CANbedded and Operating Systems

Version 1.0
2007-01-18

Application Note AN-ISC-2-1052

Author(s)	Hartmut Hörner, Patrick Markl

Restrictions	Restricted Membership

Abstract	The Vector CANbedded software components have some requirements for the run-time environment. These requirements are described in this application note.

Table of Contents

1.0 Overview

The Vector CANbedded components have some requirements for the run-time environment. Some of these might impact the usage of the operating system elements as well as the real-time design.

Two run-time environments are explicitly supported by CANbedded:

- A system without operating system
- An OSEK Operating System (version 2.1r1 onwards) such as Vector osCAN

In the following chapters the requirements are listed and for some cases where no explicitly support exists workarounds are described.

2.0 General Requirements

Higher layer CANbedded components require an initialization function and one or more cyclic functions. CANbedded hardware drivers have an initialization function, ISRs in interrupt driven mode and cyclic functions in polling mode. The initialization function must be called before the cyclic function is called for the first time and before the first ISR occurs.

The CANbedded stack requires a shared address space with the application since some functionalities such as signal and flag access are implemented by macros.

There are no requirements for dynamic memory management. All not constant global variables are explicitly initialized in the initialization functions. There are no requirements for memory initialization in the start-up code unless constant data is located in RAM.

Contact Information: www.vector-informatik.com or ++49-711-80 670-0

Since the CANbedded components have to interface directly with the communication hardware adequate access privileges are required.

3.0 Interrupt Handling

3.1 Interrupt Service Routines (ISRs)

If the CAN Driver is used in an interrupt driven mode one or more ISRs must be installed.

Two categories of ISRs are supported:

- ISR implemented with the related compiler pragma or qualifier. This is used in a system without operating system and for OSEK-OS ISRs of category 1.
- OSEK-OS ISR category 2.

If an operating system other than OSEK is used the following workaround is possible:

If OSEK-OS and OSEK-OS category 2 ISR support are selected in the generation tool the ISR has the following C syntax:

```
ISR(CanInterrupt)
{
  …
}
```

Supply a header file osek.h with the following macro, to make the ISR available in other modules as a void-void function.

```
#define ISR(x) void x(void)
```

Example for an operating system which provides a service OSRegisterISR to register an ISR:

```
OSRegisterISR(CanInterrupt, 10);     /* second parameter is interrupt vector number */
```

If the prototype of the ISR required by the OS is not of the kind void x(void) an intermediate function must be used to circumvent type conflicts.

3.2 Access to interrupt control registers

To achieve its own data consistency and to export such services to the application the CANbedded components require access to the interrupt control registers of the communication controller and to interrupt control registers of the interrupt controller or the CPU which allow to disable interrupts globally or up to a certain level.

If it is not desirable or possible that the CANbedded components access the global interrupt directly this functionality can be replaced by other mechanisms (“interrupt control by application”, supported in CAN driver RI version 1.4 onwards).

If a resource locking mechanism of the operating system is used it must support nested critical sections.

4.0 Data consistency

For efficiency reasons some services of the CANbedded components are implemented as macros with unprotected critical sections. If these macros are used in multiple full-preemptive tasks or in multiple ISRs data consistency problems can be caused. One of the following solutions is possible:

- Use these services only in non-preemptable tasks
- Use these services only in one cooperative task group (OSEK: internal resource concept)
- Supply critical section in the application code

Some CANbedded components have specific requirements concerning processing priorities and reentrancy. These requirements are described in the technical reference of the individual component.

5.0 Memory protection and virtual memory management

This chapter is only applicable if the processor and the run-time environment support such mechanisms.

Since some CANbedded services are implemented as macros which access to global variables the application part which uses the CANbedded services and the complete CANbedded stack must reside in one address space.

In addition all CANbedded tasks and ISRs must be able to access the peripheral address space of the communication hardware.

The following figure shows a possible configuration with three different applications with an own address space. Two applications perform application tasks (drawn in green), one application is responsible for the communication (drawn in red).

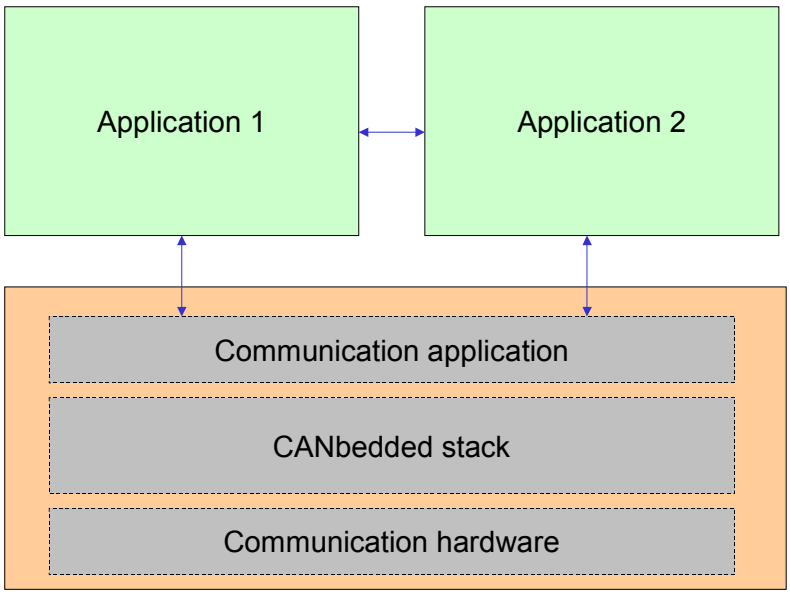

Figure 1 Different applications with own address space

To move data between the applications operating system services are required which have the capability to tunnel the address barrier (drawn with blue arrows)..

6.0 Privilege modes

This chapter is only applicable if the processor and the run-time environment support such mechanisms.

In some architectures different privilege modes such as user and supervisor mode are defined. If a task or ISR is running in user mode the access to peripheral hardware and global resources such as interrupt control registers may be restricted.

It must be possible to assign all tasks and ISRs which provide or use CANbedded services a sufficient privilege level to allow for:

- Access to communication hardware
- Access to communication related interrupt control
- Access to global interrupt control registers including the capability to execute instructions related to global interrupt control registers unless interrupt control by application is used (s. chapter 3.2).

If such a privilege level is not possible for tasks but only for ISRs a cyclic timer ISR can be used to perform all required CANbedded cyclic processing. Note that such an ISR must have a lower interrupt level than the ISRs of the CANbedded drivers.

7.0 Operating systems with stringent driver models

This chapter is only applicable if the run-time environment supports such mechanisms.

In some operating systems hardware access is only allowed for specific hardware drivers which have to follow a predefined implementation model. When a hardware driver is called in such a system the required privileges are assigned. To allow the usage of CANbedded a wrapper layer between application and CANbedded is required to implement the required driver interface.

Such a design is illustrated in the figure below, the driver interface is drawn with blue arrows:

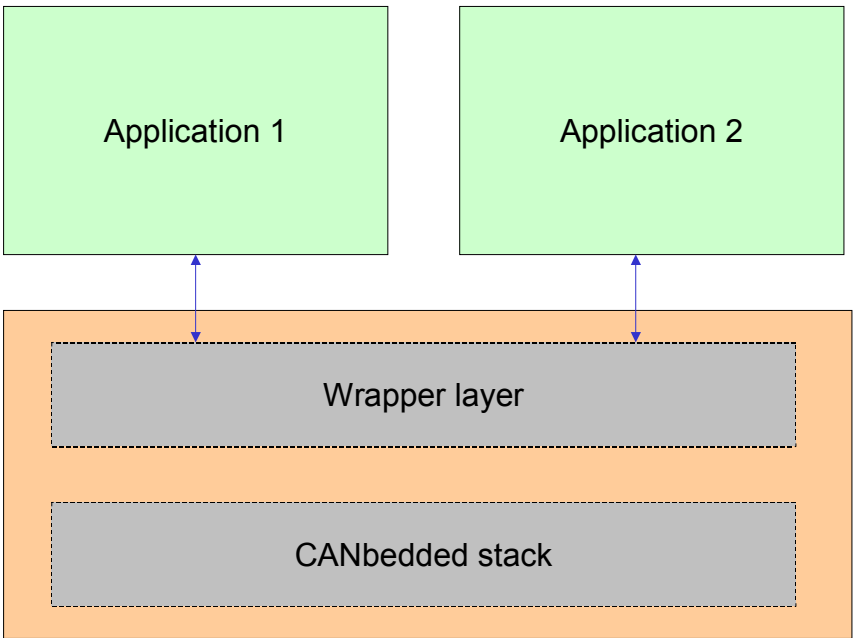

Figure 2 With Wrapper Layer

8.0 Startup Time

Some Operating systems (especially more sophisticated and larger ones) are running out of RAM, while the binaries are stored in non-volatile memories like Flash. This requires that the startup code transfers the OS image into RAM after power on reset. Depending on the size of the OS image this can take up to several hundreds of milliseconds. As a consequence the initialization of the CANbedded stack occurs after the kernel is loaded completely. Often this long startup times do not match the OEMs' startup requirements. There are two possibilities to solve this issue. Either the OEM accepts the longer startup time or the operating system supports a mechanism that allows initialization of the CANbedded stack before the kernel is loaded completely.

If the operating system supports some kind of a runtime environment before the kernel is loaded it could be also relevant to have a possibility to hand over the information received during the startup phase via CAN to the fully loaded kernel environment.

9.0 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 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 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

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

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