VECTOR

Distributing BSW Components across Partitions

Version 1.2
2017-05-17
Application Note AN-ISC-8-1196

Author Wolf, Jonas

Restrictions Customer Confidential – Vector decides

Abstract MICROSAR BSW is supposed to be executed in one memory partition (with some exceptions). Architectural constraints, e.g. a safety concept, may require distributing MICROSAR BSW across different memory partitions. This application note describes the general approach and common techniques for distribution.

Table of Contents

1 Overview

AUTOSAR was initially developed for single core microcontrollers without memory protection unit (MPU). The availability of more powerful microcontrollers and the introduction of ISO 26262 led to new concepts in AUTOSAR addressing these changes.

This document provides an approach how to distribute basic software across partitions on one core. Distributing the basic software across different cores is out of scope of this application note. The term basic software is used in this application note for all software components below the Runtime Environment (RTE), i.e. AUTOSAR modules, like ECUM or COM as well as complex drivers (CDs).

Above the RTE, i.e. for application software components (SWC), partitioning is usually automatically handled by configuration of the RTE. Thus, this is also out of scope for this application note.

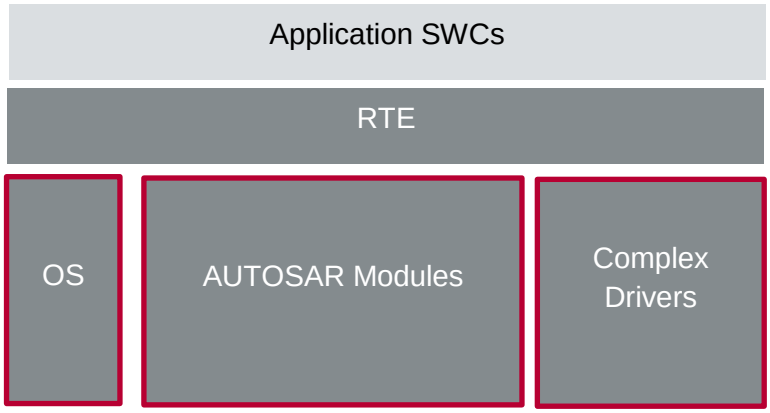

Figure 1-1 Basic Software

Apart from a few exceptions it is not possible to distribute the basic software into different partitions by configuration. This application note describes an approach for partitioning the basic software. It details the available techniques for implementing the partition crossing and shows an example use-case.

An AUTOSAR operating system with Scalability Class 3 (SC3) and an MPU is required for memory partitioning. The term partition and OS Application are used synonymously.

2 Introduction of Partitioning

Use the following three step approach to introduce working and effective partitioning of the basic software:

1. Define partitioning scheme

   Use the software architecture of the ECU to describe the partitions and assign all SWCs and basic software components to a partition. Assignment to a partition is mainly defined by your safety concept. Partitions can be used if software components with different quality levels are to be executed on one ECU.

2. Identify interfaces between components

   Use the information from the Technical References and the software architecture of the ECU to identify the interfaces between basic software components. Interfaces between software components are called functions and shared data.

   Make yourself aware of buffer ownership and also assign any buffers to a partition.

3. Select technique to implement partition crossing

   Select one of the techniques from Section 3 to implement a crossing of partitions.

Note
A lot of interfaces between BSW modules can be deactivated by configuration. Evaluate whether deactivation of the interface is an option and deactivate the interface if possible. This may especially apply for interfaces to DEM and DET. If the interface is deactivated, there is no need to introduce cross-partition calls.

3 Techniques for Crossing Partitions

3.1 Trusted Functions

Trusted Functions are services exported by trusted applications for the use by other applications and are realized by the operating system (OS). Calling Trusted Functions usually implies switching from user mode to supervisor mode (and back when returning).

Trusted Functions are executed on the stack of the caller.

AUTOSAR specifies that interrupts have to be enabled when calling a Trusted Function. To ensure data consistency for an exclusive area in which the Trusted Function is called, OS resources can be used instead of disabling interrupts. This restriction does not apply to versions of Generation7 of MICROSAR OS. They do not require enabled interrupts when calling Trusted Functions.

For more information on Trusted Functions see AUTOSAR SWS OS (Version 4.3, Section 8.4.4) and Technical Reference MICROSAR OS (e.g. Version 1.8.0, Section 3.3).

3.2 Non-trusted Functions

Non-trusted Functions are services exported by non-trusted OS applications for the use by other applications and are realized by the operating system (OS). They have the following characteristic:

- They run in user mode.
- They run with the MPU access rights of the owning OS application.
- They perform a stack switch to specific and secured Non-trusted Function stacks.

Parameters are passed to the Non-trusted Function with a reference to a data structure provided by the caller. Returning of values is only possible if the caller passes the Non-trusted Functions parameters as pointer to global accessible data.

3.3 Inter OS Application Communicator (IOC)

If available in the operating system, IOC can be used to communicate between components as well. Since effort for configuration is higher than for the previously described techniques and there is only a benefit for rare use-cases, usage of IOC is not further described. The principles described for Non-trusted Functions, Trusted Functions and hooking into interfaces (see Section 4) applies.

3.4 Sharing Data

It is in general not acceptable for two different partitions that comprise software of different quality levels to have write access to the same data in memory. Thus, usage of a shared buffer is usually avoided.

An alternative is to restrict write access to the applications' memory and use a buffer for each direction of communication. The sender has read and write access to the buffer. The receiver only has read access to the buffer. Read and write access can be enforced by configuring the MPU respectively.

To signal updates to the sending buffer, both the sender and the receiver hold a flag. If the sender puts in new data it toggles the flag on sender side. The receiver can detect new data if its own flag and the flag of the sender differ. After reading, the receiver toggles its own flag. This way, the sender can

detect if data was read by the receiver. The two flags are readable from sender and receiver, but writable only by their respective owner.

4 Hooking into Interfaces

Usually Trusted Functions and Non-trusted Functions are not used by AUTOSAR basic software components. They can, however, be introduced manually by using the technique shown in the following example.

There are two components A and B. A calls a function in B ( ). Both components are assigned to different partitions. A is allocated to a non-trusted partition. B is allocated to a trusted partition. Thus, the function call from A to B needs to be redefined to a Trusted Function call.

The original call and include graph are shown in Figure 4-1 and Figure 4-2.

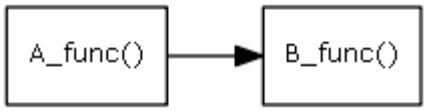

Figure 4-1 Original call graph

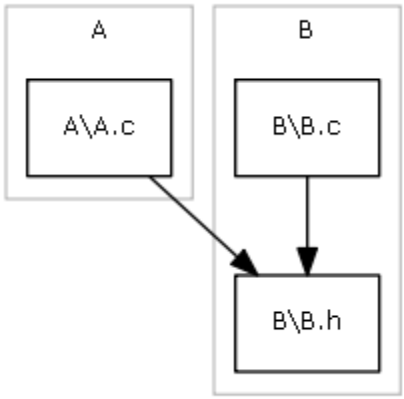

Figure 4-2 Original include graph

Redefinition of functions requires the introduction of a new component Bmod. Bmod will redefine the to a different function name with the same signature, i.e. the Trusted Function configured in the operating system. In this example the Trusted Function call is named . The Trusted Function call in the operating system switches the memory protection to settings defined for the trusted application and finally calls .

Figure 4-3 shows the modified call graph.

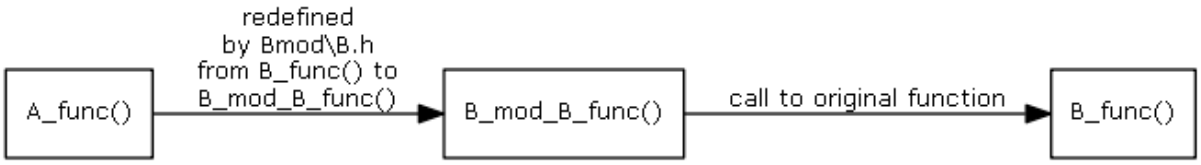

Figure 4-3 Modified call graph

To make A call a function in Bmod instead of B, the include graph needs to modified as well. Figure 4-4 shows the modified include graph.

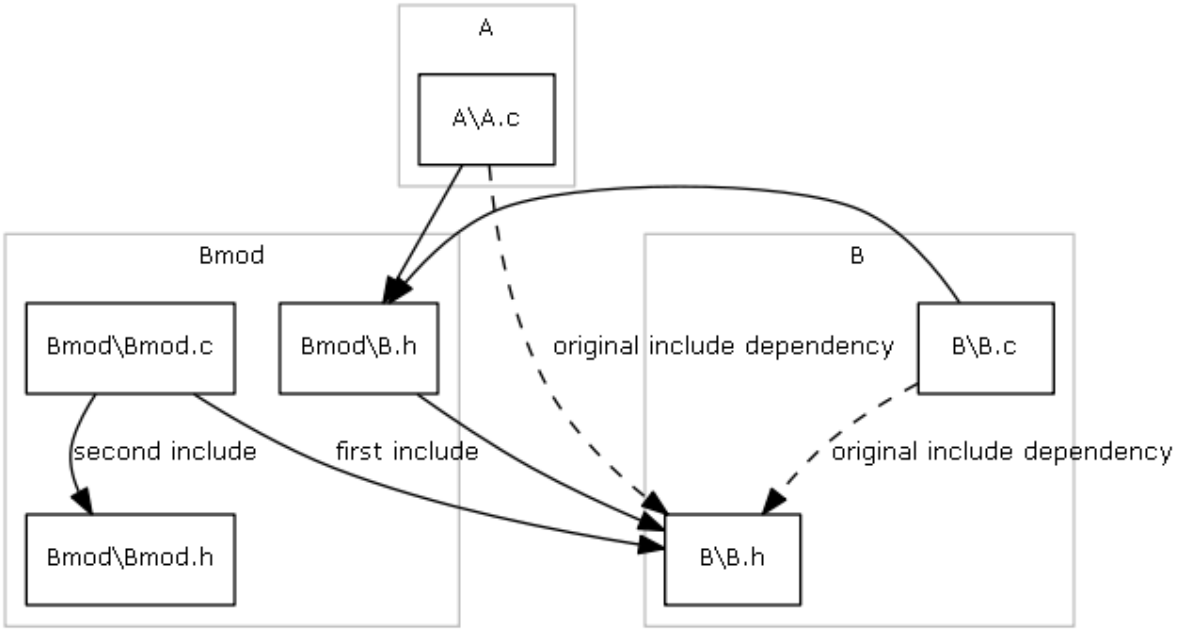

Figure 4-4 Modified include graph

To make this modification to the include graph the include path that pointed to B is modified to point to Bmod. Please note that in Bmod\B.h the include path is resolved in the header file itself and not using include path compiler options (see line 8 of Bmod\B.h below).

The approach described above can also be used if no preprocessor define in the source file exists. The build system can also be used to set those defines individually for each source file, e.g. using the –D compiler option available for several compilers. All Vector components provide such a define in source code (see line 3 of B.c below).

4.1 Example Code

4.1.1 Source File of A (A\A.c)

4.1.2 Header File of B (B\B.h)

4.1.3 Source File of B (B\B.c)

4.1.4 Header File of Bmod (Bmod\Bmod.h)

4.1.5 Header File of Bmod (Bmod\B.h)

4.1.6 Source File of Bmod (Bmod\Bmod.c)

5 Additional Resources

Intentionally left empty.

6 Contacts

For a full list with all Vector locations and addresses worldwide, please visit http://vector.com/contact/.