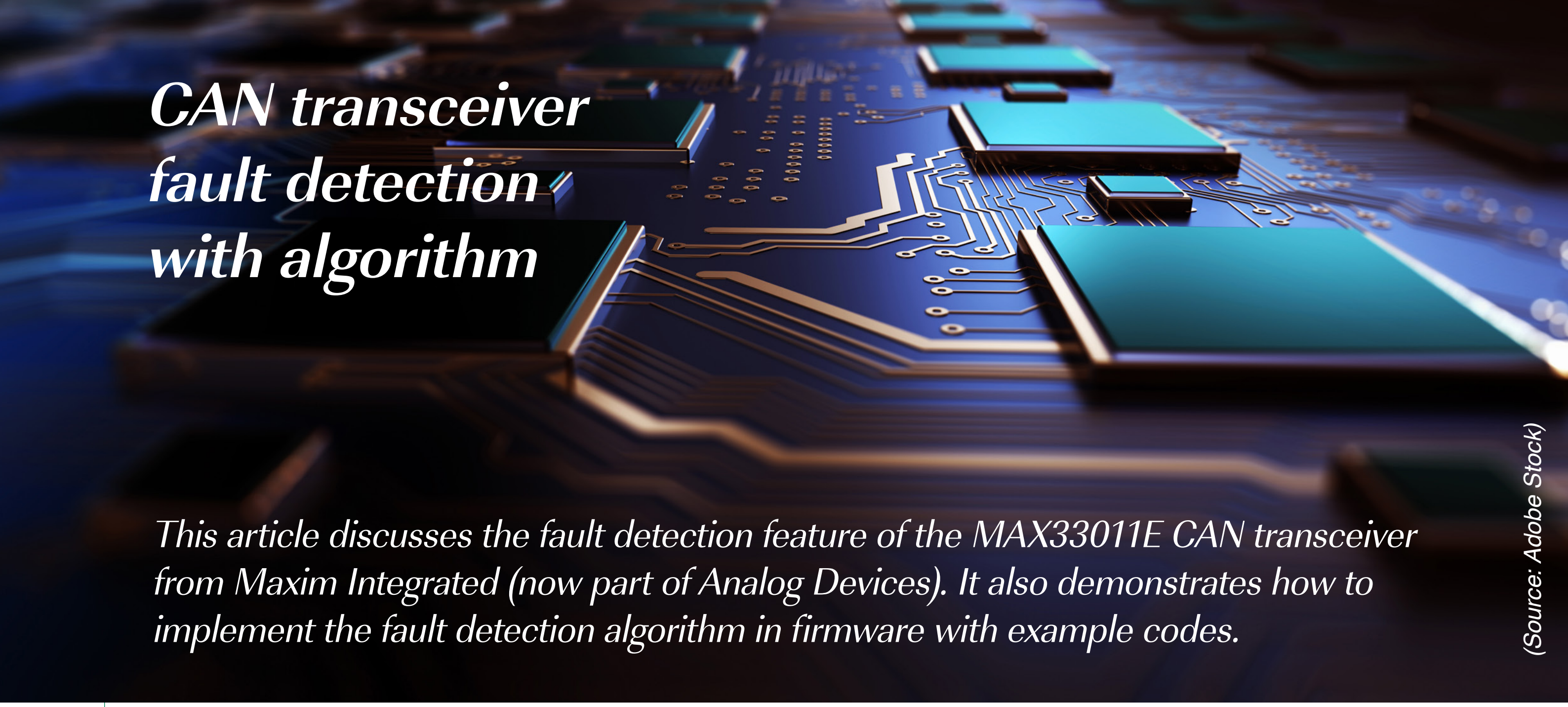

CAN transceiver fault detection with algorithm

This article discusses the fault detection feature of the MAX33011E CAN transceiver from Maxim Integrated (now part of Analog Devices). It also demonstrates how to implement the fault detection algorithm in firmware with example codes.

(Source: Adobe Stock)

Troubleshooting a CAN network when data is not able to be transmitted or received 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.

Fault detection circuit function

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

When a fault condition is detected, the transmitter will be disabled and the Fault pin will be pulled to high through the external pull-up resistor. When the system controller receives the Fault-pin signal, 16 low-to-high transitions on the TXD pin are required to shift out the fault code as shown in Table 1. Additional 10 low-to-high transitions on the TXD pin 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.

Fault conditions

The MAX33011E is the first CAN transceiver with a built-in fault detection circuit, claims the company. 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 line as listed in Table 1.

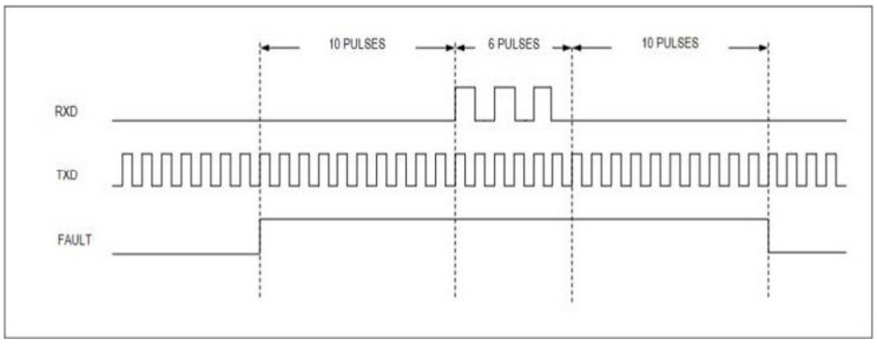

Figure 1: Overcurrent fault reporting timing diagram (Source: Maxim Integrated)

Overcurrent fault

Overcurrent fault is detected when the source current of CANH and the sink current of CANL are both higher than 85 mA (typically). The more probable cause of the fault is that CANH and CANL are shorted on the bus line. However, if the short is far away from the CAN node, it may not be detected due to a high cable impedance. Slowing down the CAN

Table 1: Three types of common fault conditions, which can be detected by the fault detection circuit (Source: Maxim Integrated)

Fault	Condition (Fault Detection Enabled)	Fault Code	Possible Cause
Overcurrent	CANH output current and CANL input current are both > 85 mA	101010	• CANH shorted to CANL • CANH connected to GND and CANL connected to V_{DD}
Overvoltage	CANH > +29 V or CANL < -29 V	101100	• CMR fault
Transmission Failure	RXD unchanged for 10 consecutive TXD pulses, recommended minimum frequency = 200 kHz	110010	• 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

signal frequency could lower the cable impedance and help to detect the short from a further distance. But, if the total resistance of the cable becomes significantly high, a short will not be detected even when the CAN signal is constantly in a dominant state. Figure 2 shows the maximum operating frequency for overcurrent detection versus cable length as a reference. A Cat5E copper clad aluminum cable is used. The maximum frequency will vary with the type of cable.

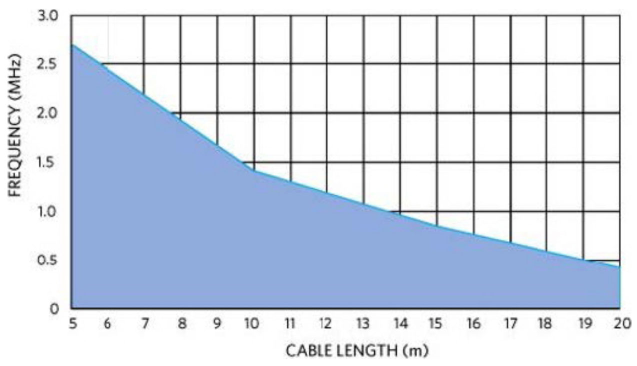

Figure 2: Maximum operating frequency for overcurrent detection vs. cable length (Source: Maxim Integrated)

Overvoltage fault

The MAX33011E common-mode input range (CMR) is ±25 V. An overvoltage fault is detected when either the voltage on CANH is higher than 29 V or at CANL is lower than -29 V. This is caused by the CMR going beyond the specified value.

Transmission failure fault

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

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 line.
2. When the common-mode voltage exceeds the transceiver's common-mode range (-5 V to +10 V), the transceiver is turned off. When the transceiver is off, the CANH/CANL output will not reflect the signal on the TXD pin, and the receiver will see a fixed signal on the CAN bus line.
3. If no termination resistor is connected to the CAN node, it may cause a transmission failure. The termination resistors play a very important role of bringing CANH and CANL to the same voltage level in the 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 line could also slow down CANH and CANL voltages from merging. When the controller sends pulses to the TXD pin, 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 signal stays low for the 10 TXD-signal pulses), a transmission failure fault will be reported. This also means if the TXD-signal high-time is too long, the CAN-bus-line signal could enter the recessive mode and the RXD signal will become high, no transmission failure fault will be reported. The recommended minimum TXD pulse frequency to detect a trans-mission-failure fault, is 200 kHz.

Fault detection algorithm

Maxim has developed an algorithm that can detect fault conditions on the CAN network using the MAX33011E transceiver, without interrupting the normal CAN communication. The following example Mbed codes were developed for the Nucleo-F303K8 platform by ST Microelectronics.

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

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

- Fault-pin signal goes high,
- Transmitter generates a bit-error frame,
- Transmitter error counter rises above 255 and the node enters the bus-off state.

The example code in Figure 3 configures the micro-controller pins as a GPIO when any of the above-mentioned conditions becomes true. In this example, the STM32F303K8 MCU (micro-controller unit) was used as the CAN controller. The example reference code was developed on Mbed-OS, which is a free open-source embedded operating system. Mbed offers APIs (application programming interface) to configure the micro-controller's I/Os as digital input/output pins. Any other similar low-level APIs may also be used.

In the fault detection mode, the 2-µs-TXD positive pulses should be used. After the positive pulse, the TXD signal can stay low as long as needed to process the algorithm. This allows the fault detection circuit to reliably ▷

```
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
```

Figure 3: The example code configures the micro-controller pins as a GPIO when any of the above-mentioned conditions becomes true (Source: Maxim Integrated)

```
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.Prescalar = 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
```

Figure 4: Example code to set up the timer (Source: Maxim Integrated)

```
InterruptIn fault (PA_0);                                    //Configure Fault pin as interrupt pin
fault.rise(&fault_init);                                               // Attach an interrupt callback function when fault pin goes hig
h
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 tim
er
               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(&fautl_init);             //Enable fault interrupt
                toggle_txd_i_ticker.detach();  //Disable state machine
              }
                break;
            }
}
```

Figure 5: Example code of the algorithm when the Fault-pin signal goes high (Source: Maxim Integrated)

```
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 s
ection
       }
```

Figure 6: Example code of the algorithm if bit-error frame and transmitter error count are higher than 255 (Source: Maxim Integrated)

detect both overcurrent and transmission failure faults. To ensure the TXD pulse width is accurate, a timer should be used. Figure 4 shows an example code to set up the timer.

Low-level APIs are preferred to configure the timer with a 2-μs resolution. Mbed timer provides a minimum resolution of 8 μs. In this example, the TIM2 timer in the STM32F303K8 CAN controller was used. More description details on register settings are available in the STM32F303K8 programming manual.

In case the Fault-pin signal goes high, it can go high at any moment within a CAN frame transmission. That means the frame most likely ends after the Fault-pin signal is high, therefore reading the error code from the next fault detection cycle is more reliable. After the Fault-pin signal becomes high, the CAN peripheral pins are configured as GPIOs. The fault detection algorithm will repeatedly generate 2-μs TXD positive pulses until the RXD signal becomes high after the rising edge of the Fault pin. It is recommended to sample the RXD signal after the falling edge of the TXD signal. After the RXD signal 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. Figure 5 shows an example code of the algorithm when the Fault-pin signal goes high.

For the other two cases (bit-error frame and transmitter error count >255), the Fault-pin signal does not necessarily become high. The algorithm will configure the CAN peripheral pins as GPIOs and will repeatedly generate 2-μs TXD positive pulses until the RXD-signal becomes high after the rising edge of the Fault-pin signal. It is recommended to sample the RXD-signal after the falling edge of the TXD-signal. After the RXD-signal becomes high, five more TXD-pulses are required to shift out the Fault code. Ten more TXD-pulses disable the fault detection. If Fault-pin signal does not go high after 110 TXD-pulses, this means no fault was detected and the fault detection mode will be exited. Figure 6 shows an example code of the algorithm.

In the STM32F303K8 CAN cotroller, the ESR (error status register) has the transmitter error counter [TEC] bits 7:0. If this counter exceeds 255, the state machine to detect faults will get started. Also the ESR has 2 bits, LEC [2:0] (last error code) to indicate the error condition such as a bit-error frame. ◀

of

Source

Maxim Integrated (Analog Devices)
Application note: https://www.maximintegrated.com/en/design/technical-documents/app-notes/7/7333.html
www.maximintegrated.com

DIS sensors

Dynamic inclinometer

AVAILABLE NOW

Stable real-time measurement, even during rapid movement

- Gyro-compensated
- Based on MEMS technology
- Measuring range: 1 axis up to 0 - 360°, 2 axes up to ± 90°
- CANopen or J1939 interface (data and configuration)
- User-friendly configuration tool
- Advanced hardware and software technology
- Compact, robust housing

DIS Sensors bv | Oostergracht 40 | 3763 LZ Soest | The Netherlands | +31 (0)35 - 603 81 81

www.dis-sensors.com