Synchronization between AUTOSAR Cry and Fls

Version 1.00.03
2017-08-14
Application Note AN-ISC-8-1207

Author
Restrictions
Abstract

Table of Contents

1 Introduction

AUTOSAR defines Crypto Driver (Cry) and Flash Driver (Fls) as MCAL modules for independent hardware modules. But as the crypto module also provides key storage functionality, this is often implemented in the microcontroller by using a shared flash unit. This can lead to race conditions, if AUTOSAR Crypto Driver and Flash Driver are used concurrently.

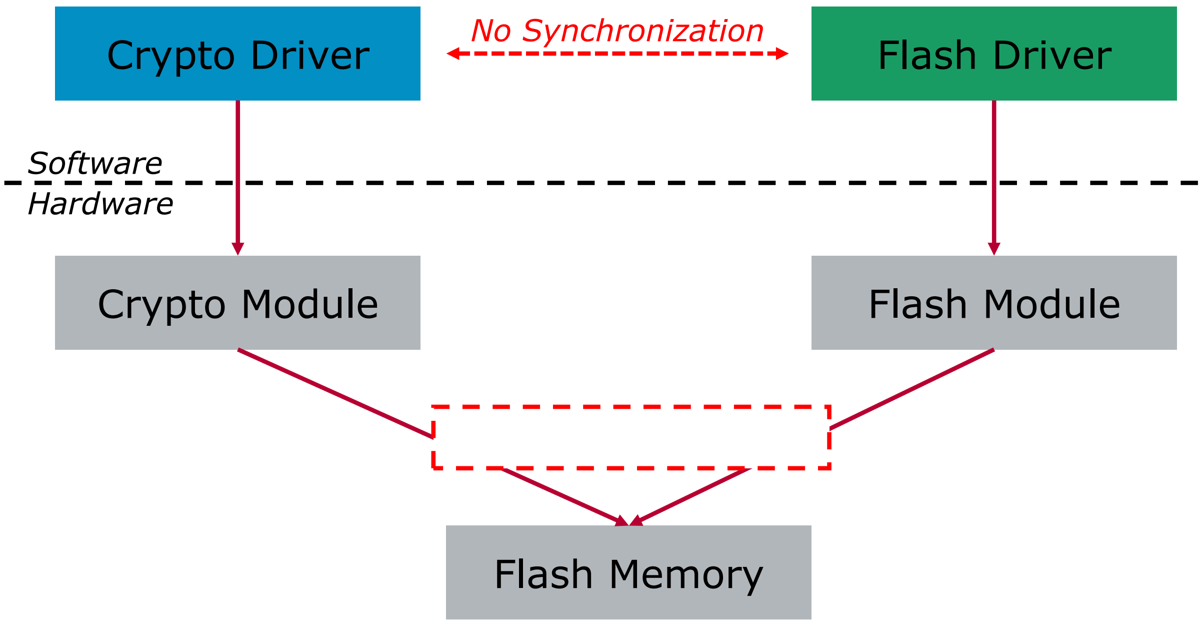

Figure 1 - Race Condition during Flash Memory Access

This application note describes several synchronization methods and gives advice which method should be applied. The synchronization itself must be implemented by the application during the integration of the MICROSAR stack, as the used method depends on the system layout and requirements.

Caution

-

Caution

1.1 Access Conflict Types

Table 1 lists possible access conflict types. In this application note, it is assumed that each of these combinations will lead to a race condition and must be prevented. E.g. even if the Crypto driver performs only a read access to the key (like MacVerify), a parallel read access of the flash driver to the hardware is not allowed.

If the used microcontroller is less restrictive, a less restrictive synchronization scheme might be applied. Such optimized synchronization scheme is not covered by this application note.

Crypto Driver	Flash Driver	Conflict

Table 1 - Access Conflict Types

1.2 Dependency of Access Conflics to the ECU use case

The possibility of an access conflict heavily depends on the ECU use case. As an access conflict can only happen if the Flash module and Crypto module are used at the same time, a detailed analysis of this use case is needed.

> E.g. if the Crypto module is used only during ECU run state for secure communication, whereas Flash module access is limited to ECU startup and shutdown, no access conflict might occur.
> The access conflict might be limited to system startup, in case the Crypto module can cache the keys in secure RAM and therefore has no need to access data flash during later usage.

The potential access conflicts need to be analyzed by the user. This application note gives an example how to solve such a conflict for secure communication and data flash access.

Caution

2 Use Case Description

The synchronization methods are discussed based on three different users of Crypto and Flash Driver. Please adapt the described concepts in case of a different use case shall be implemented.

Caution

-

User	Functionality	Call context of hardware access

Table 2 - Users of Crypto Driver and Flash Driver

The Fls is used asynchronously: the flash operation is started in , but will be finished later.

The Cry is used synchronously: the crypto hardware is idle after the execution of and is completed.

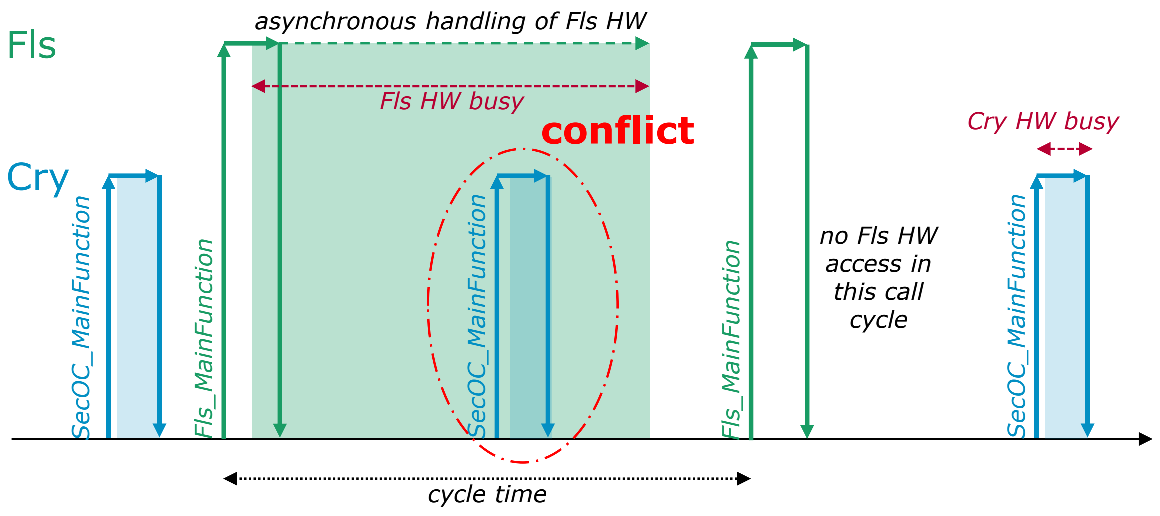

Figure 2 - Conflict between Fls and Cry

3 Synchronization Methods of SecOC and Fls

This chapter describes synchronization methods between Fls and Cry for the SecOC use case.

Caution
It is assumed that cannot interrupt . E.g. is mapped to the same or a lower priority OS task as .

Table 3 gives an overview and comparison of the different methods which are described in this chapter.

Method	Characteristics	AUTOSAR Extensions

Table 3 Comparison Matrix

3.1 Prevent Execution of SecOC_MainFunction

The current state of Fls can be polled by the function . Therefore should not be called in case

Caution

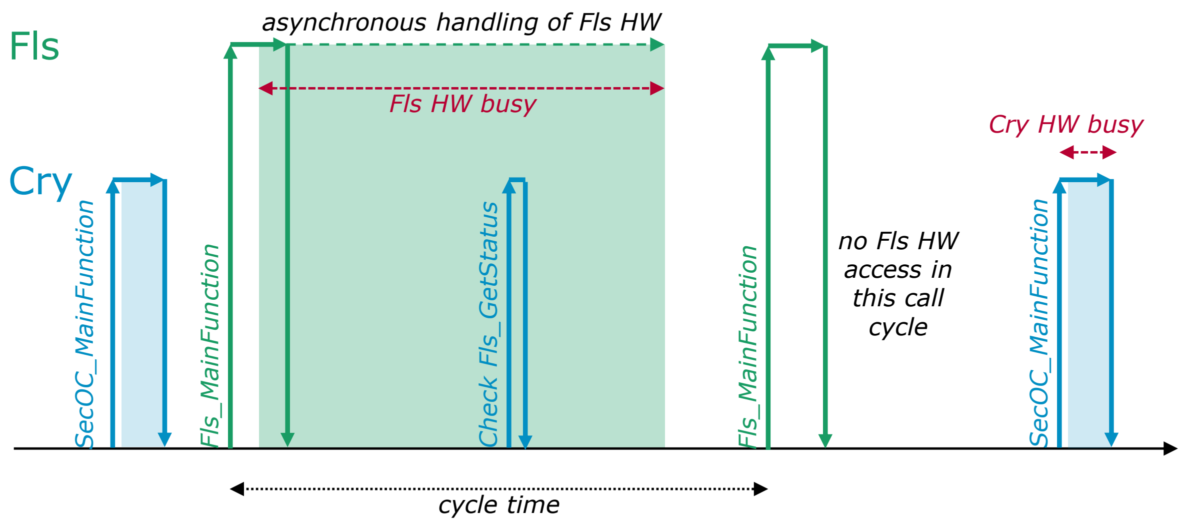

Figure 3 - Prevent Execution of SecOC_MainFunction

3.2 Prevent Execution of Cry Function

This approach is similar to chapter 3.1. But instead of preventing the execution of , an API like would be used. With the API , the read permission is checked by the Cry driver: this shall be denied in case the Fls is busy.

The will retry with the next cyclic call, if the parameter in SecOC is set accordingly.

Note

3.3 Optimized Execution Prevention

The API returns the status of the Fls driver. Whereas this is implementation-specific, it might be only updated in context of . Therefore it would report whereas the operation in the Fls hardware might be already finished.

It is assumed an API like is available, which reads the status of the Fls hardware itself. An optimized version of methods in chapter 3.1 and chapter 3.2 can be applied in this case:

> In task scheduling, is executed immediately before . Therefore the Fls operation from the previous might be already completed.
> The API is used instead of

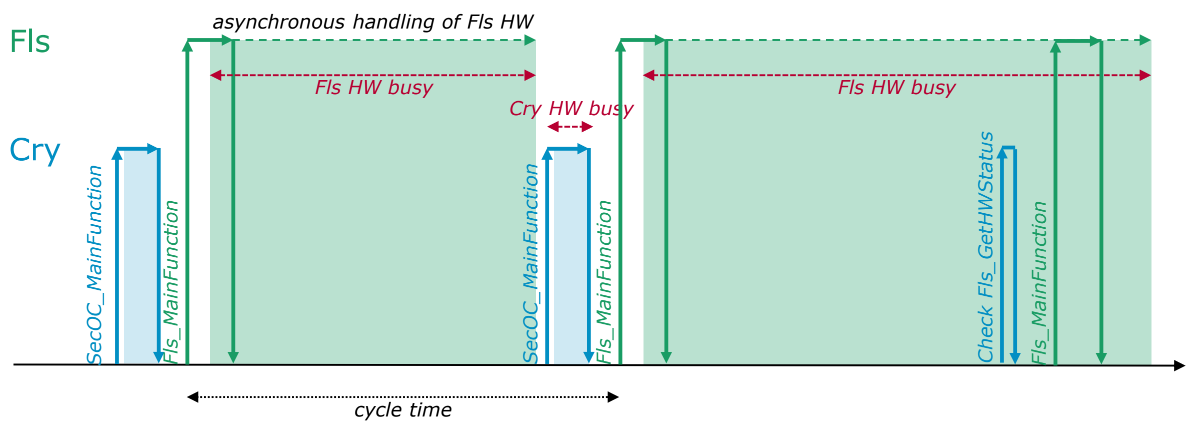

Figure 4 - Optimized Execution Prevention

With this approach, Cry handling is only skipped if the Fls needs a long processing time.

3.4 Interrupt Fls Operation

Some Fls drivers have a possibility to suspend and resume the current Fls operation. Such functionality is used for this approach:

> In task scheduling, is executed immediately before . Therefore the Fls operation from the previous might be already completed.
> In case Fls is busy: Fls operation is suspended before is called
> If Fls operation was suspended: resume Fls operation after is completed

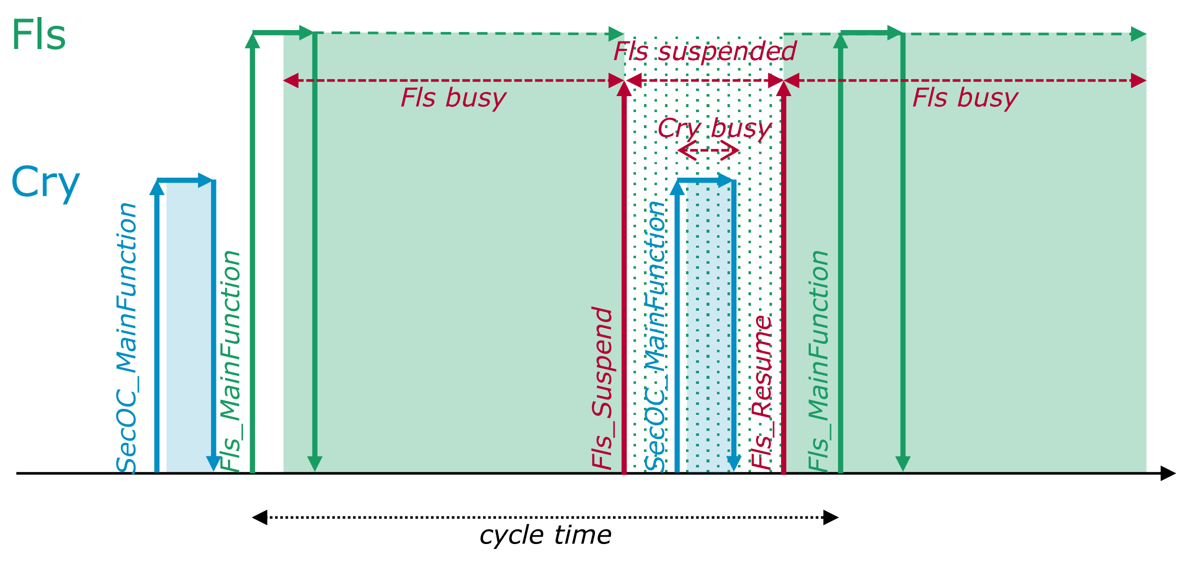

Figure 5 - Interrupt Fls Operation

Caution

3.5 Asynchronous Operation of SecOC

In case needs a long processing time, a full-preemptive operating system might be used during waiting time.

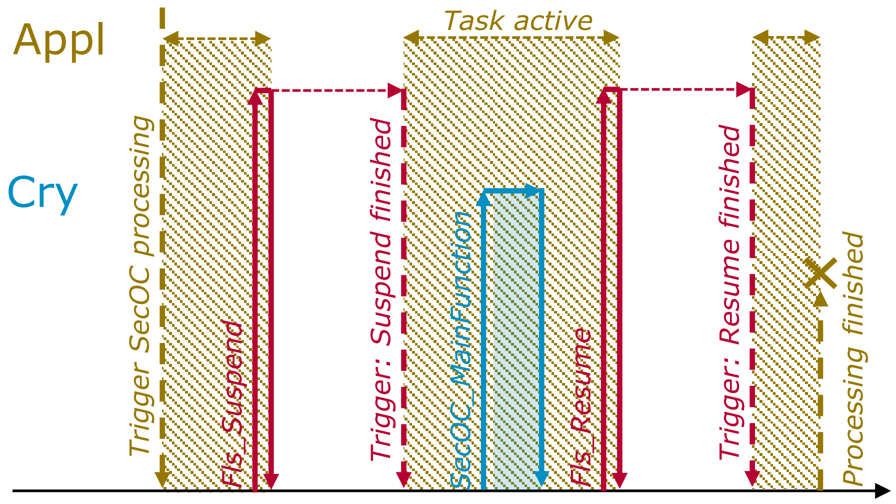

Figure 6 - Asynchronous Operation of SecOC

is called by a separated extended OS task. Following example code illustrates the concept.

SecOC_Task:

The Fls must inform the when is finished.

Suspend_Finished:

Resume_Finished:

The application can trigger either cyclic, or only if SecOC processing is needed.

The transmission trigger can be obtained in following way.

ComIPduCallout: //get transmission condition added for all secured messages in Com

After call to Com_MainFunction:

The reception trigger can be obtained in following way. Please make sure the CAN driver is operated in interrupt mode.

CanGenericPrecopy: // enable "CanGenericPrecopy" feature in CAN driver

The OS Event is set before the message data is transferred to SecOC. But the is activated only after the CAN ISR processing is finished due to OS priority handling.

4 Synchronization Method for Key Management

Synchronization between SecOC and Key Management is not needed, as both utilize the same Cry driver. Therefore the Cry driver should take care of this synchronization.

Caution
It is assumed that Fls_MainFunction cannot interrupt KM_MainFunction. E.g. Fls_MainFunction is mapped to the same or a lower priority OS task as KM_MainFunction.

Synchronization of Key Management and Fls is less time critical than the synchronization between SecOC and Fls. Therefore simpler methods as described in chapter 3.1 and chapter 3.2 can be applied.

If chapter 3.2 is applied, a callback form Cry driver which queries the write permission should be used. Such an API is implemented as an AUTOSAR extension for some Cry drivers developed by Vector.

5 Contacts

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