

HOW TO USE A CAN TRANSCEIVER FAULT DETECTION CIRCUIT WITH ALGORITHM

Abstract:

This application note discusses the MAX33011E's control area network (CAN) bus fault detection feature and demonstrates how to implement the fault detection algorithm in firmware with example codes.

Introduction

Troubleshooting a control area network (CAN) when data is not transmitting or receiving can be frustrating. Maxim has developed a built-in fault detection mechanism in the CAN transceiver that helps users to quickly identify the root cause. This application note presents the capabilities of the fault detection mechanism, explains how it works, and demonstrates how to implement a fault detection algorithm in firmware with example codes.

Enabling the Fault Detection Circuit

The MAX33011E CAN transceiver requires 100 TXD rising edges (typically a few CAN protocol messages) to enable the fault detection circuit. After the fault detection circuit is enabled, the transceiver can still transmit messages as normal.

Reading and Clearing the Fault Code

When a fault condition is detected, the transmitter will be disabled and the FAULT pin will pull high through the external pullup resistor. When the system controller receives the FAULT pin signal, 16 low-to-high transitions on TXD are required to shift out the fault code as shown in Table 1. An additional 10 low-to-high TXD transitions clear the fault and disable the fault detection circuit. For example, the overcurrent fault code is 101010 and its timing diagram is shown in Figure 1.

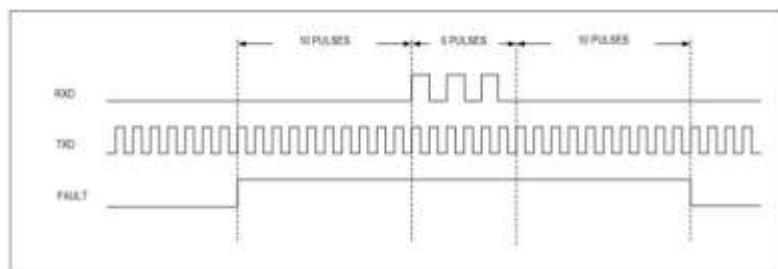


Figure 1. Overcurrent fault reporting timing diagram

Fault Conditions

The MAX33011E is the first CAN transceiver with built-in fault detection circuit. When the fault detection circuit is enabled, it can detect three types of common fault conditions (overvoltage, overcurrent, and transmission failure) on a CAN bus as listed in Table 1.

Fault	Condition (Fault Detection Enabled)	Fault Code	Possible Cause
Overcurrent	CANH output current and CANL input current are both > 85mA	101010	<ul style="list-style-type: none"> CANH shorted to CANL CANH connected to GND and CANL connected to V_{DD}
Overvoltage	CANH > +29V or CANL < -29V	101100	<ul style="list-style-type: none"> CMR fault
Transmission Failure	RXD unchanged for 10 consecutive TXD pulses, recommended minimum frequency = 200kHz	110010	<ul style="list-style-type: none"> Open load (both termination resistors missing) on CANH and CANL Exceeds driver's common-mode range CANH and /or CANL connected to a fixed voltage source

Overcurrent Fault

Overcurrent fault is detected when the source current of CANH and the sink current of CANL are both higher than 85mA (typically). The more probable cause of the fault is that CANH and CANL are shorted on the bus. However, if the short is far away from the CAN node, it may not be detected due to high cable impedance. Slowing down the CAN signal frequency could lower the cable impedance and helps detect the short from a further distance. But if total resistance of the cable becomes significantly high, a short will not be detected even when the CAN signal is constantly in dominant mode. **Figure 2** shows the maximum operating frequency for overcurrent detection versus cable length as a reference. A Cat5E copper clad aluminum cable is used. Maximum frequency will vary with the type of cable.

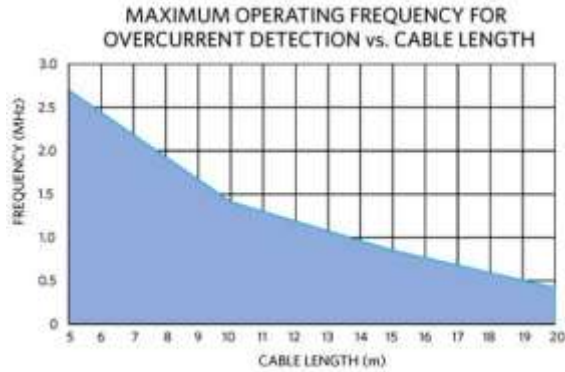


Figure 2. Maximum Operating Frequency for Overcurrent Detection vs. Cable Length

Overvoltage Fault

The MAX33011E common-mode input range (CMR) is $\pm 25V$. Overvoltage fault is detected when either CANH is higher than 29V or CANH is lower than -29V. This is caused by the CMR going beyond the specification.

Transmission Failure Fault

After the fault detection circuit is enabled, the transceiver can still transmit messages. In normal operating condition, RXD echoes the TXD signal. In case RXD does not echo the TXD signal for 10 consecutive pulses, the detection circuit generates the transmission failure fault. There are a few common possible causes that make RXD unable to echo TXD:

1. If CANH and CANL are shorted to a supply and the transceiver is not able to overdrive the supply, the receiver will always see a fixed signal on the CAN bus.
2. When the common-mode voltage exceeds the driver's common-mode range (-5V to +10V), the driver is turned off. When the driver is off, the CANH/CANL output will not reflect the signal on TXD, and the receiver will see a fixed signal on the CAN bus.
3. If termination resistor is not connected to the CAN node, it may cause transmission failure. The termination resistors play a very important role of bringing CANH and CANL to the same voltage level in recessive mode. Without the termination resistors, the transceiver's internal common-mode voltage buffer can still bring CANH and CANL together, but at a much slower rate. The capacitive load on the bus could also slow down CANH and CANL voltages from merging. When the controller sends pulses to TXD, and if the recessive interval is not long enough for the differential voltage (CANH – CANL) to go below the input low threshold for 10 consecutive pulse cycles (RXD stays low for the 10 TXD pulses), transmission failure fault will be reported. This also means if the TXD high time is too long, the CAN bus signal could enter recessive mode and RXD will become high, no transmission failure fault will be reported. The recommended minimum TXD pulse frequency to detect transmission failure fault is 200kHz.

Fault Detection Algorithm

Maxim has developed an algorithm that can reliably detect fault conditions on the CAN bus using the MAX33011E without interrupting normal CAN communication. All the following example Mbed® codes were developed for NUCLEO-F303K8 platform.

Typically, a microcontroller with CAN peripheral is used in every node on a CAN network. To perform fault detection, the TXD and RXD pins of the microcontroller must be configured as GPIOs to bit-bang the TXD signal and read the fault code from RXD. An interrupt pin is needed to connect to the fault signal of the MAX33011E.

To avoid interrupting normal communication, the algorithm needs strong indication of communication failure before entering fault detection mode. The algorithm will enter fault detection mode if one of the following happens:

- FAULT goes high.
- Transmitter generates a bit-error frame.
- Transmitter Error Counter rises above 255 and the node enters the Bus-Off state.

The following example code configures the microcontroller pins as GPIO when any of the above conditions becomes true.

We have used STM32F303K8 MCU as the CAN controller. This reference code was developed on Mbed-OS, which is a free open-source embedded operating system. For more details visit: <https://os.mbed.com/mbed-os/>

Mbed offers APIs to configure microcontroller's IOs as digital input/output pins. Any other similar low-level APIs may also be used.

```
DigitalOut txd (PA_12); // Configures PA_12 of MCU (TXD of CAN controller) as digital o/p pin
DigitalIn rxd (PA_11); // Configures PA_11 of MCU (RXD of CAN controller) as digital i/p pin
```

In fault detection mode, 2 μ s TXD positive pulses should be used. After the positive pulse, TXD can stay low as long as needed to process the algorithm. This allows the fault detection circuit to reliably detect both overcurrent and transmission failure faults. To ensure the TXD pulse width is accurate, a timer should be used. Below is an example code to set up the timer.

Low-level APIs are preferred to configure timer with 2 μ s resolution. Mbed timer provides minimum 8 μ s resolution. In this example TIM2 timer in STM32F303K8 controller was used. Refer to STM32F303K8 programming manual for more details on description of register settings.

```
static TIM_HandleTypeDef s_TimerInstance = { //Creates a timer instance (TIM2)
    . Instance = TIM2
};
__HAL_RCC_TIM2_CLK_ENABLE (); //Enable TIM2 APB clock (72MHz)
s_TimerInstance.Init.Period.Prescaler = 16; //Set the counter prescalar value
s_TimerInstance.Init.CounterMode = TIM_COUNTERMODE_UP; // Set the counter in "UP" mode
s_TimerInstance.Init.Period = 500; //Set the counter period
s_TimerInstance.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1; // Set the clock divisor value to none
s_TimerInstance.Init.RepetitionCounter = 0; //Disable the auto-reload
HAL_TIM_Base_Init(&s_TimerInstance); // Initializes TIM base unit according to specified parameters
HAL_TIM_Base_Start(&s_TimerInstance); //Start the timer in time-base mode
```

In case FAULT goes high, it can go high at any moment within a CAN message transmission. That means the message most likely ends after FAULT is high, therefore reading the error code from the next fault detection cycle is more reliable. After FAULT becomes high, the CAN peripheral pins are configured as GPIOs. The fault detection algorithm will repeatedly generate 2 μ s TXD positive pulses until RXD becomes high after the rising

edge of the FAULT pin. It is recommended to sample RXD after the falling edge of TXD. After RXD becomes high, five more TXD pulses are required to shift out the Fault code. Ten more TXD pulses are used to disable the fault detection. Below is an example code of the algorithm when the FAULT goes high:

```

InterruptIn fault (PA_0);           //Configure Fault pin as interrupt pin
fault.rise(&fault_init);           // Attach an interrupt callback function when fault pin goes high
int state=0;                        // This is the state of state machine for fault detection
void fault_init()
{
    fault.rise (NULL);              //Detach an interrupt till state machine gets completed
    toggle_txd_i_ticker.attach_us(&toggle_txd_i,6); // Initiate a state machine to detect a fault, each cycle is 6us
}
void toggle_txd_i()
{
    DigitalOut txd(PA_12);          //Configure CAN TXD pin as digital o/p
    DigitalIn rxd(PA_11);          // Configure CAN RXD pin as digital i/p
    DigitalIn fault_pin(PA_0);     //Configure fault pin as digital i/p
    int status = fault_pin.read(); //Fault pin status is stored in status variable
    static int count,N;

    do {                            //This code toggles TXD for 2us duration using TIM2 timer
        txd = 1;
    } while ( __HAL_TIM_GET_COUNTER(&s_TimerInstance) < 9);
        txd=0;
        count++;
    switch (state){                 //State machine for fault detection
        case 0: // Ignore the first high fault, giving txd pulses to clear the fault without reading fault code
            if (count >= 26 && status == 0 ) {
                count = 0;
                state = 1;
            }
            break;

        case 1: // Fault pin is low and giving 100 fault pulses/waiting for fault pin to go High for second time
            if (count>=100 && status == 1 ) {
                count = 0;
                state = 2;
            }
            break;

        case 2: // Second time fault activated, need to read the fault code(10 pulses + 6 pulses to read the rxd +
                //10 pulses to clear the fault
            if (status == 1){
                if( (rxd.read() == 1 || rxd_read == 1) && i<6) {
                    arr[k] = rxd.read(); //Read RXD (fault code bit) and store in array
                    k++;
                    rxd_read = 1; //Flag to indicate that fault has been read
                    i++;
                }
            }
            else if ( status == 0 ) { // Once fault pin becomes 0, move to state 3
                fault_read = 1;
                rxd_read = 1;
                count = 0;
                state = 3;
            }
    }
}

```

```

    break;
case 3:
if (status == 0) {
    InterruptIn fault (PA_0);    //Configure fault pin as interrupt pin
    Fault.rise(&faultl_init);    //Enable fault interrupt
    toggle_txd_i_ticker.detach(); //Disable state machine
    }
    break;
}
}

```

For the other two cases (bit-error frame and transmitter error count >255), FAULT does not necessarily become high. The algorithm will configure the CAN peripheral pins as GPIOs and will repeatedly generate 2µs TXD positive pulses until RXD becomes high after the rising edge of FAULT. It is recommended to sample RXD after the falling edge of TXD. After RXD becomes high, five more TXD pulses are required to shift out the Fault code. Ten more TXD pulses disable the fault detection. If FAULT does not go high after 110 TXD pulses, this means no fault was detected and the fault detection mode will exit. Below is an example code of the algorithm:

In STM32F303K8 CAN controller, ESR (Error status register) has Transmitter error counter [TEC] bits 7:0. If this counter exceeds 255, state machine to detect fault will get started. Also ESR has 2 bits, LEC[2:0] (LAST ERROR CODE) to indicate the error condition such as bit-error frame.

```

Int tec_count= ((CAN1->ESR) && (0x00FF0000))>>16;    //Get the transmit error counter value from ESR register
Int error_code= ((CAN1->ESR) && (0X00000070))>>4;    //Get the error code value from ESR register
if((tec_count >255) || (error_code !=0 ))
{
    fault_init();    //Run the fault detection algorithm as implemented in above section
}

```

Trademarks

Mbed is a registered trademark and service mark of Arm Limited.

Related Parts

MAX33011E +5V, 5Mbps CAN Transceiver with ±65V Fault Protection, Fault Detection and Reporting, ±25V CMR, and ±40kV ESD Protection

MAX33012E +5V, 5Mbps CAN Transceiver with ±65V Fault Protection, Fault Detection and Reporting, ±25V CMR, and ±40kV ESD Protection
