

- Texas Instruments, *Getting Started with the MCAN (CAN FD) Module*, application note
- Texas Instruments, *Introduction to the Controller Area Network (CAN)*, application note
- Texas Instruments, *Basics of debugging the controller area network (CAN) physical layer*, analog design journal
- Texas Instruments, MCAN Loopback Interrupt, webpage
- Texas Instruments, How to enable MCAN in Linu, webpage

# IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265