Transceiver Handling

Version 1.2
2007-09-17
Application Note AN-ISC-2-1029

Author(s) Hussain Darwish
Restrictions Restricted membership
Abstract This application note describes how to handle the transceiver before or after calling Vector's API.

Table of Contents

1.0 Introduction

In communications, transceivers are used to convert the voltage level of the signal to a different signal form such as lights, frequency or even different voltage level. the appropriate bus voltage level to communicate through a medium. to be more resistant to transient noise, and reduce the Electromagnetic Emission (EME). Transceivers also detect collisions in the bus. There are different transceivers for different applications. For instance, the automotive industry uses three types of transceivers:

1. **High Speed Transceivers (ISO 11898-2)**: the most common type used in the automotive industry. They are mostly used in power train buses and can handle baud rates up to 1Mbps.
2. **Fault-Tolerant Transceivers (ISO 11898-3)**: used mostly in slow baud rate buses like body bus and can handle baud rates up to 125kbps. The major advantage of this type of transceivers is that they can communicate without errors even if one bus line gets disconnected, shorted to ground or battery. Also, they can communicate without errors if both bus lines are shorted together. However, they cannot communicate if both bus lines are disconnected, shorted to ground or battery. Up to 32 nodes can be connected to the fault-tolerant bus.

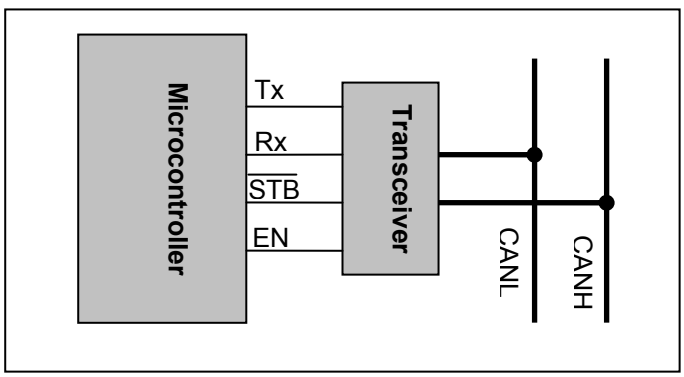

Figure 1: High-Speed and Fault-Tolerant Transceiver Wiring

When using Vector's embedded software, the user has to switch the transceiver to a certain mode before or after calling specific functions. In the following sections, the transceiver handling is explained.

2.0 Transceiver Handling

Essentially, the transceiver shall be handled in three situations:

1. During initialization (after a reset)
2. After putting the CAN cell into sleep mode
3. During a wakeup

2.1 Initialization

The transceiver must be switched into Normal mode before calling the CAN driver initialization function **`CanInitPowerOn()`**. Some CAN cells needs to monitor 11 consecutive recessive bits to initialize (or enter configuration mode). If the CAN transceiver is not in Normal mode, it may assert the Rx line to a dominant state. Consequently, if the transceiver is not in Normal mode and the function **`CanInitPowerOn()`** (or **`CanInit()`**) is called, the function may not finish executing as it is waiting for 11 recessive bits to synchronize the CAN cell to the bus. Some CAN cells do not need to monitor any recessive bit to initialize and thus the transceiver do not have to be in Normal mode before calling the CAN driver initialization function. However, for consistency and portability reasons, we strongly recommend that the transceiver is in Normal mode before calling the initialization function.

2.2 Sleep Mode

When using the sleep mode feature of the CAN cell, make sure to switch the transceiver into low-power or standby mode AFTER switching the CAN cell into sleep mode. In other words, the transceiver should be switched into low-power or standby mode only after calling the function **`CanSleep()`**. If the function **`CanSleep()`** returns **`kCanFailed`**, the application must not put the transceiver into low-power or standby mode.

2.3 Wakeup

When the application wants to wake up the CAN cell to transmit a message, he/she shall switch the transceiver into Normal mode before waking the CAN cell (or calling the function **`CanWakeUp()`**), Some CAN cells cannot wakeup properly until they monitor 11 consecutive recessive bits in the bus. If the transceiver is not in Normal state, it may assert the Rx line to dominant state (due to a received wakeup signal from the CAN bus). Consequently, the **`CanWakeUp()`** function may not finish executing as it is waiting for 11 recessive bits to synchronize the CAN cell to the bus.

3.0 Special Considerations

3.1 Possible Deadlock

Make sure that the CAN cell is in sleep mode when the transceiver is in sleep or standby mode. If the CAN cell is not in sleep mode and the transceiver is in sleep or standby mode, activities from the bus will never be detected. This is because the CAN cell wakes up from the sleep mode when it detects a dominant bit on the Rx pin. When the transceiver is in sleep or standby mode, it will try to wake up the CAN cell by signaling a dominant level on the Rx pin. From there, the microcontroller will send a wakeup event to the application and the application will switch the transceiver into Normal mode.

On the other hand, if the transceiver is in sleep or standby mode while the CAN cell is in Normal operation mode, the microcontroller will never detect a wakeup event as the CAN cell is not in sleep mode. Since the wakeup event

is not detected nor passed to the application (via the function **`ApplCanWakeUp()`**), the application may never be able to switch the transceiver into Normal mode. Consequently, the application will not be able to transmit nor receive CAN messages on the bus and may stick in a deadlock situation.

3.2 Return Value of the Function CanSleep()

As a result of the possible deadlock mentioned in the previous section, the application must check the return value of the function **`CanSleep()`** when calling it. If **`CanSleep()`** returns **`kCanOk`**, the application can put the transceiver into sleep or standby mode. If **`CanSleep()`** returns **`kCanFailed`**, the state of the CAN cell might be unknown and thus the application must not put the transceiver into sleep or standby mode. In this case, the CAN cell might be in sleep or normal operation mode depending on the hardware. What to be done if **`CanSleep()`** returns **`kCanFailed`** is OEM specific.

Pseudocode example:

```
if(CanSleep() == kCanOk)
{
    DeactivateTransceiver(); /* Application function to put the
                                transceiver into standby or
                                sleep mode */
}
```

4.0 CAN Communication Layer (CCL)

To make the implementation more consistent, Vector offers a layer called ‘Communication Control Layer (CCL)’ that takes care of transceiver handling (including initialization, sleep, wakeup, and power-off). In addition, this layer controls all other Vector’s software layers initializations and tasks. In other words, CCL is an abstraction layer that provides a more consistent interface to the application.

For more information on the CCL layer, please visit our website or contact us.

5.0 Contacts

Vector Informatik GmbH
Ingersheimer Straße 24
70499 Stuttgart
Germany
Tel.: +49 711-80670-0
Fax: +49 711-80670-111
Email: info@vector-informatik.de

Vector CANtech, Inc.
39500 Orchard Hill Pl., Ste 550
Novi, MI 48375
USA
Tel: +1-248-449-9290
Fax: +1-248-449-9704
Email: info@vector-cantech.com

VecScan AB
Lindholmspiren 5
402 78 Göteborg
Sweden
Tel: +46 (0)31 764 76 00
Fax: +46 (0)31 764 76 19
Email: info@vecscan.com

Vector France SAS
168 Boulevard Camélinat
92240 Malakoff
France
Tel: +33 (0)1 42 31 40 00
Fax: +33 (0)1 42 31 40 09
Email: information@vector-france.fr

Vector Japan Co. Ltd.
Seafort Square Center Bld. 18F
2-3-12, Higashi-shinagawa,
Shinagawa-ku
J-140-0002 Tokyo
Tel.: +81 3 5769 6970
Fax: +81 3 5769 6975
Email: info@vector-japan.co.jp