AUTOSAR && MCAL介绍

AMP GPIS AE

Stephen Du

DEC 2019

NXP

SECURE CONNECTIONS
FOR A SMARTER WORLD

目录

AUTOSAR
是啥

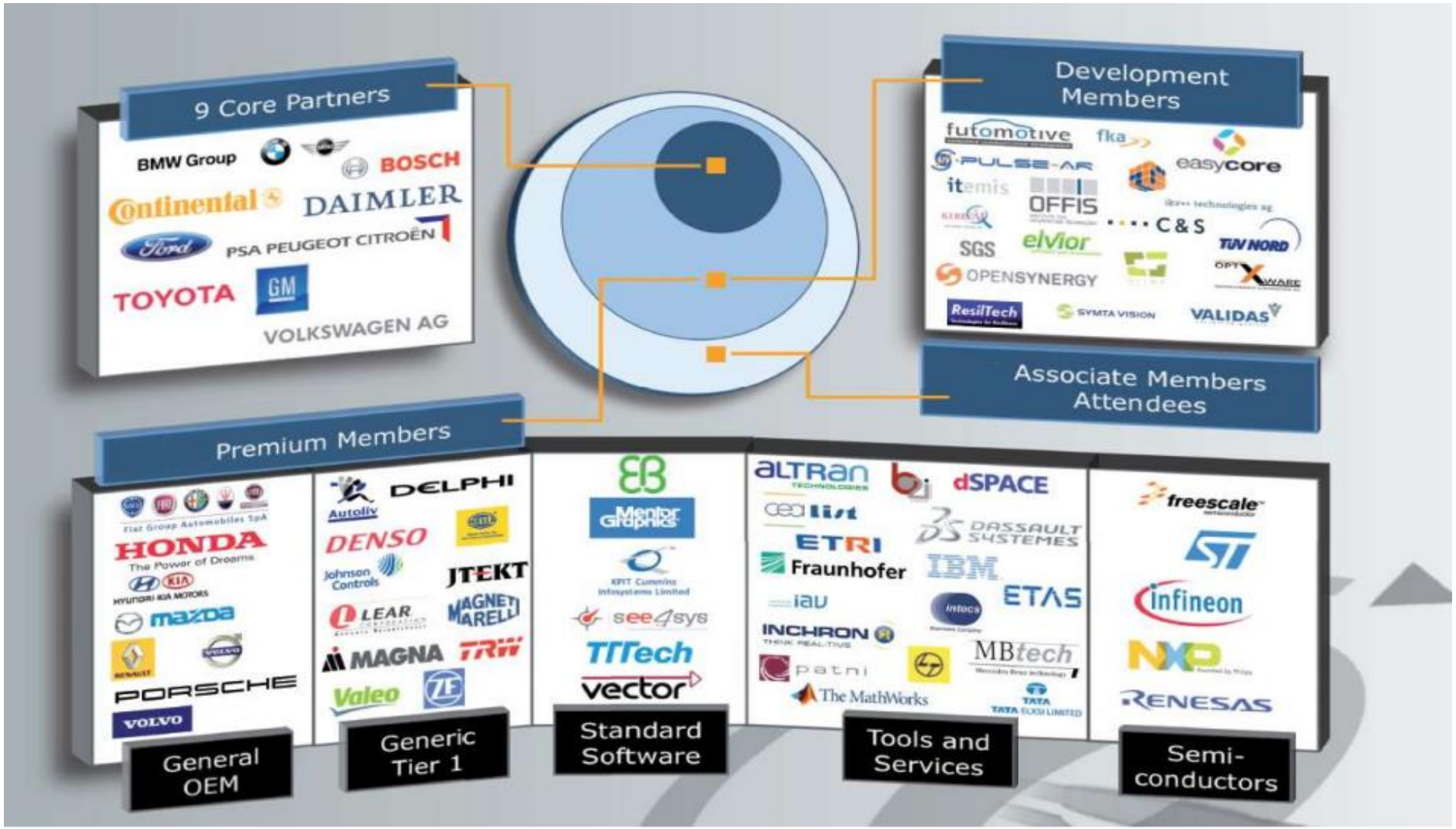
AUTOSAR是啥?

AUTOSAR (AUTomotive Open System ARchitecture):汽车开放系统架构的简称

AUTOSAR是领先的汽车制造商和供应商的标准化计划。
AUTOSAR联盟成立于2003年，现有9个核心成员单位，57个高级成员及上百个开发成员/参与者。联盟中的各个成员保持着汽车业内的开发合作关系。

AUTOSAR官网链接：https://www.autosar.org

AUTOSAR 伙伴/成员



伙伴类别 2018.11	数量
核心伙伴	9
高级伙伴	57
开发伙伴	49
普通伙伴	132
参与者	22

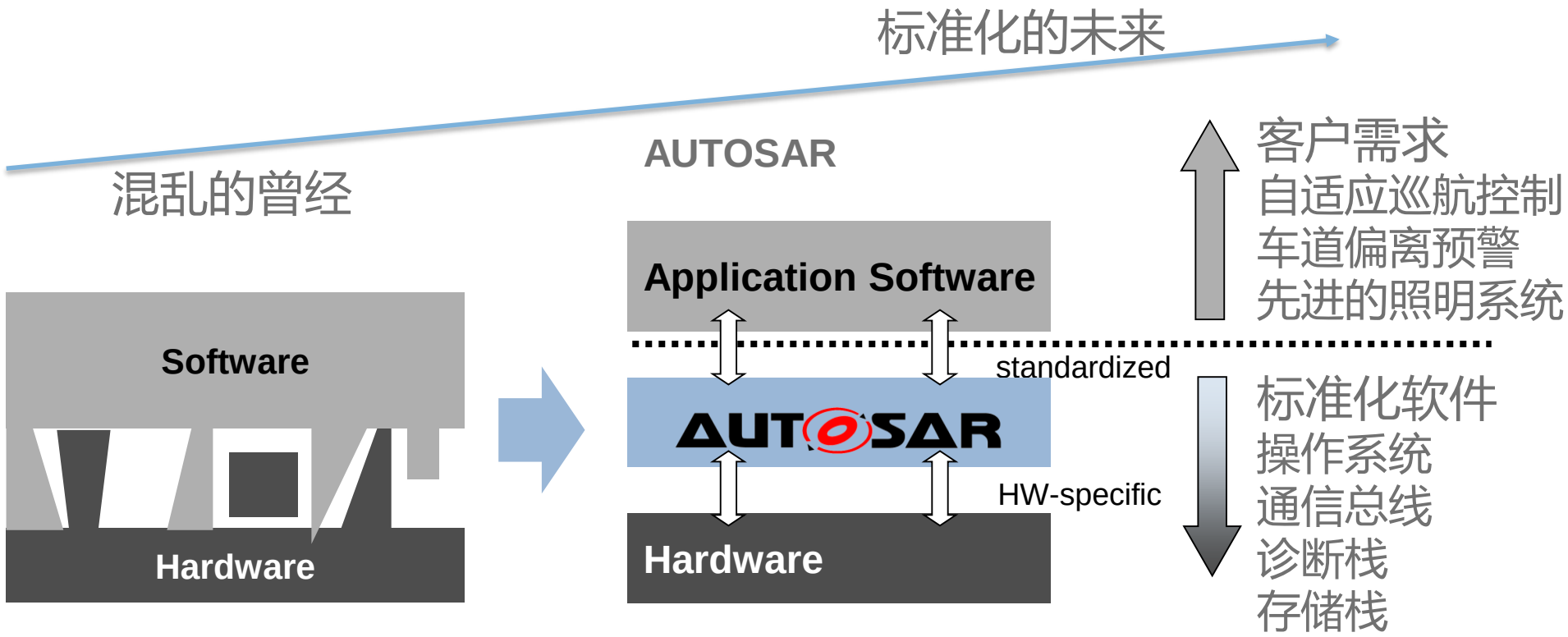
AUTOSAR 干什么?

随着复杂功能的增加，车载电子设备的发展范围越来越广，且越来越复杂。
AUTOSAR的主要想法是避免不断重复开发相同或类似的软件模块。
而全球开发合作伙伴关系的模式是为了汽车行业的系统架构建立开放标准。

为了减少开发工作量及提高质量，该系统平台采入统一且独立于生产者的方法，将硬件与软件彼此分离，各功能实现模块化且定义好统一接口以达到目标。

所以AUTOSAR联盟致力于为汽车电子控制装置开发一个开放的、标准化的软件架构。其目标包括不同车款和平台的延展性开发、软件迁移、有效性和安全需求的考量、各方合作关系的建立、自然资源的持续利用、整个“产品生命周期”的维护服务。

AUTOSAR愿景/目标

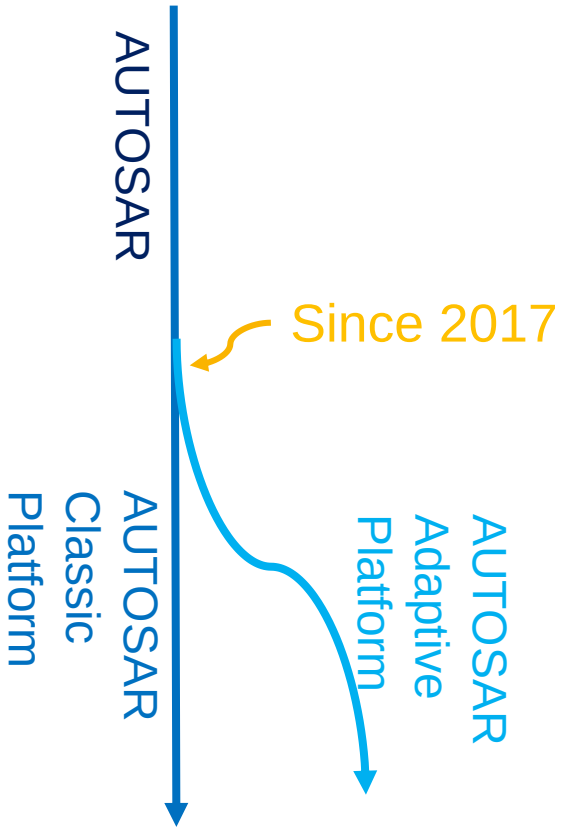


AUTOSAR分支简介

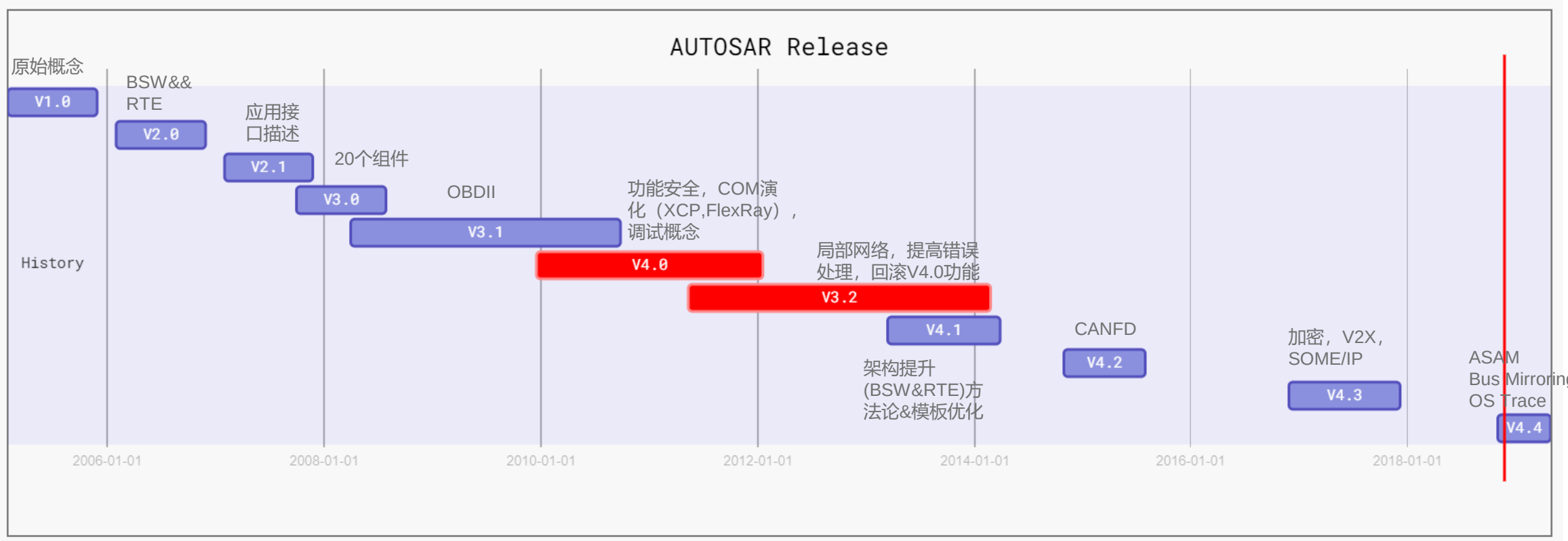
AUTOSAR联盟于2017年为其产品增加了一个新的标准。
叫AUTOSAR自适应平台（AUTOSAR Adaptive Platform）。
标准基于POSIX操作系统。

新平台的主要动机是满足自动驾驶，V2X应用以及不断增长的车辆外部网络链接需求。

以前的标准将继续支持。
命名为AUTOSAR经典平台（AUTOSAR Classic Platform）。
本文只介绍经典AUTOSAR，如无特别说明，后文出现的AUTOSAR默认指经典AUTOSAR。



经典AUTOSAR平台版本发布



AUTOSAR 常用术语/常用缩写

SWC/SW-C(Software Component): 软件组件

BSW(Basic Software Module): 基础软件模块

MCAL(Microcontroller Abstraction Layer): 微处理器抽象层

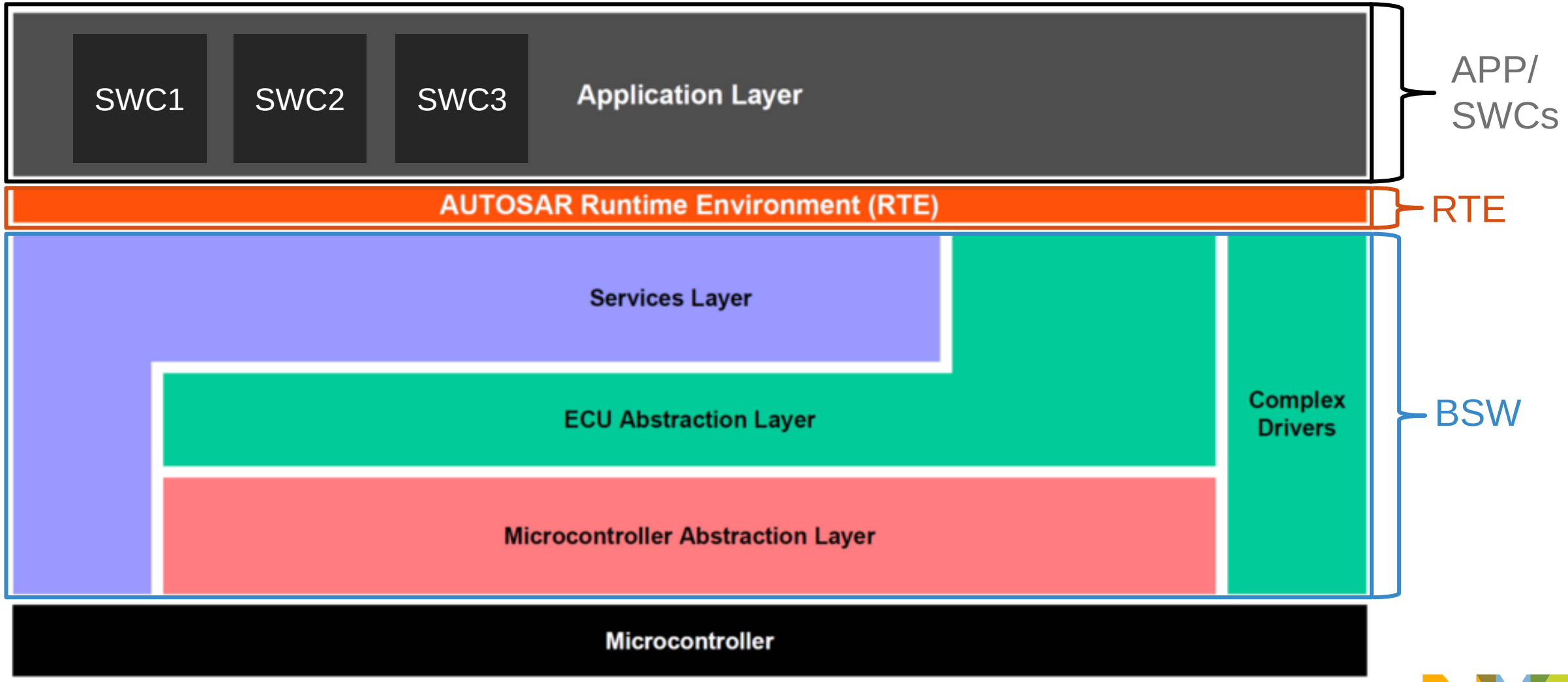
CDD(Complex Device Driver): 复杂设备驱动/复杂驱动

RTE(Run-Time Environment): 运行时环境

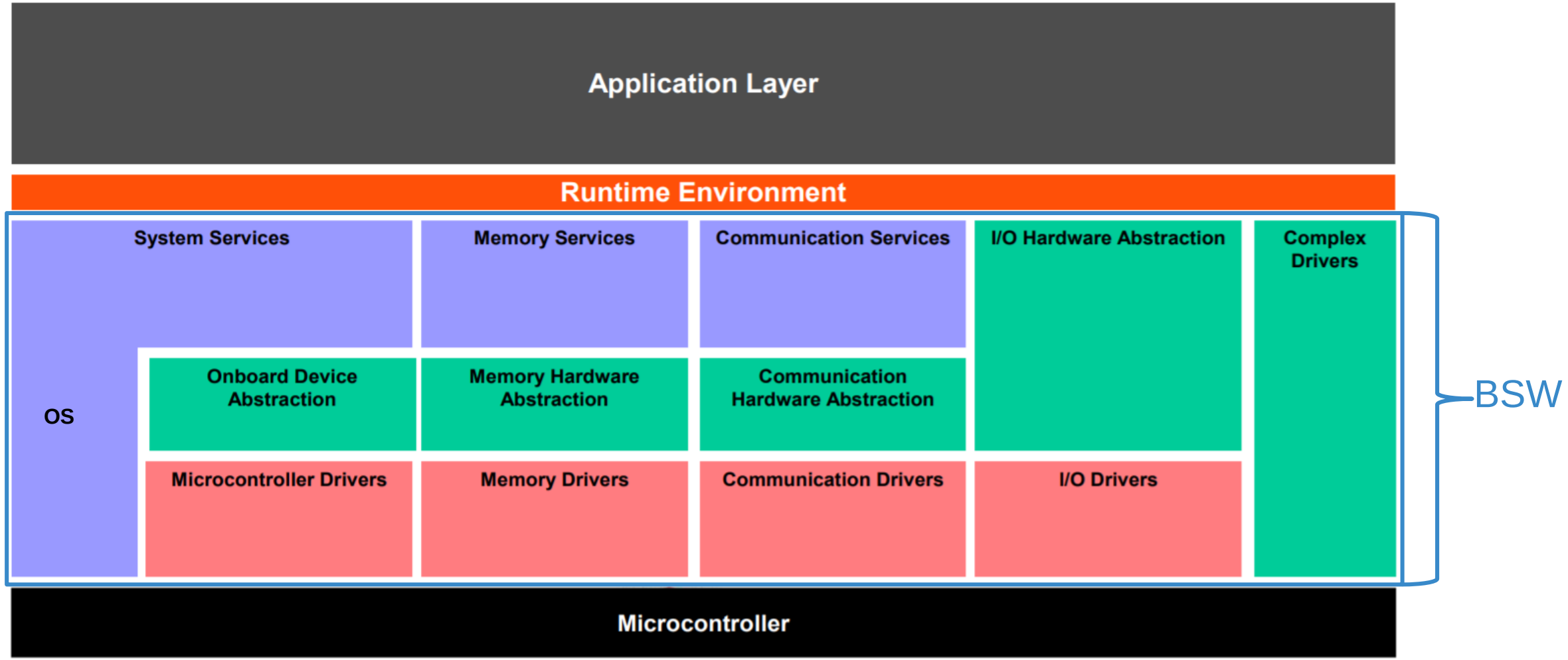
VFB(Virtual Function Bus): 虚拟功能总线

AUTOSAR
思想

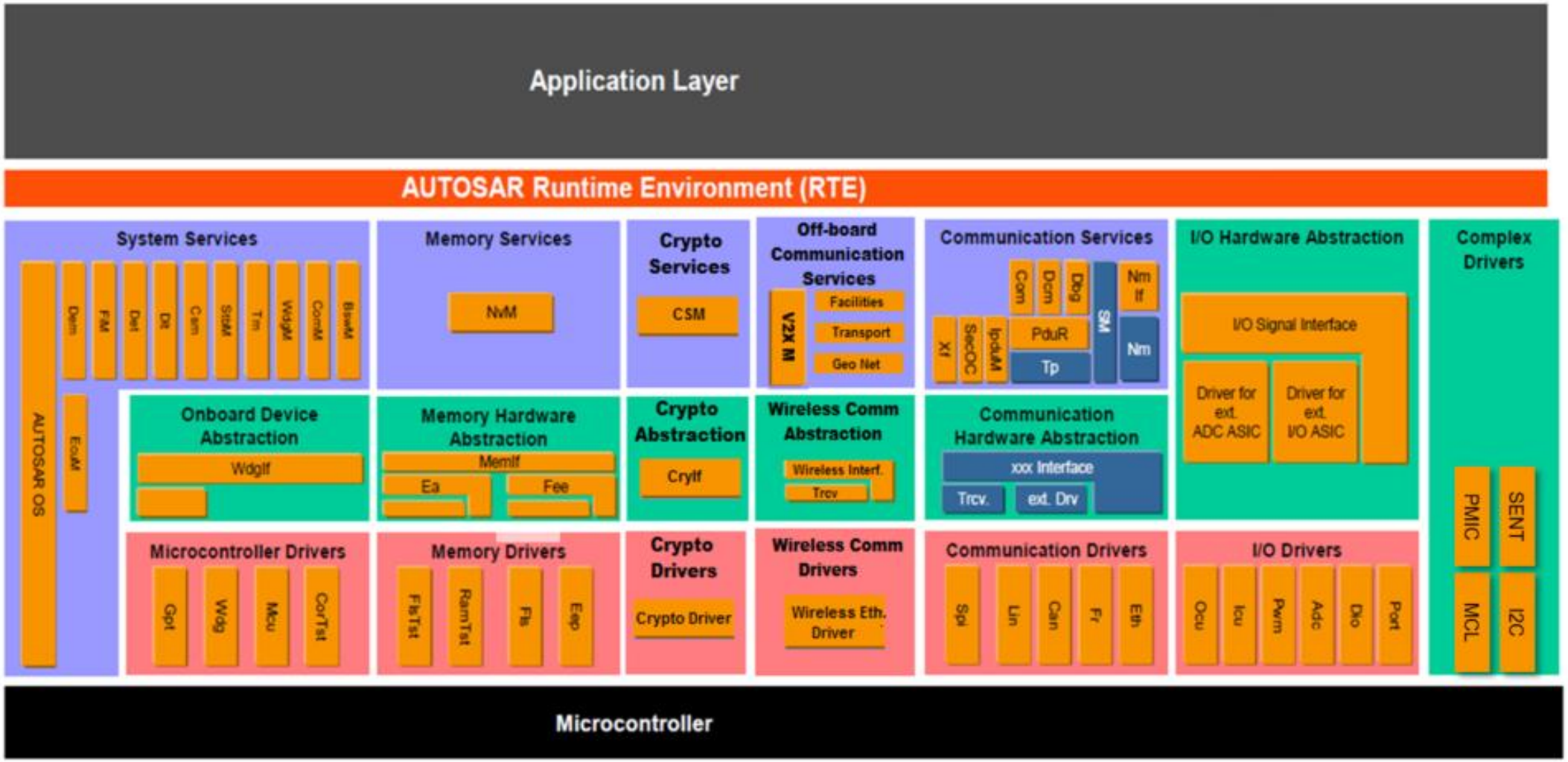
AUTOSAR思想 --- 分层



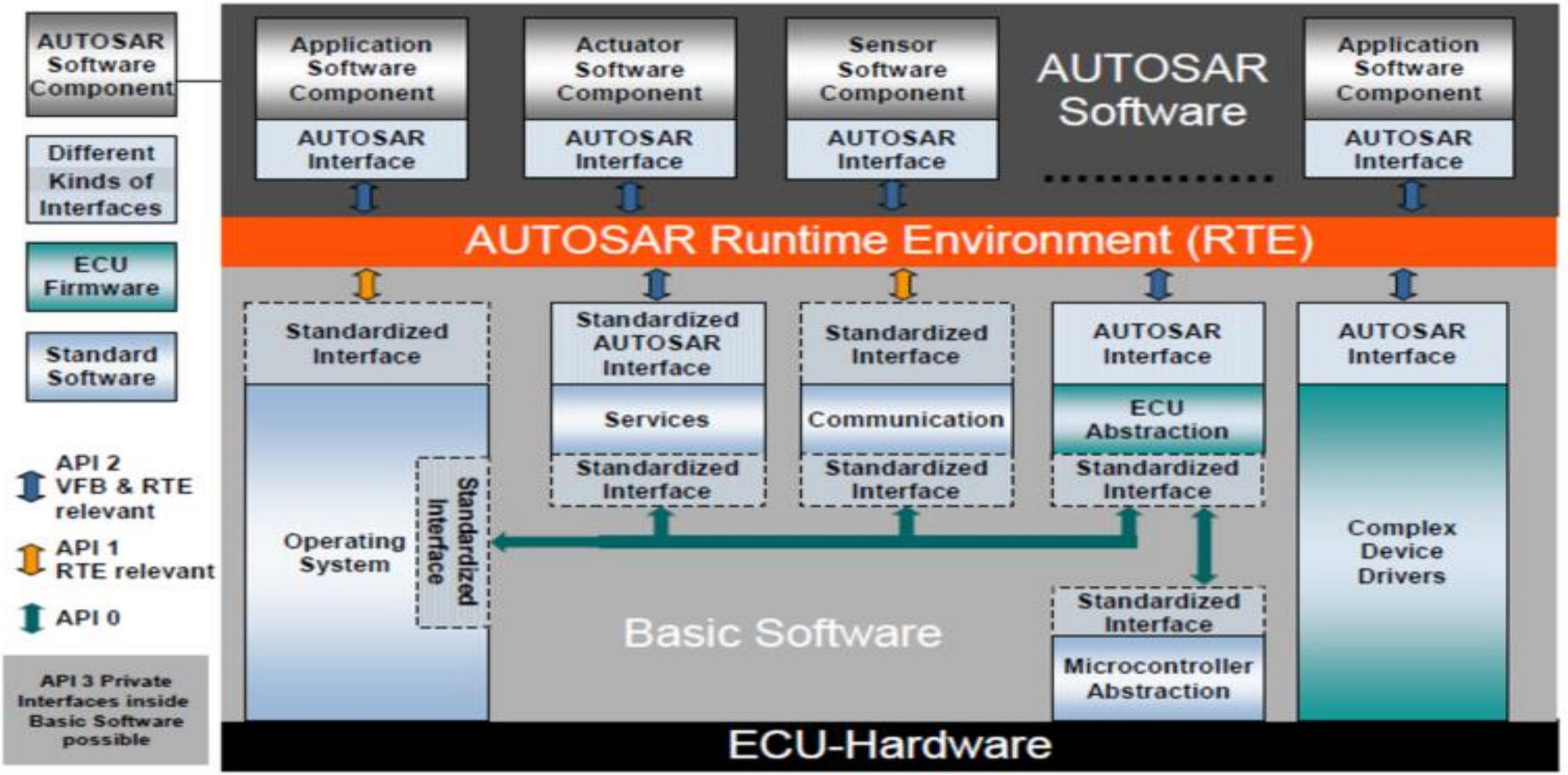
AUTOSAR思想 --- 模块化



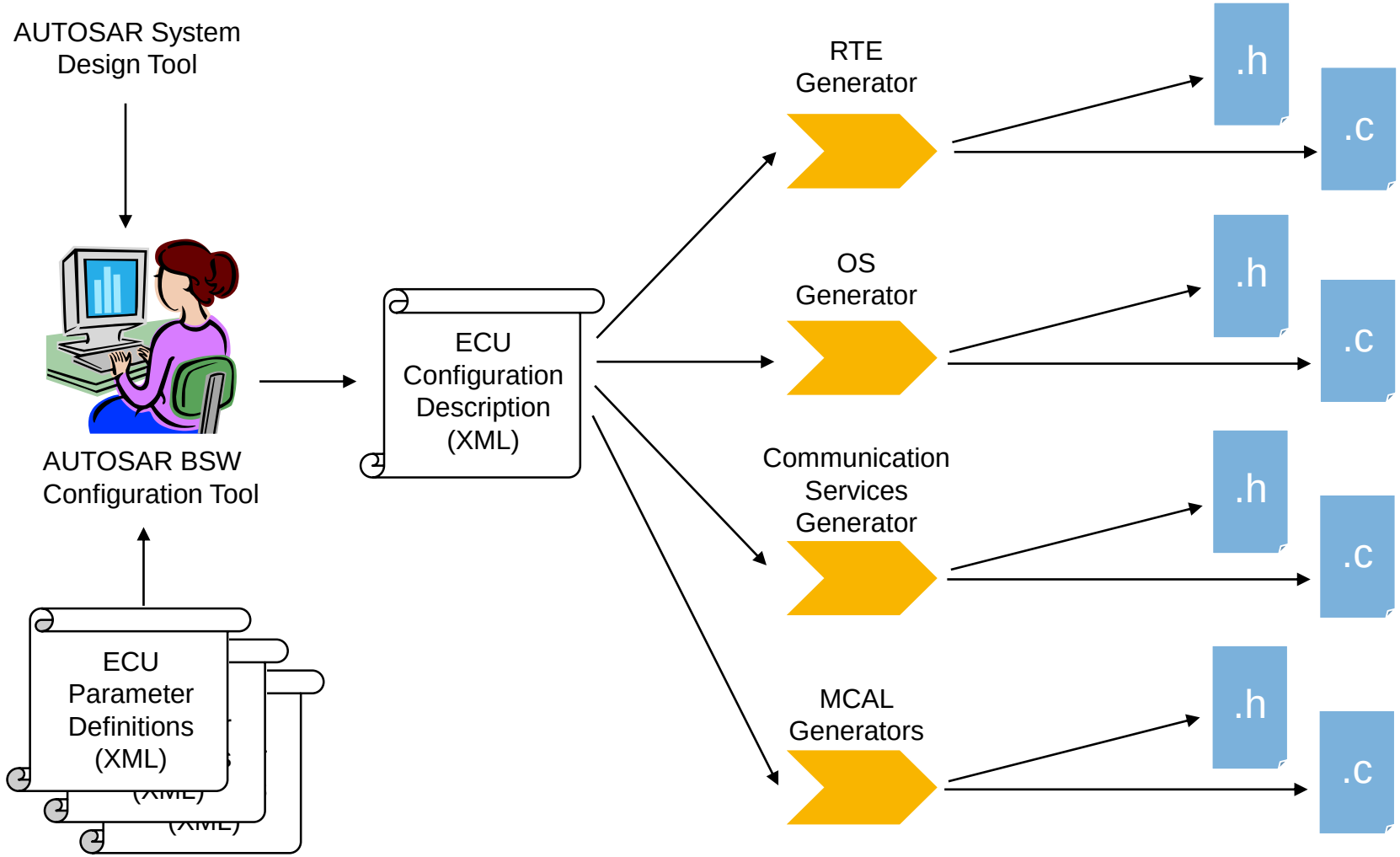
AUTOSAR思想 --- 模块化



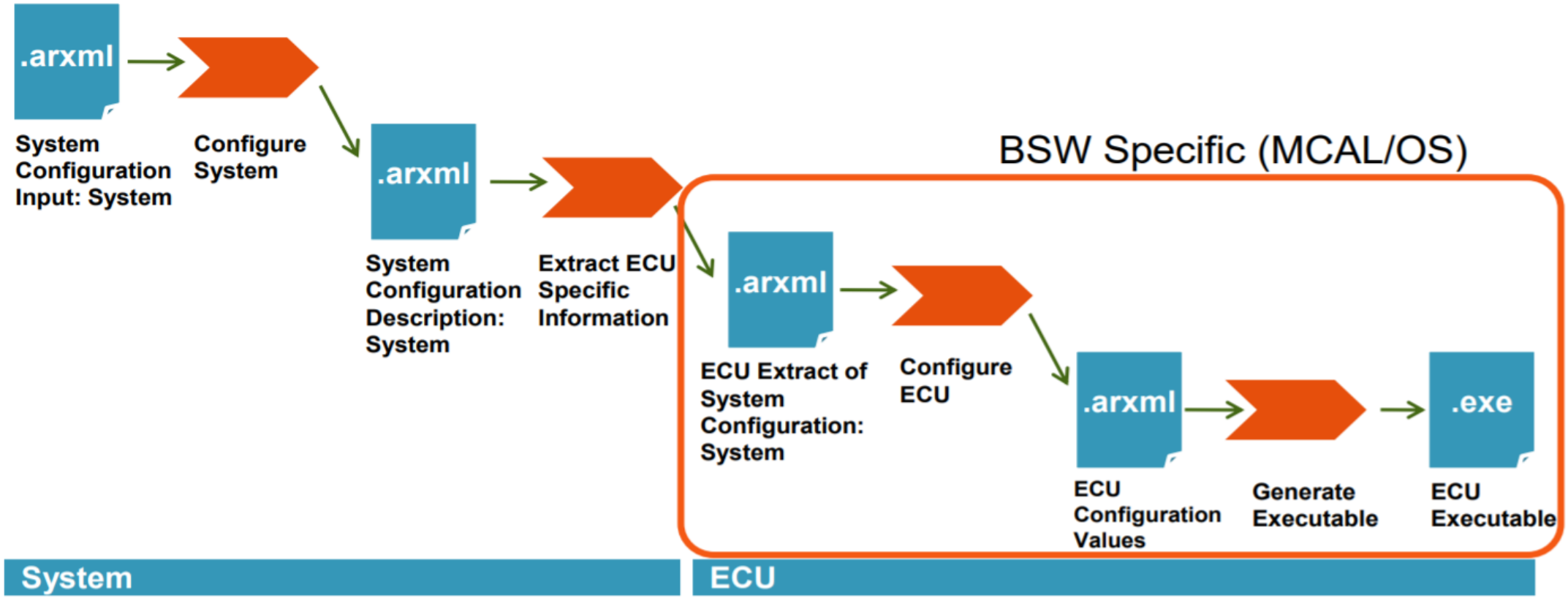
AUTOSAR思想 --- 接口标准化



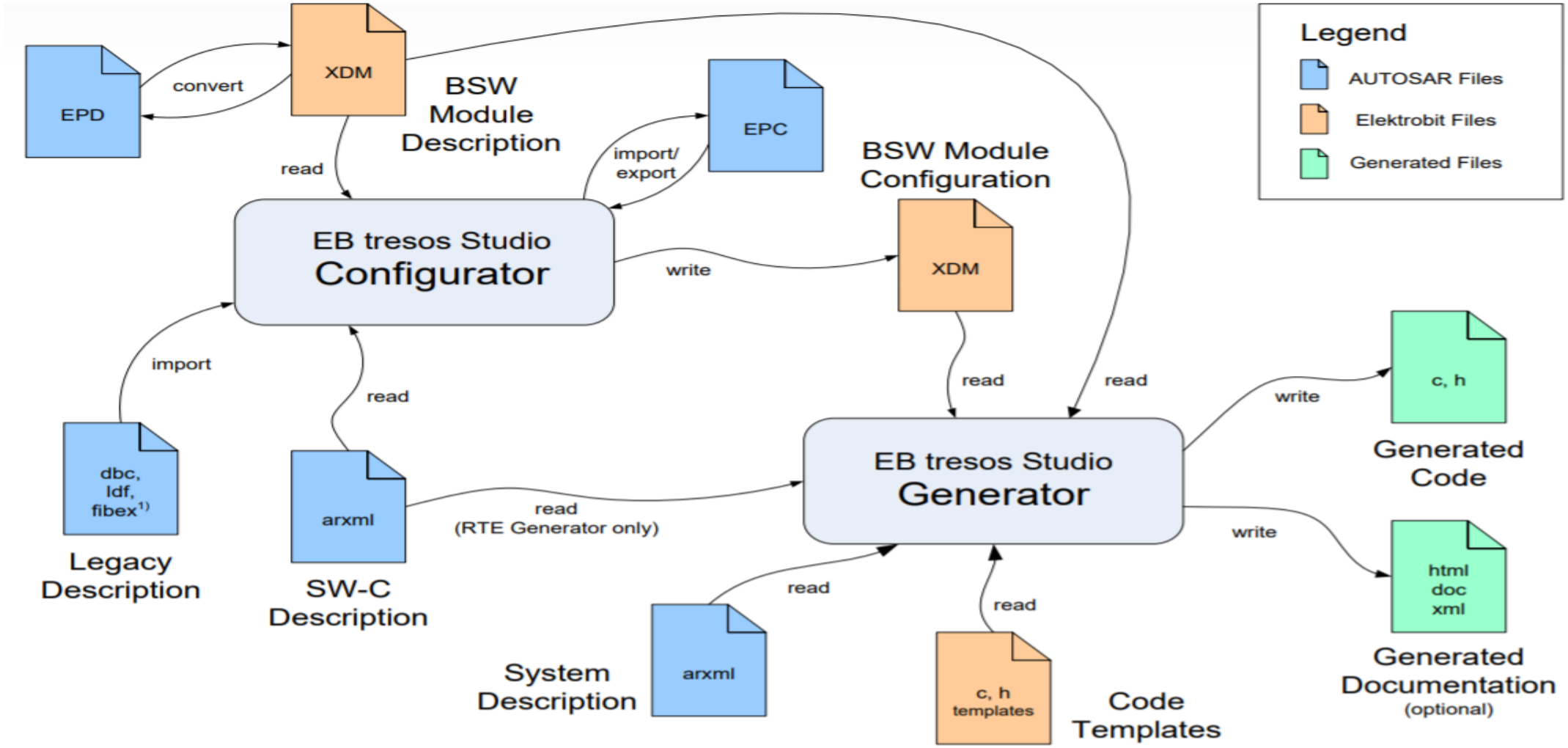
AUTOSAR思想 --- 工具标准化配置



AUTOSAR思想 --- 配置流程



AUTOSAR思想 --- 代码自动生成



AUTOSAR思想 --- 其他重要机制

VFB(Virtual Functional Bus): 虚拟总线
在系统设计阶段，我们可以使用虚拟总线将各个软件组件集成在一起。

SWC接口/端口链接:
每个SWC的接口都被定义好。
接口分为两大类： Sender/Reciever 和Client/Server
Sender/Reciever：端口用于指定软件组件的输入或输出数据。
Client/Server：端口用于指定软件组件提供或需要的服务。

AUTOSAR思想 --- 参数配置分类

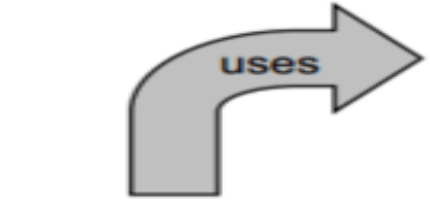
Pre-compile: 预编译配置
- 这些配置参数在编译后不能改变。

比如：代码里面的预编译宏指令，或者你映射到某个信号的端口等等。

Link-time: 链接时配置（用于产生目标文件给集成者）
- 配置参数由链接脚本决定。链接完成后配置参数不能改变。

Post-build: 编译后（用于一个软件适配不同车型）
- 编译后，不用重新完整刷写ECU，配置参数也可以被改变。
- 在系统启动阶段，配置参数集可以根据需要在多个配置里面进行选择，但是所有可能的配置需要在编译阶段包含进去。
- 这些配置参数存放在一个确定的内存空间。
- 编译后参数也可能包含预编译参数及链接时参数

AUTOSAR思想 --- 调用规则



uses	System Services / OS	Memory Services	Crypto Services	Communication Services	Off-board Comm. Services	Complex Drivers	I/O Hardware Abstraction	Onboard Device Abstr.	Memory HW Abstraction	Crypto HW Abstraction	Comm. HW Abstraction*	Microcontroller Drivers	Memory Drivers	Crypto Drivers	Communication Drivers*	I/O Drivers	Microcontroller Hardware
SW Components / RTE	✓	✓	✓	✓	✓	✓	✓	×	×	×	×	×	×	×	×	×	×
System Services / OS	✓	✓	✓	✓	✓	Δ	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Memory Services	✓	✓	✓	×	×	Δ	×	×	✓	×	×	×	×	×	×	×	×
Crypto Services	✓	✓	✓	×	×	Δ	×	×	×	✓	×	×	×	×	×	×	×
Communication Services	✓	✓	✓	✓	✓	Δ	×	×	×	×	✓	×	×	×	×	×	×
Off-board Comm. Services	✓	✓	✓	✓	✓	Δ	×	×	×	×	✓	×	×	×	×	×	×
Complex Drivers	restricted access -> see the following two slides																
I/O Hardware Abstraction	✓	×	✓	×	×	×	✓	✓	×	×	✓	✓	×	×	✓	✓	×
Onboard Device Abstr.	✓	×	×	×	×	×	×	✓	×	×	✓	✓	×	×	✓	✓	×
Memory HW Abstraction	✓	✓	×	×	×	×	×	✓	✓	×	✓	×	✓	×	✓	×	×
Crypto HW Abstraction	✓	✓	✓	×	×	×	×	✓	×	✓	×	×	×	×	×	✓	×
Comm. HW Abstraction*	✓	×	×	✓	✓	×	×	✓	×	×	✓	×	×	×	✓	✓	×
Microcontroller Drivers	✓	×	×	×	×	×	Δ	Δ	×	Δ	×	Δ	×	×	×	Δ	✓
Memory Drivers	✓	×	×	×	×	×	×	×	Δ	×	×	×	×	×	×	×	✓
Crypto Drivers	✓	×	×	×	×	×	×	×	×	Δ	×	×	×	×	×	×	✓
Communication Drivers*	✓	×	×	×	×	×	×	Δ	×	×	Δ	×	×	×	×	✓	✓
I/O Drivers	✓	×	×	×	×	×	Δ	Δ	×	Δ	×	Δ	×	×	×	Δ	✓

*: includes wired and wireless communication

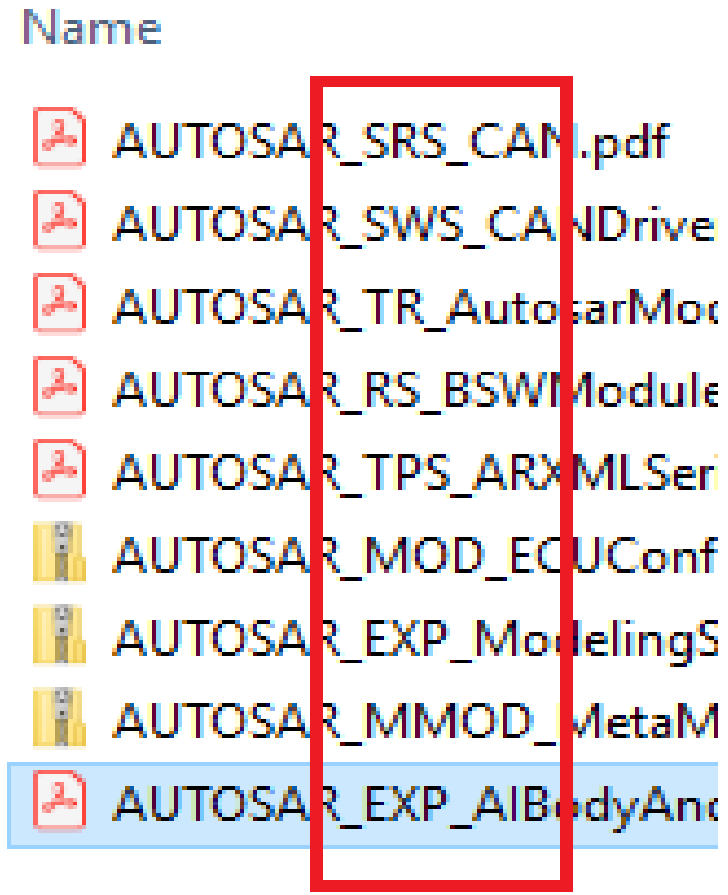
√： 可以调用
×： 不可调用
Δ： 只能使用回掉函数

AUTOSAR
文档

AUTOSAR 文档分类

目前联盟将文档细分为九类：

- TMPL(Template): 模板
- SWS(Software Specification): 软件规范
- SRS(Software Requirement Specification): 软件需求规范
- RS(Requirements Specification): 需求规范
- TR(Technical Report): 技术报告
- EXP(Explanatory Documents): 解释性文件
- TPS(Template Specification): 模板规范
- MMOD(Meta Model): 元模型
- MOD(Model): 模型



AUTOSAR软件开发重要文档怎么阅读 --- SRS

SRS文档为需求描述文档。BSW文档为技术规格文档，其必须完全覆盖SRS文档所描述的需求内容。
AUTOSAR SRS文档主要章节内容概览：

第一章：定义相应基础软件模块的应用范围

第二章：定义文档结构

第三章：定义本文档使用的缩略词

第四章：本文档主要内容章节。包含该基础软件模块功能描述和详细需求。

AUTOSAR软件开发重要文档怎么阅读 --- SWS

本文档包含了相应模块的详细技术实现描述，其章节结构如下：

第一章：介绍相应模块的功能描述

第二章：定义本文档使用的缩略词

第三章：参考文档/依赖文档

第四章：定义汽车领域的限制和适用性

第五章：本模块的文件结构及与其他模块的依赖关系

第六章：需求矩阵表，与SRS的对应关系。

第七章：本模块的详细功能描述

第八章：API描述

第九章：与相关模块交互的时序图

第十章：配置参数描述

AUTOSAR
代码

AUTOSAR代码 --- 类型定义

AUTOSAR标准类型头文件：
Std_Types.h。

这里面定义了一些与硬件平台及编译器无关的通用类型，各模块都能使用。

- E_OK, E_NOT_OK
- STD_HIGH, STD_LOW
- STD_ACTIVE, STD_IDLE
- STD_ON, STD_OFF
- Std_ReturnType (E_OK, E_NOT_OK)
- Std_VersionInfoType

AUTOSAR平台类型头文件：
Platform_Types.h。

这里面定义了与硬件平台及编译器有关的一些类型及符号。

- boolean
- uint8, uint16, uint32
- sint8, sint16, sint32
- uint8_least, uint16_least, uint32_least
- sint8_least, sint16_least, sint32_least
- float32, float64

AUTOSAR代码 --- 编译抽象

- AUTOSAR规定了一些宏定义用于编译器抽象，这些宏用于变量及代码的声明和定义处。
- 编译器抽象适用于各个AUTOSAR软件模块。
- 这些定义放在Compiler.h, Compiler_Cfg.h文件里面。
- 对于不同的硬件平台及编译器，其抽象参数不一样，需要单独提供。
- Compiler.h文件里面提供以下宏定义（但不限与以下内容）：
  - FUNC
  - P2VAR
  - P2CONST
  - CONSTP2VAR
  - CONSTP2CONST
  - P2FUNC
  - CONST
  - VAR

AUTOSAR代码 --- 编译抽象

每个软件模块应该至少支持以下不同（存储）属性类型定义。这些定义放在Compiler_Cfg.h文件里面：

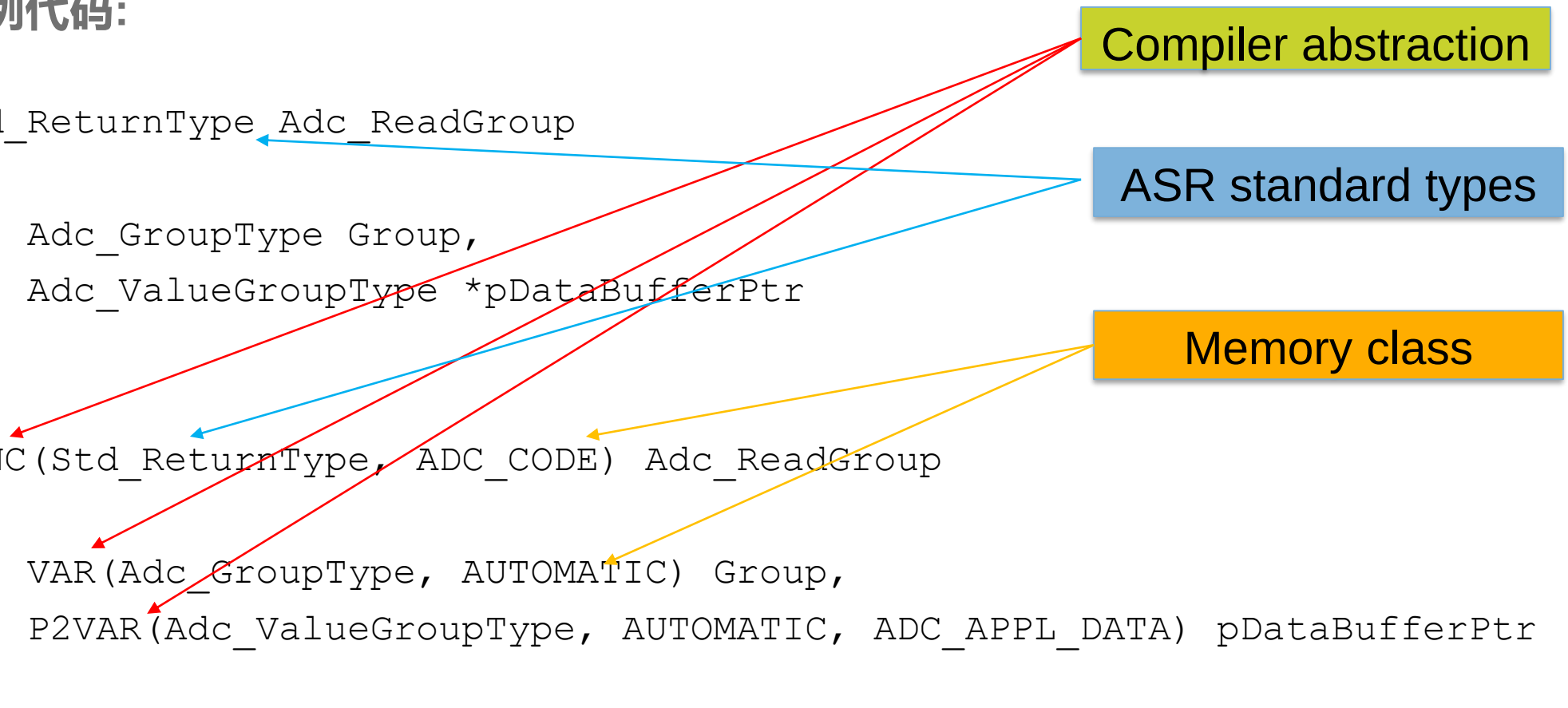
<MSN>_CODE
<MSN>_CONST
<MSN>_APPL_DATA
<MSN>_APPL_CONST
<MSN>_APPL_CODE
<MSN>_VAR_NOINIT
<MSN>_VAR_POWER_ON_INIT
<MSN>_VAR_FAST
<MSN>_VAR

AUTOSAR代码 --- 示例

示例代码:

```
Std_ReturnType Adc_ReadGroup
(
    Adc_GroupType Group,
    Adc_ValueGroupType *pDataBufferPtr
);

FUNC(Std_ReturnType, ADC_CODE) Adc_ReadGroup
(
    VAR(Adc_GroupType, AUTOMATIC) Group,
    P2VAR(Adc_ValueGroupType, AUTOMATIC, ADC_APPL_DATA) pDataBufferPtr
);
```



AUTOSAR代码 --- 存储映射

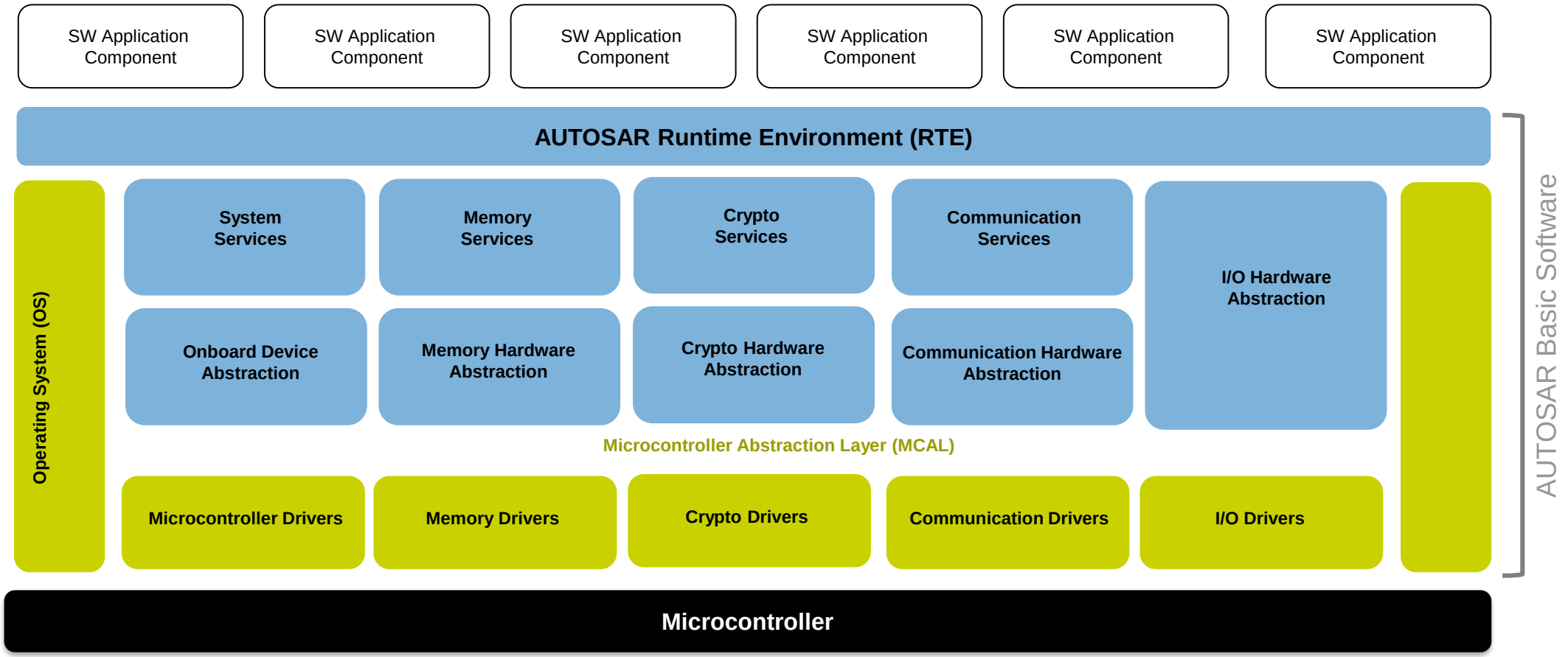
- AUTOSAR定义了将代码和数据与指定内存区域映射的机制。
- 内存映射适用于每个AUTOSAR软件模块。
- 这些映射关系在内存映射文件里面进行描述： MemMap.h (ASR4.0.3) <MDL>_MemMap.h(ASR4.2 and ASR 4.3)
- 内存映射文件不负责描述具体物理地址/内存段地址。具体地址信息由链接脚本定义。
- 内存映射通过编译器指令的方法来实现。(比如：#pragma 或 __attribute())

- 下面为一个BSW模块的示例: 变量 EepTimer 被放到 EEP_START_SEC_VAR_16BIT 内存区域。

```
#define EEP_START_SEC_VAR_16BIT
#include “MemMap.h”
static uint16 EepTimer;
static uint16 EepRemainingBytes;
#define EEP_STOP_SEC_VAR_16BIT
#include “MemMap.h”
```

NXP AUTOSAR

NXP AUTOSAR软件



NXP开发

第三方开发

NXP AUTOSAR: 服务矩阵

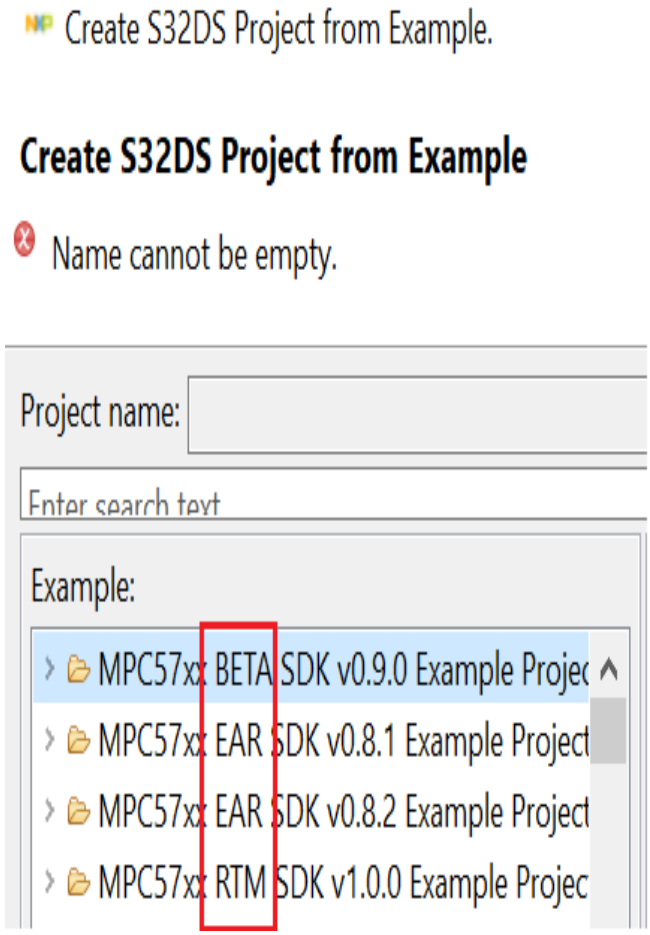
available, commercially licensed | in development / planned | not planned

	QM	ISO26262 ASIL
ASR 4.3		S32R ARM, S32S ARM, S32K2x ARM, MPC574xC-G, S32V ARM, S32G ARM, S32K1xx ARM
ASR 4.2	MPC574xC-G, S32R274, S32V234 ARM, S32R372, S32K1xx ARM	S32R274, S32V234 ARM, S32K2x ARM, S32R372, MPC574xC-G, S32K1xx ARM
ASR 4.0.3	MPC5777C, MPC5777M, MPC5746R, MPC5775K, MPC574xP, MPC574xC-G, S32K1xx ARM, S12ZVM256, S12ZVC, MAC57D5 ARM, SAC58R ARM, MPC560xB, MPC564xA, MPC560xP, MPC567xK, MPC564xB-C, MPC567xR, MPC564xL	MPC5777C, MPC5777M, MPC5746R, MPC5775K, MPC574xP, MPC574xC-G, S32K1xx ARM
ASR 3.2	MPC560xB, MPC564xB-C	
ASR 3.0	MPC564xB-C, MPC563xM, MPC560xB, MPC564xL, MPC5668G, MPC564xA, MPC567xK, S12, MPC560xS, MPC567xF, MPC560xP	
ASR 2.1	MPC551x, MPC560xB, MPC564xB-C	

For current status please contact NXP marketing

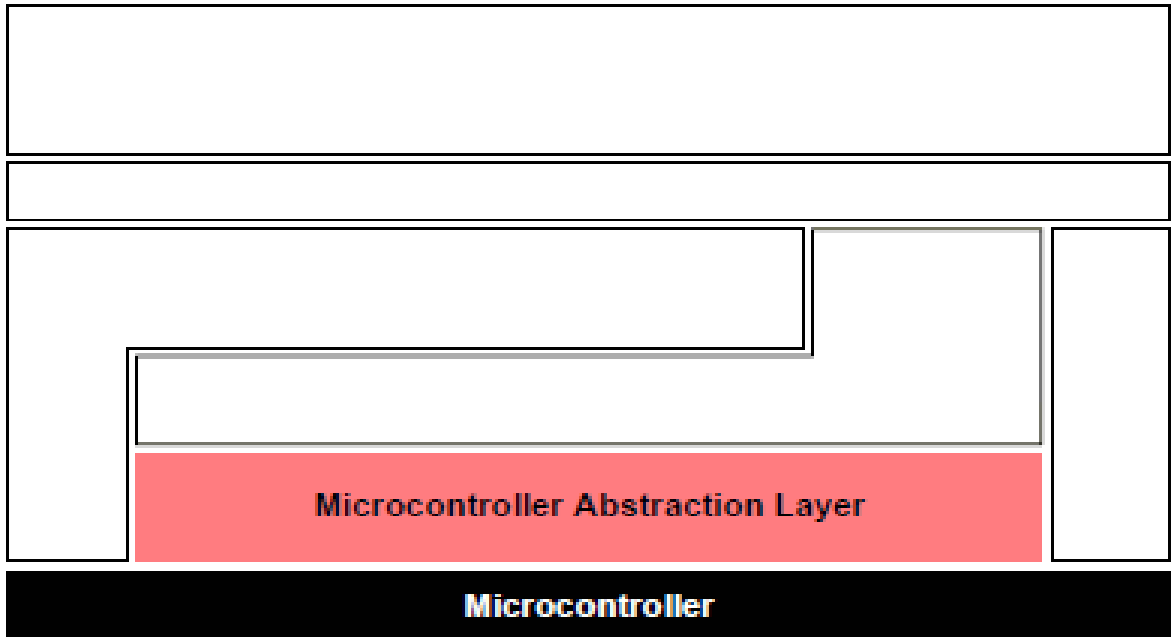
NXP AUTOSAR版本发布对比

		EAR	Beta	RTM-C
产品模块		只有部分MCAL驱动	包含所有MCAL驱动	包含所有MCAL驱动
文档	技术	只有部分用户文档	包含完整用户文档	包含完整用户文档
	质量校对	无校对	完整校对过的文档	完整验校对过的文档
测试	使用的硬件	第一版样件	未验证的样件	经过验证的样件
	测试覆盖率	有限的测试	100%验证通过	100%验证通过
	分支测试覆盖率	无	90%	100%
	扩展测试	无	集成测试	集成测试&EPD测试
功能		部分功能	100%	100%

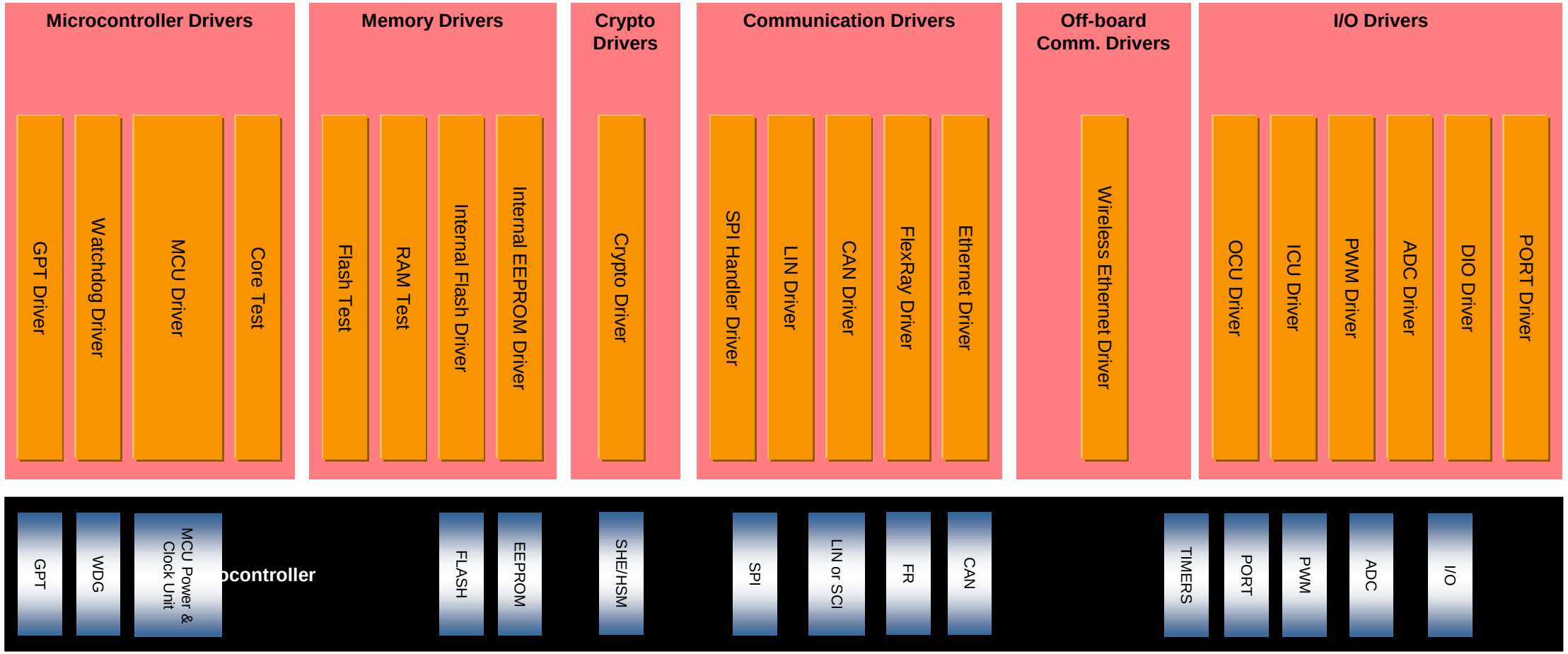


AUTOSAR MCAL简介

Microcontroller Abstraction Layer(MCAL)位于基础软件（BSW）里面最底层。
它提供各种驱动程序，其他模块通过它达到对硬件访问的目的。
这样也达到了将上层模块与底层硬件隔离的目的。



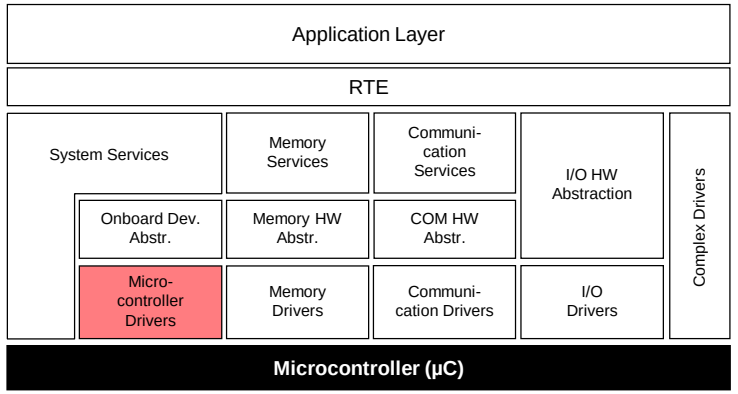
AUTOSAR MCAL组成

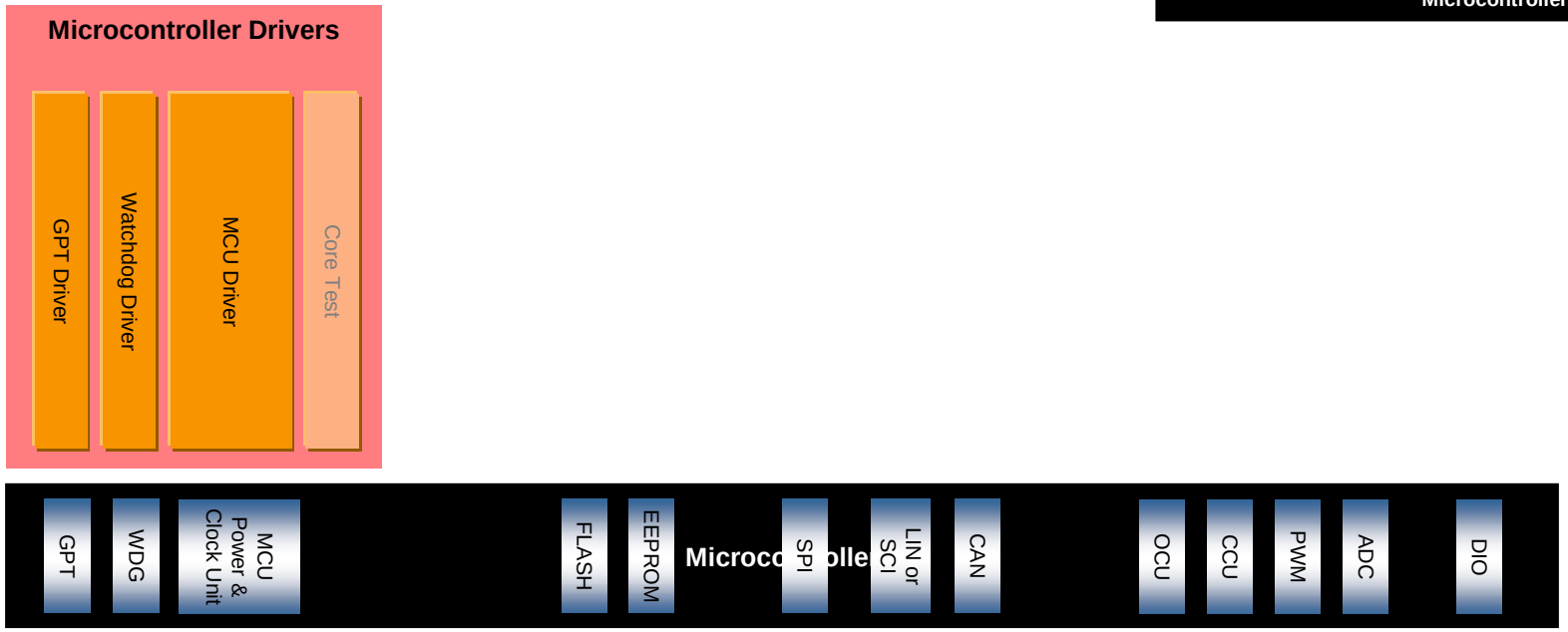


AUTOSAR MCAL --- 微控制器驱动

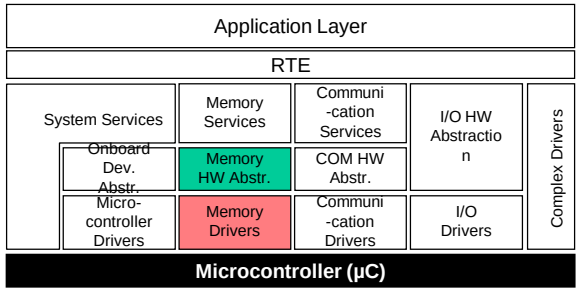
微控制器驱动

- 提供内部设备驱动（比如：看门狗，通用计时器）
- 提供对硬件直接访问的功能



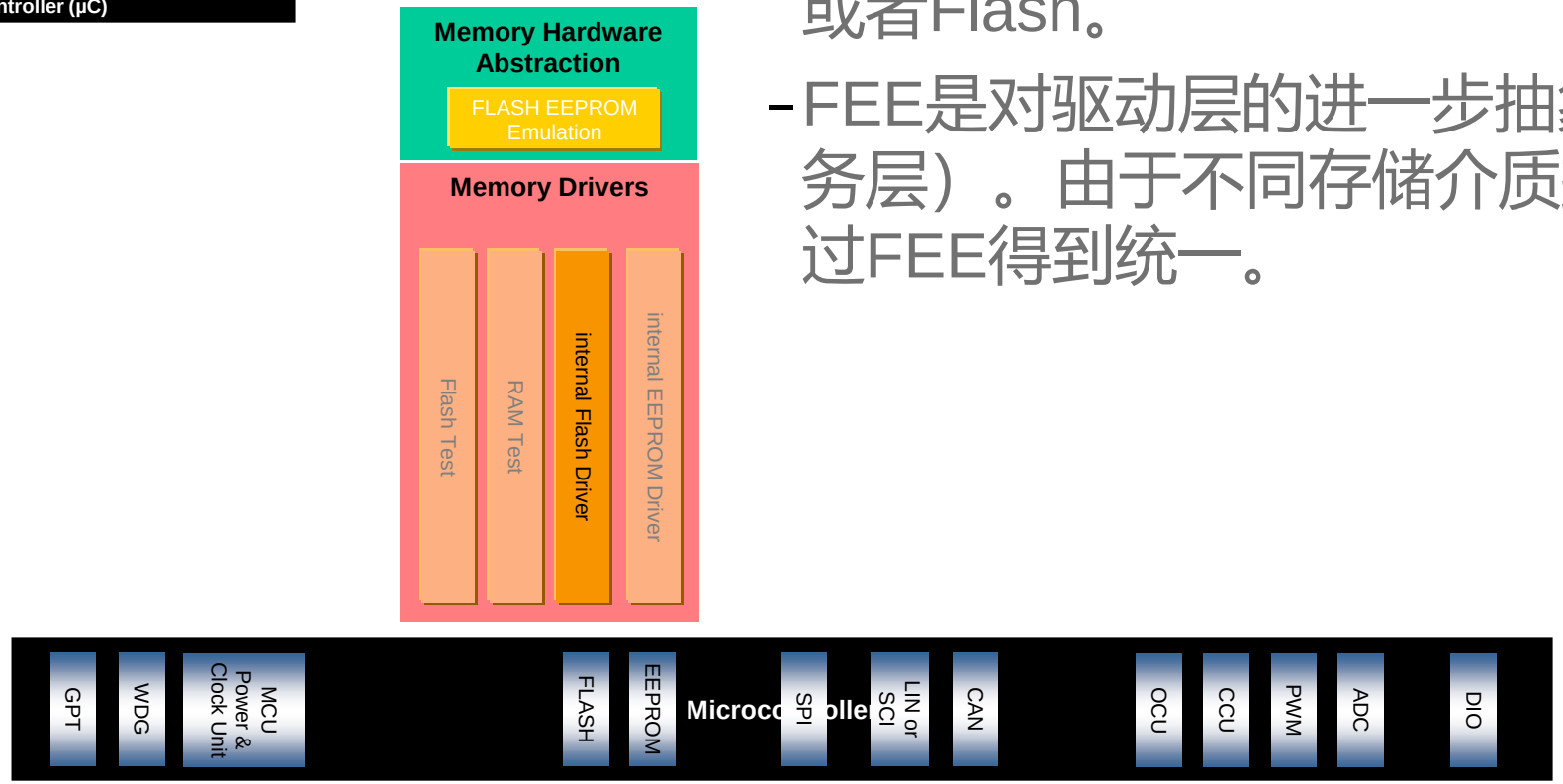


AUTOSAR MCAL --- 存储模块驱动



存储驱动

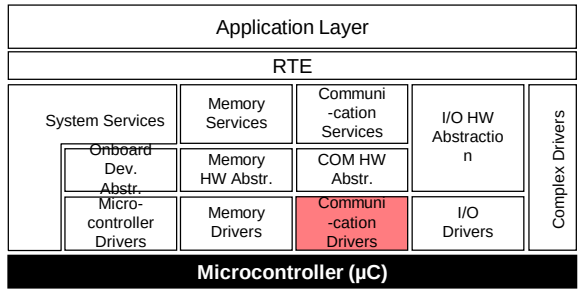
- 驱动模块提供对存储设备（片内或板级）的操作能力。比如：片内EEPROM或者外部EEPROM或者Flash。
- FEE是对驱动层的进一步抽象（属于ECU抽象服务层）。由于不同存储介质操作方式不一样，通过FEE得到统一。

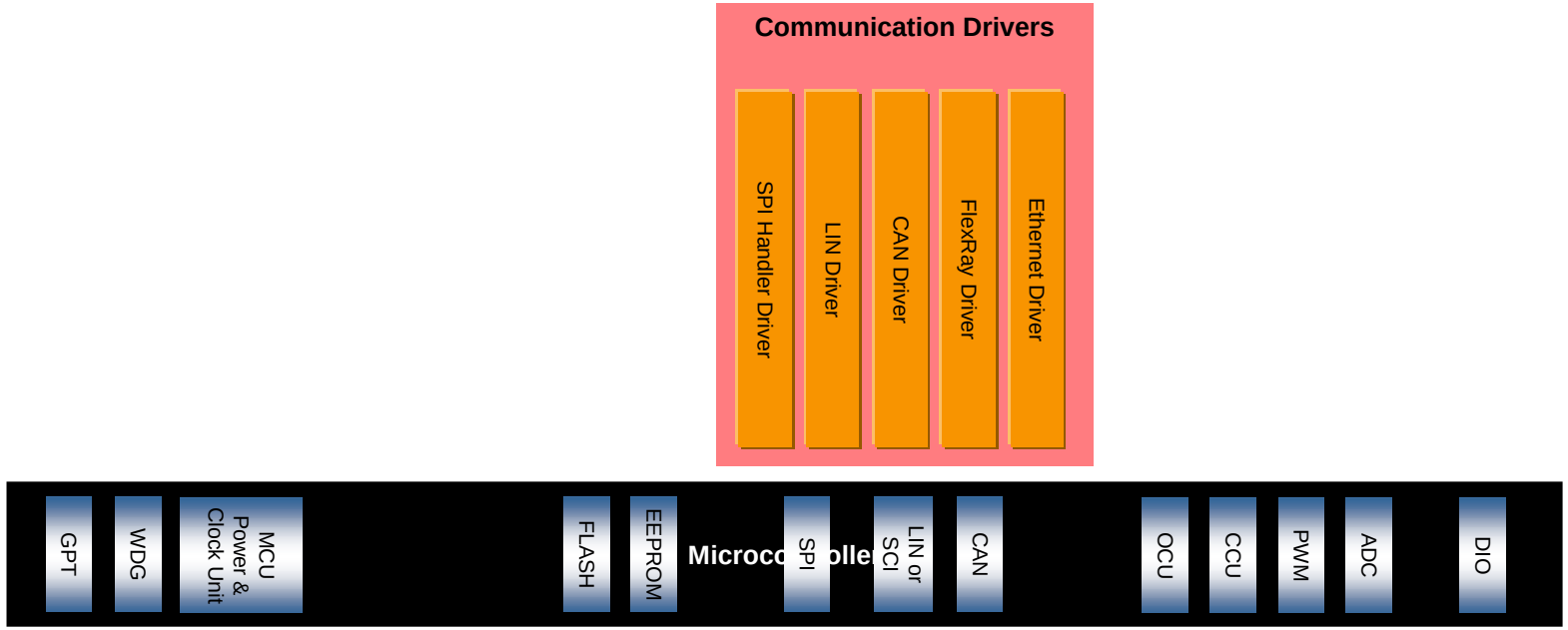


AUTOSAR MCAL --- 通信模块驱动

通信驱动

- 片上串行总线（SPI）及车身通信总线（CAN,LIN,FlexRay，Eth…）



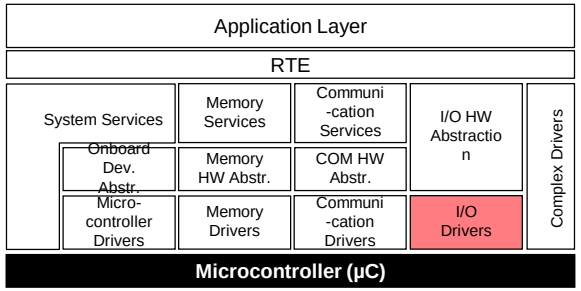


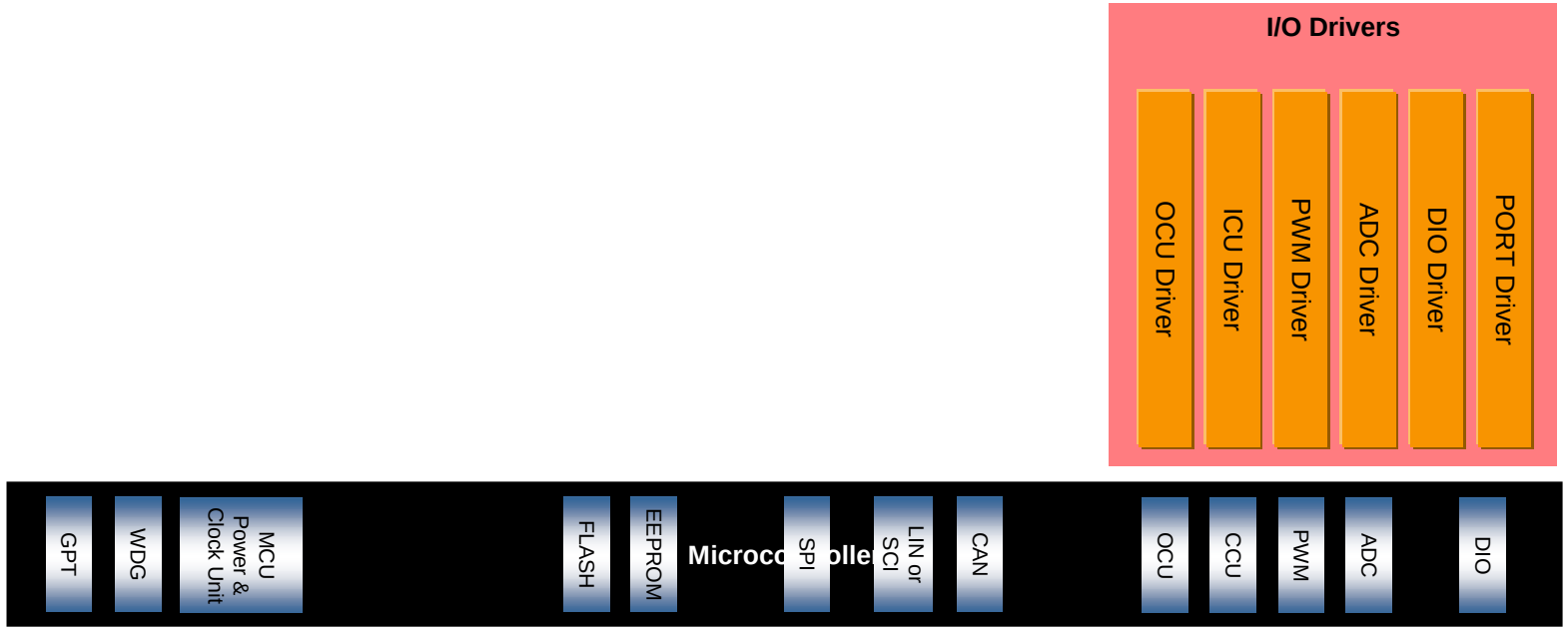
Source: AUTOSAR

AUTOSAR MCAL --- I/O端口驱动

I/O 驱动

- ADC, PWM, DIO,OCU,ICU,PORT驱动





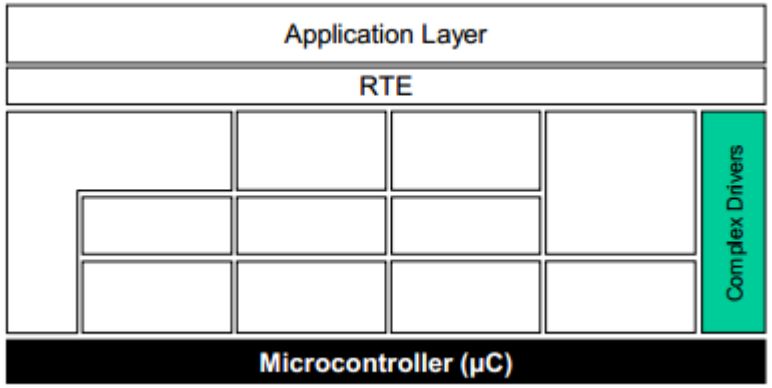
NXP 复杂驱动

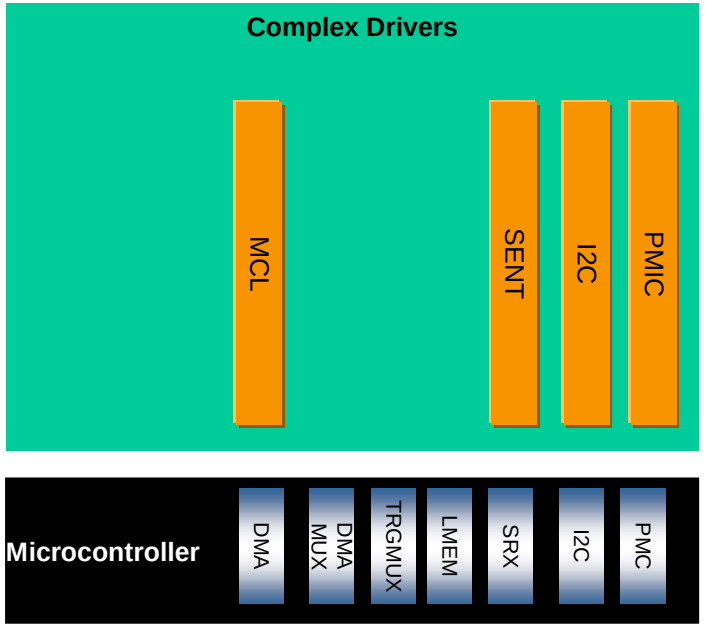
复杂驱动跨越了MCAL和ECU抽象层。

复杂驱动可以是但不仅限于驱动，可以是ECU抽象层，甚至可以是应用程序。

NXP MCAL支持：

- ✓ 未标准化通信模块：如 SENT，I2C。
- ✓ 电源管理芯片: PMIC
- ✓ 一些常用库（DMA，Cache等）。：MCL





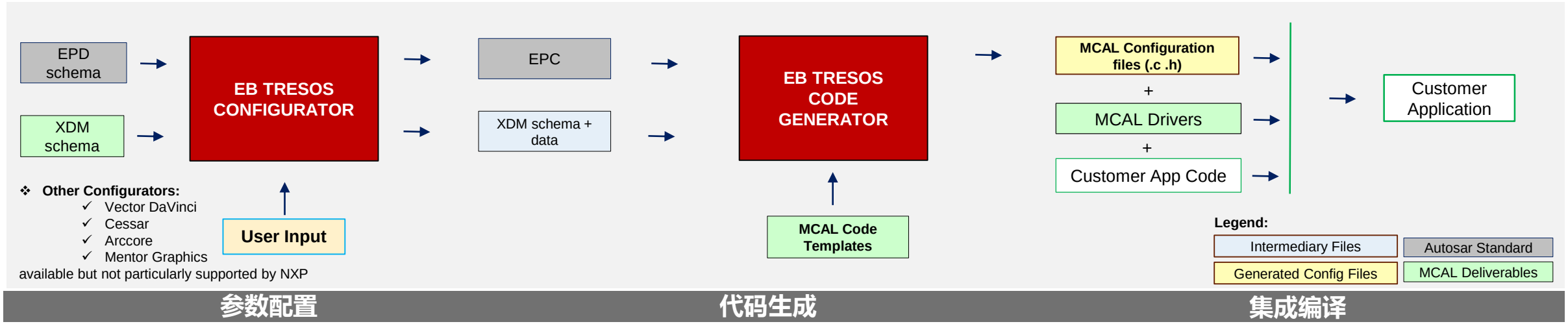
NXP AUTOSAR MCAL资源获取

可以根据你实际硬件平台到以下链接下载安装：
下载链接

安装步骤请参照以下文档：
“基于S32K14x系列的AUTOSAR平台快速引导.pdf”

使用的配置工具：
EB Tresos Studio

NXP AUTOSAR MCAL使用流程



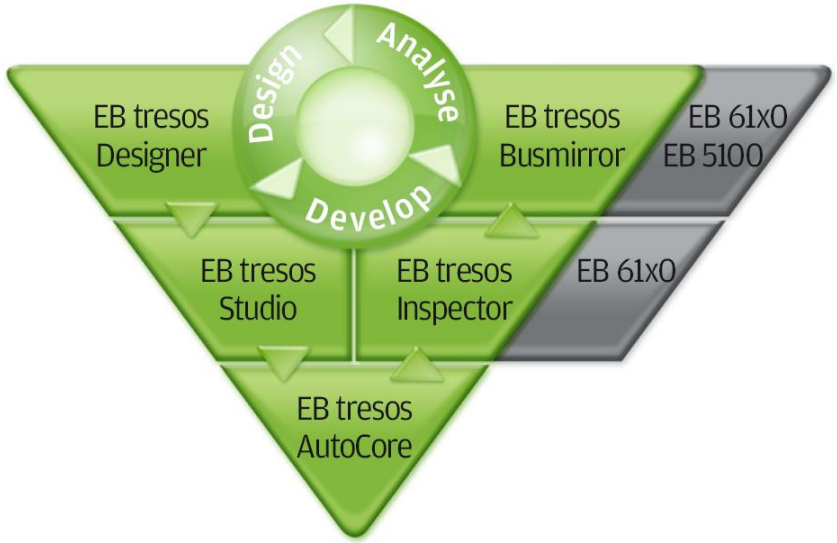
使用AUTOSAR配置工具进行参数配置

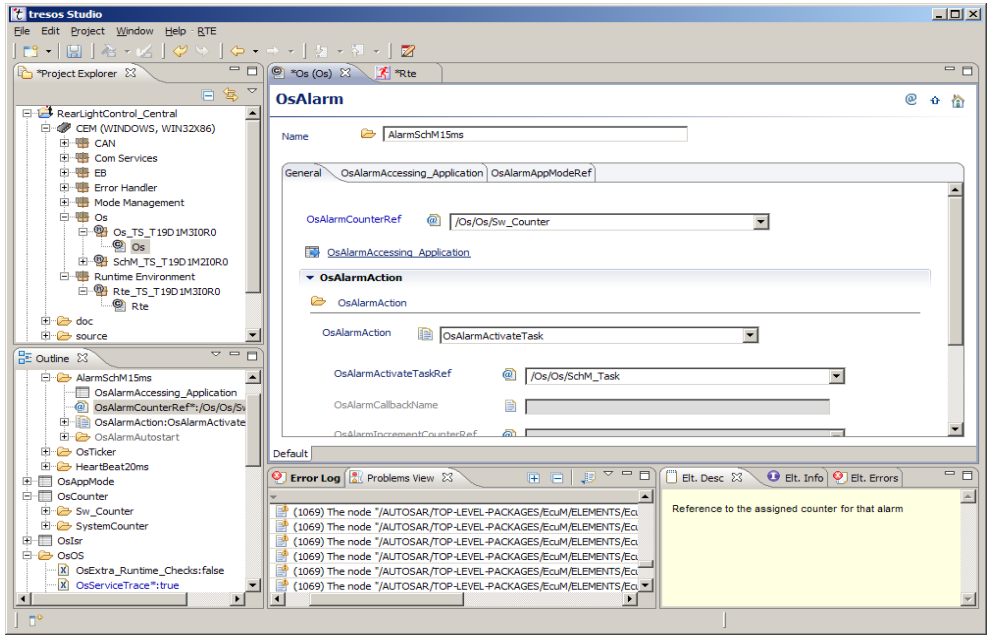
生成配置代码

将生成的配置代码与MCAL及用户代码集成，使用IDE或Makefile编译&调试

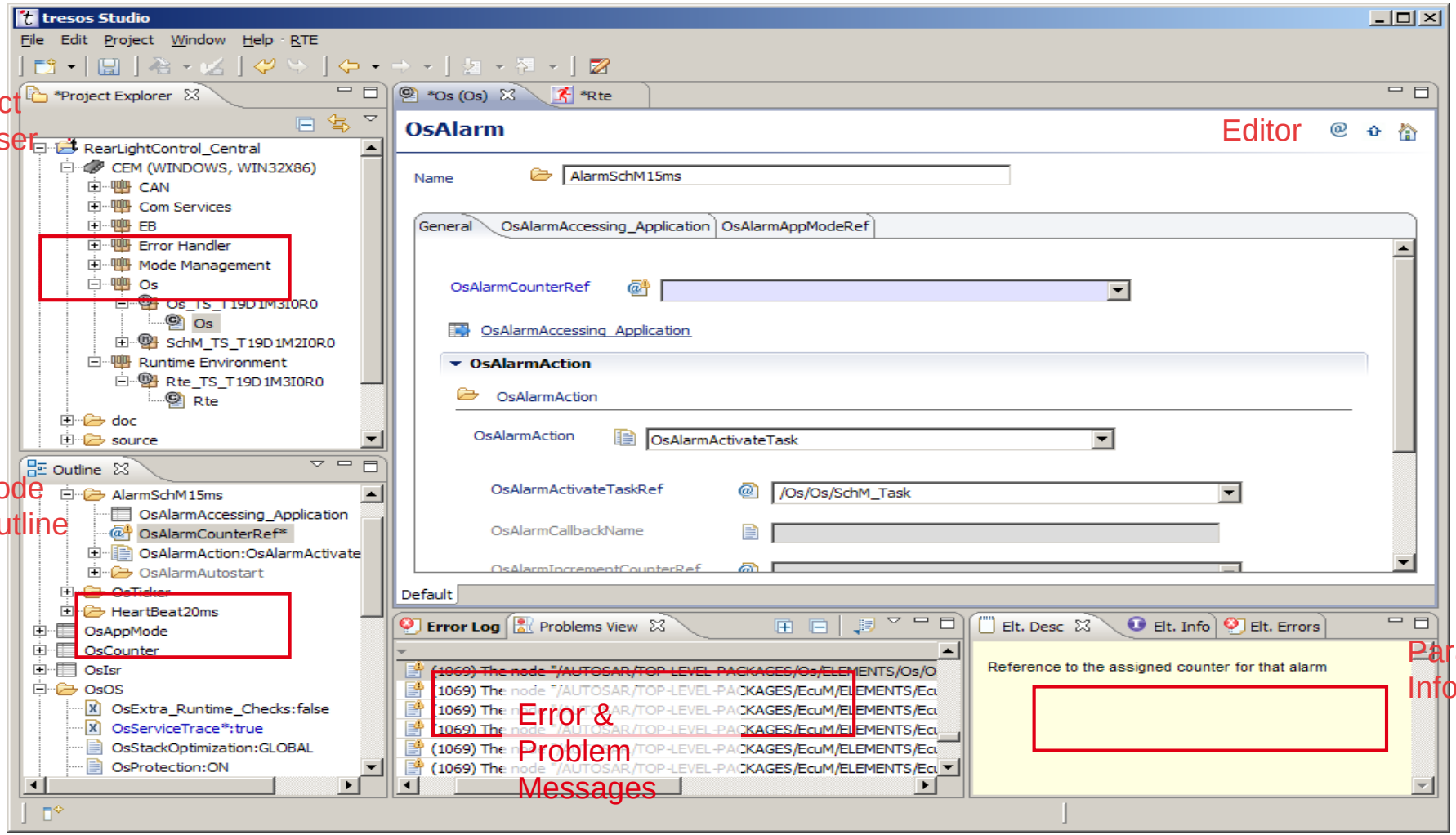
NXP AUTOSAR MCAL --- 配置工具

- ✓ EB (Elektrobit) tresos Studio是一个用于ECU标准软件配置，参数检查，代码生成的一个可视化工具。
- ✓ 该工具基于Eclipse开发，有详细的帮助文档。
- ✓ 该工具支持完整AUTOSAR标准
- ✓ NXP AUTOSAR OS及MCAL均使用该工具配置

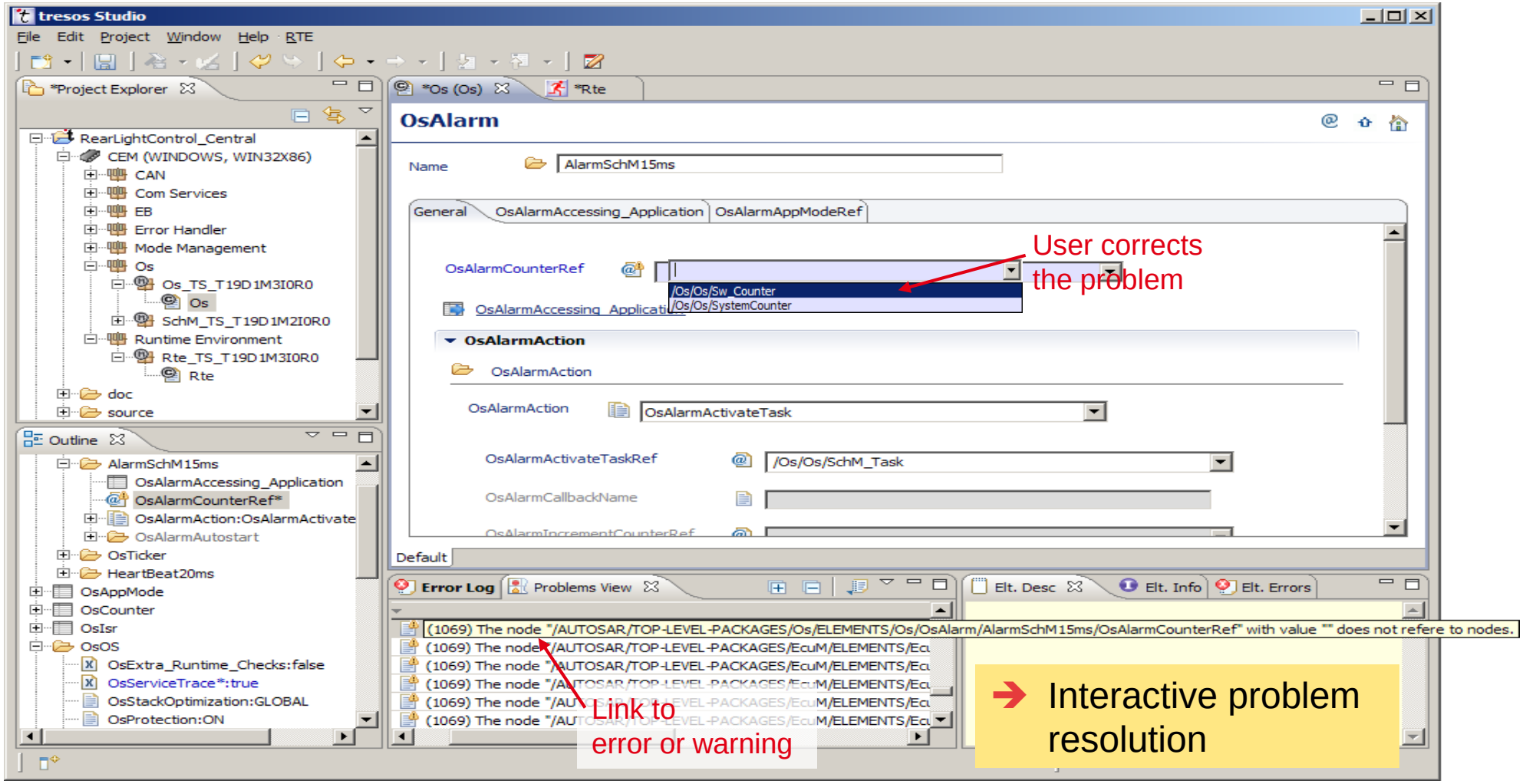




NXP AUTOSAR MCAL --- 工具界面



NXP AUTOSAR MCAL --- 工具界面



MCAL
基本概念

MCAL基本概念 --- 寄存器初始化

MCAL模块 <Mode>_Init 函数：

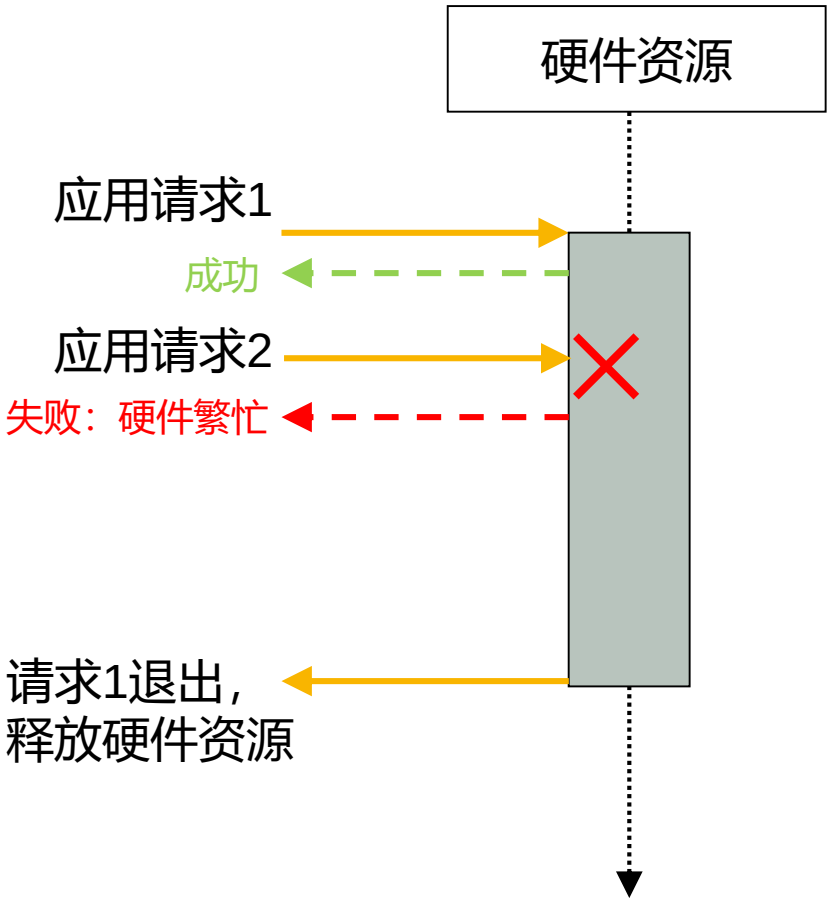
- 如果某个模块寄存器只允许本模块使用，则该模块负责初始化。
- 如果某个寄存器的值会影响很多硬件模块并且它是一个I/O寄存器，那么这个寄存器由PORT模块负责初始化。
- 如果某个寄存器的值会影响很多硬件模块但是它不属于I/O寄存器，那么这个寄存器由MCU或MCL模块负责初始化。
- 对于那些只能写一次的寄存器，在芯片复位后，由启动代码直接初始化。
- 其他寄存器由启动代码负责初始化。

MCAL基本概念 --- 初始化流程

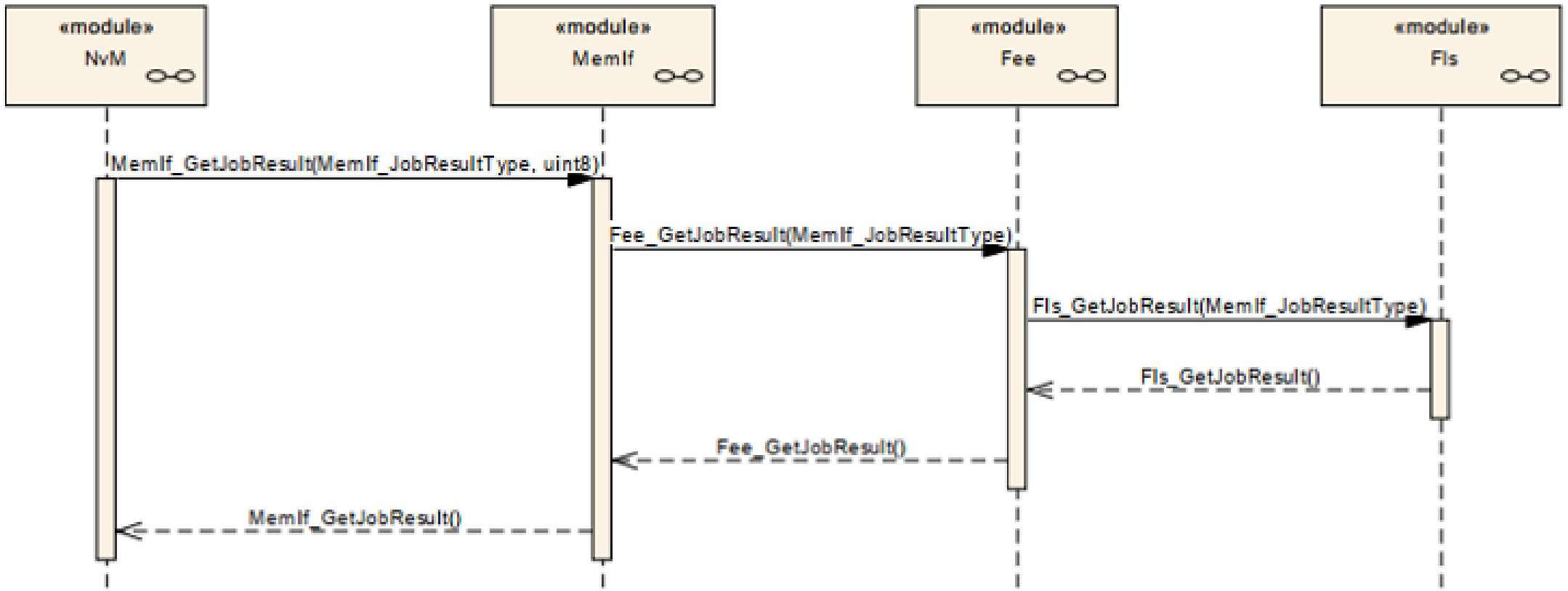
每个MCAL模块初始化按照以下流程进行（Req:BSW12068）：

- 关闭全局中断
- 初始化本模块负责的所有寄存器
- 初始化整个硬件模块
- 如有需要，打开全局中断

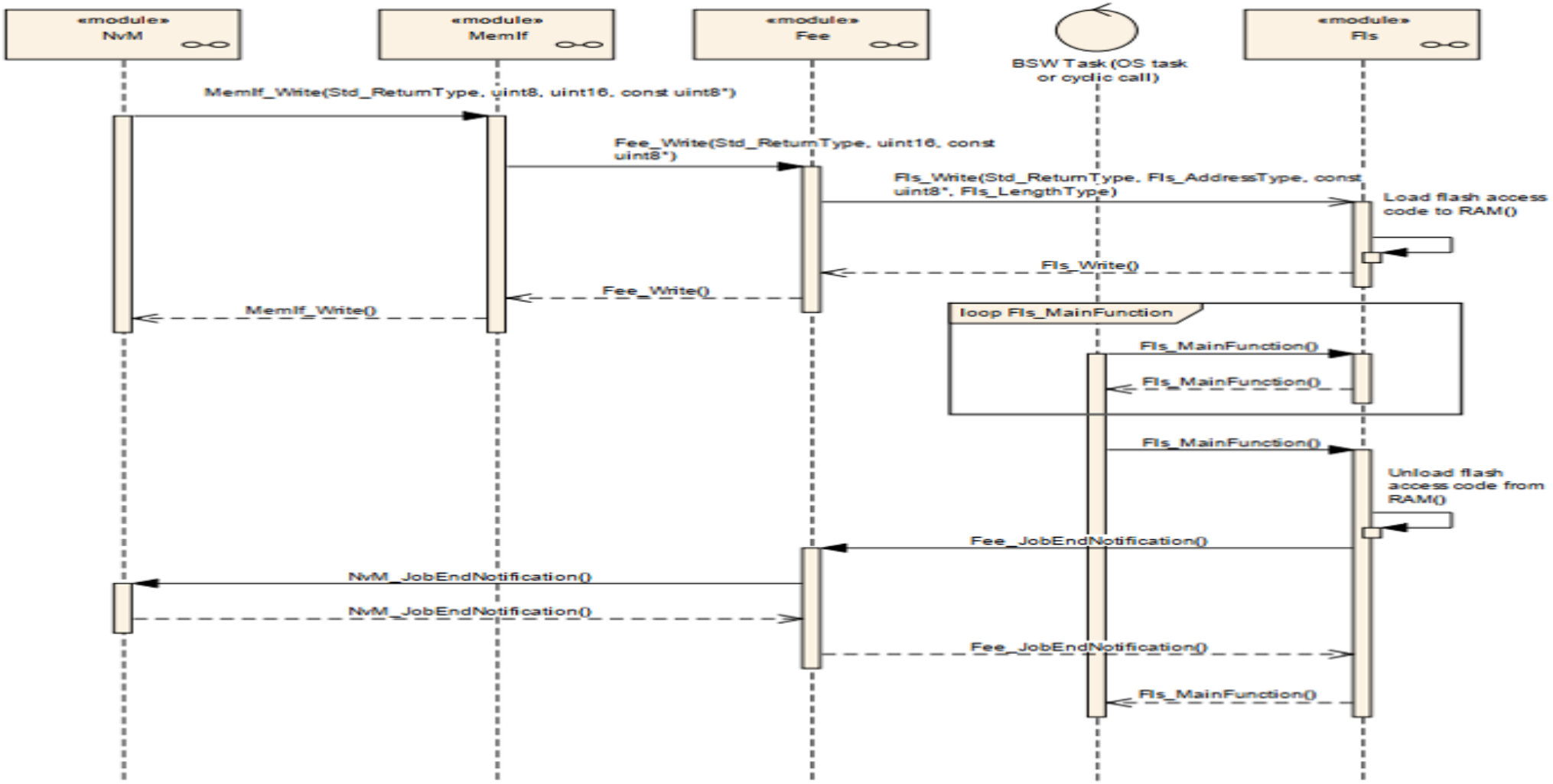
MCAL基本概念 --- 重入



MCAL基本概念 --- 同步/异步



MCAL基本概念 --- 同步/异步



MCAL基本概念 --- 中断/轮询

轮询

Can HW object 0

Can HW object 1

Can HW object 2

Can HW object 3

Can HW object 4

中断

Can HW object 0

Can HW object 1 → 中断函数处理

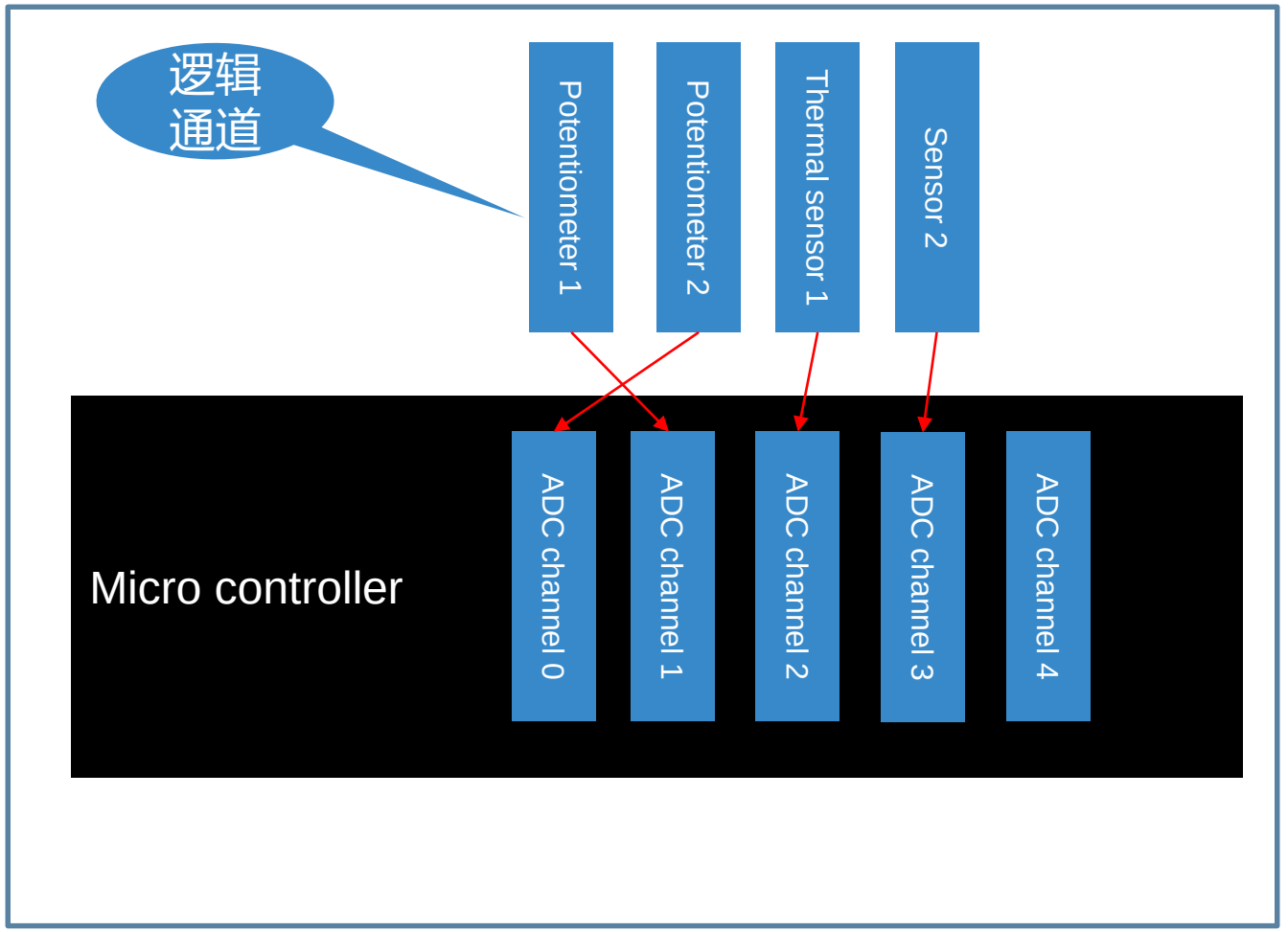
Can HW object 2

Can HW object 3 → 中断函数处理

Can HW object 4

MCAL基本概念 --- 硬件通道抽象

- 物理通道（ADC,Timer,Pwm…）被抽象为逻辑通道。
- 每个逻辑通道有一个唯一的ID映射到每个物理通道。
- 逻辑通道与物理通道的映射在配置阶段完成。



MCAL
常用模块

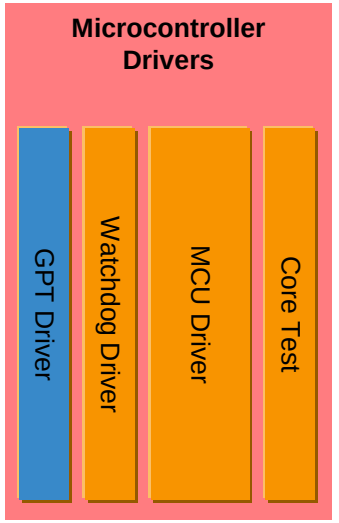
通用计时器
(GPT)

GPT

对内部硬件计时器/计数器初始化和控制

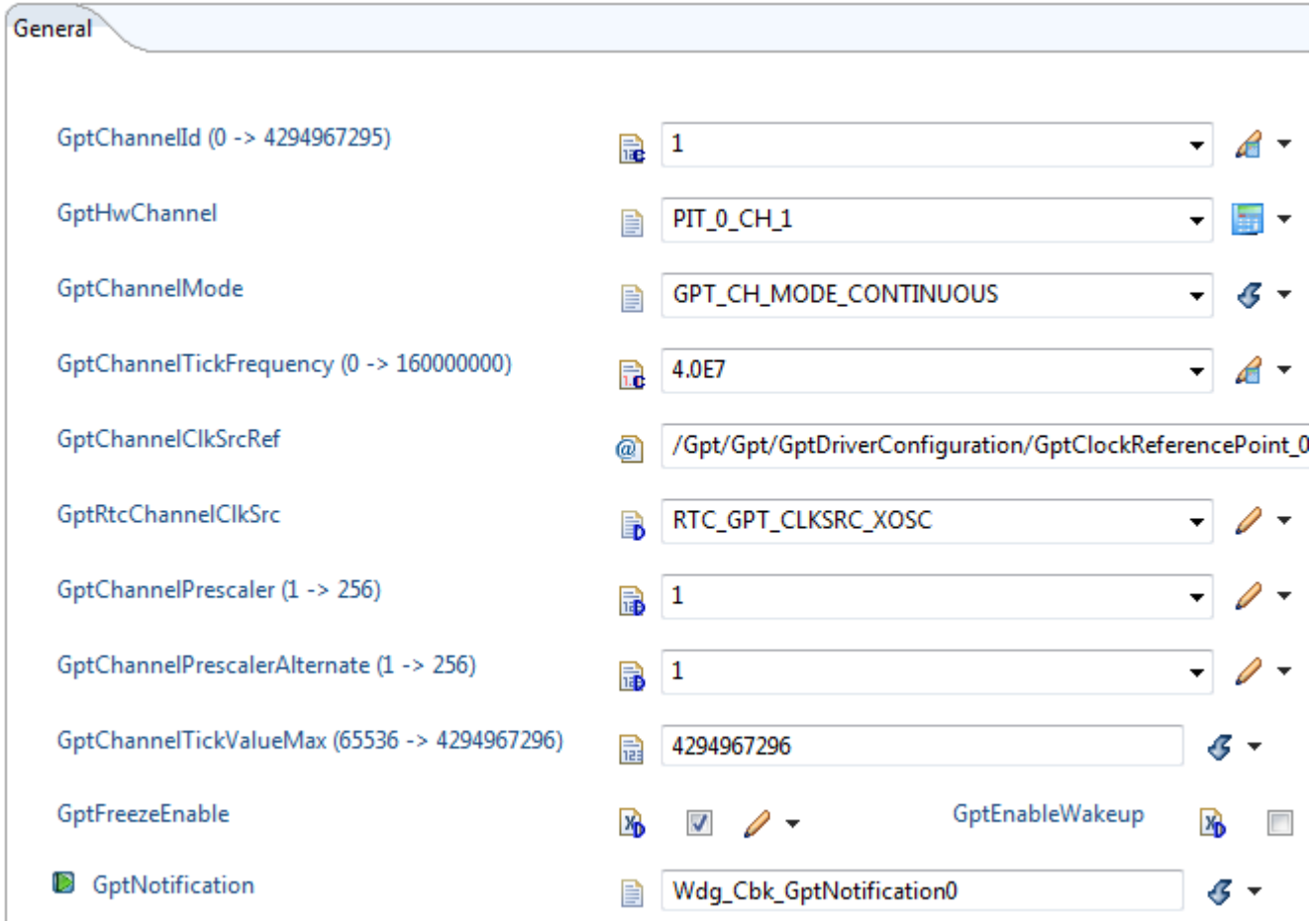
提供以下服务

- ✓ 启动/停止硬件计时器
- ✓ 获取计时器值
- ✓ 控制触发中断通知
- ✓ 控制唤醒中断通知
- ✓ 单脉冲模式
- ✓ 连续模式

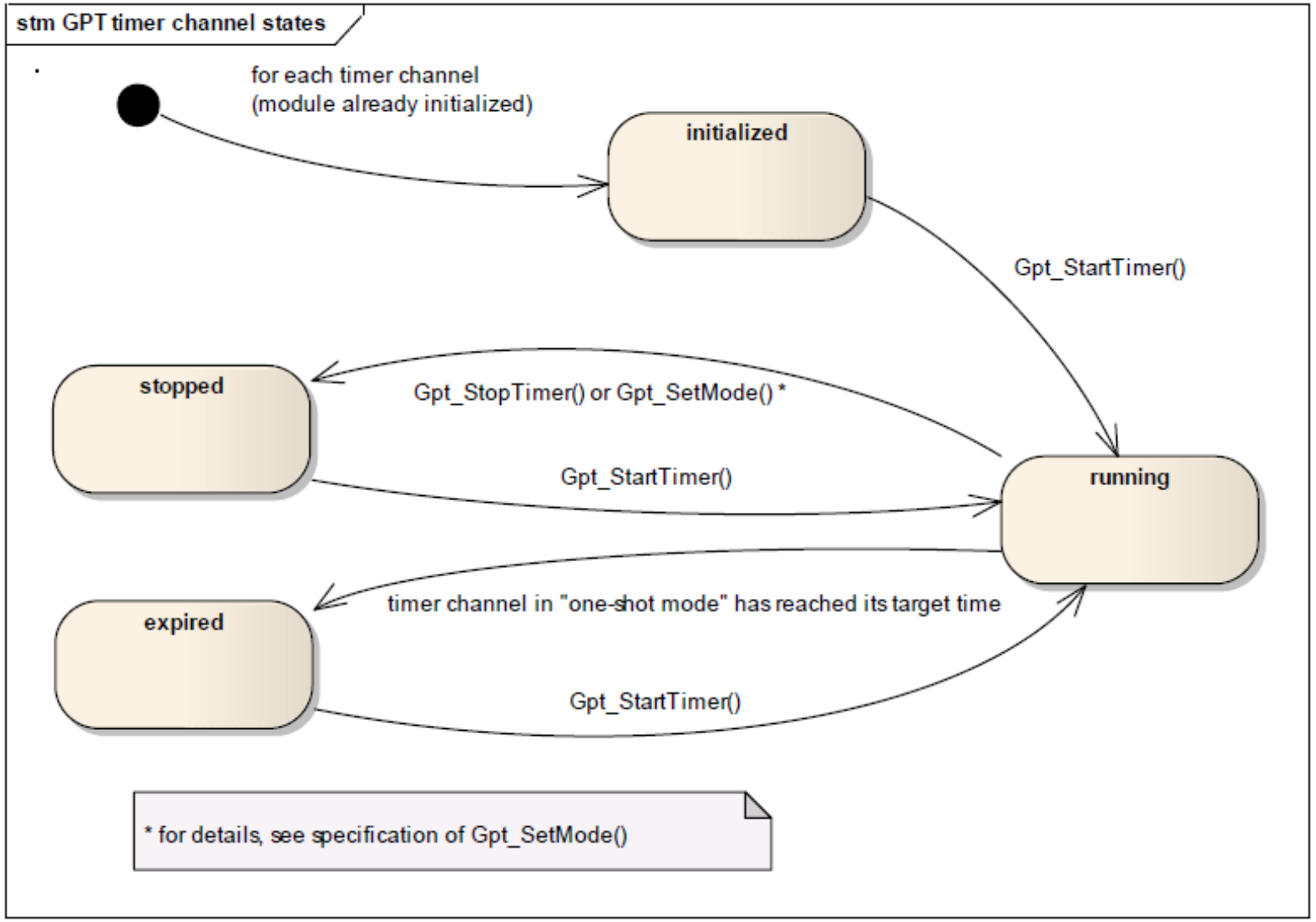


GPT 配置

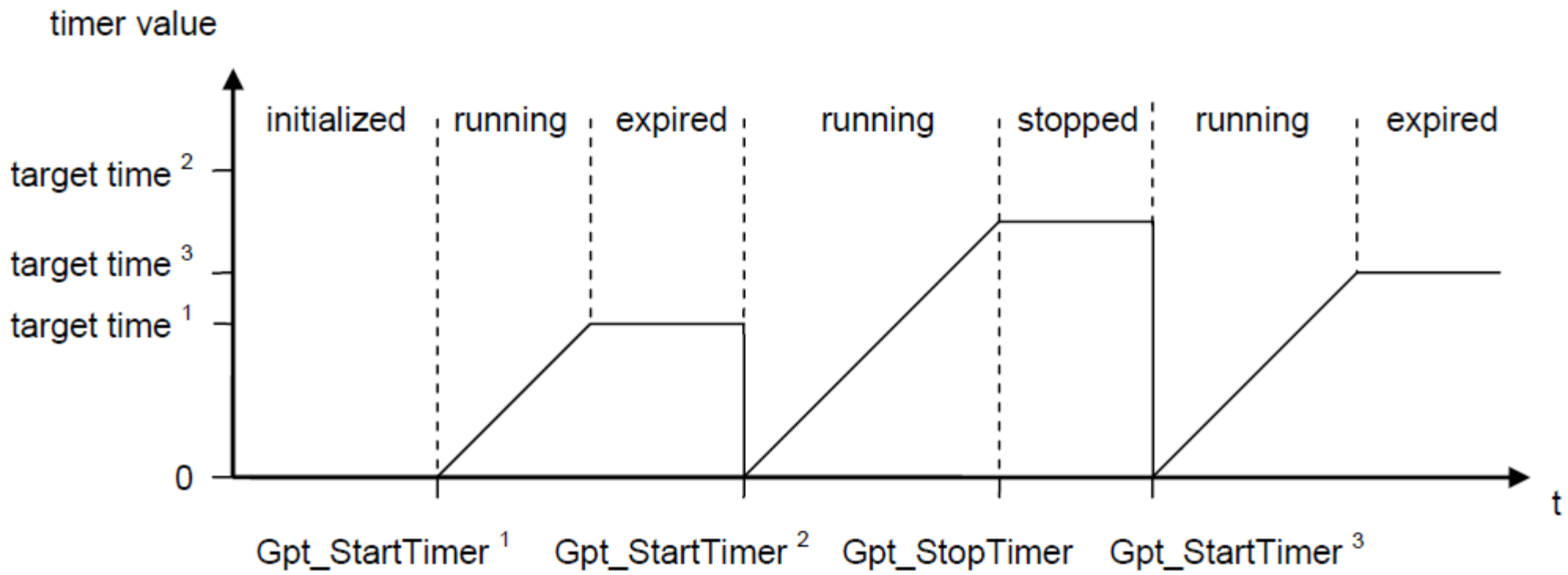
抽象层: GPT 通道 → 硬件计时器



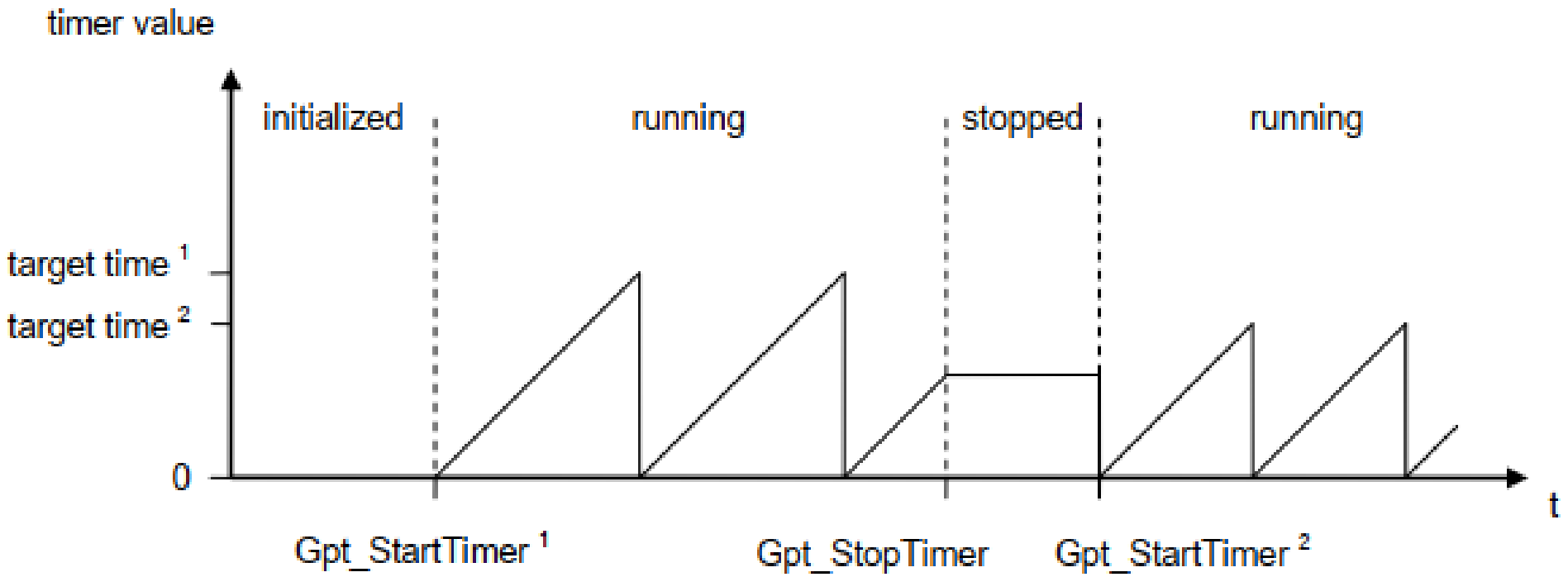
GPT 状态机


stm GPT timer channel states
for each timer channel
(module already initialized)
initialized
Gpt_StartTimer()
stopped
Gpt_StopTimer() or Gpt_SetMode() *
Gpt_StartTimer()
running
timer channel in "one-shot mode" has reached its target time
expired
Gpt_StartTimer()
* for details, see specification of Gpt_SetMode()

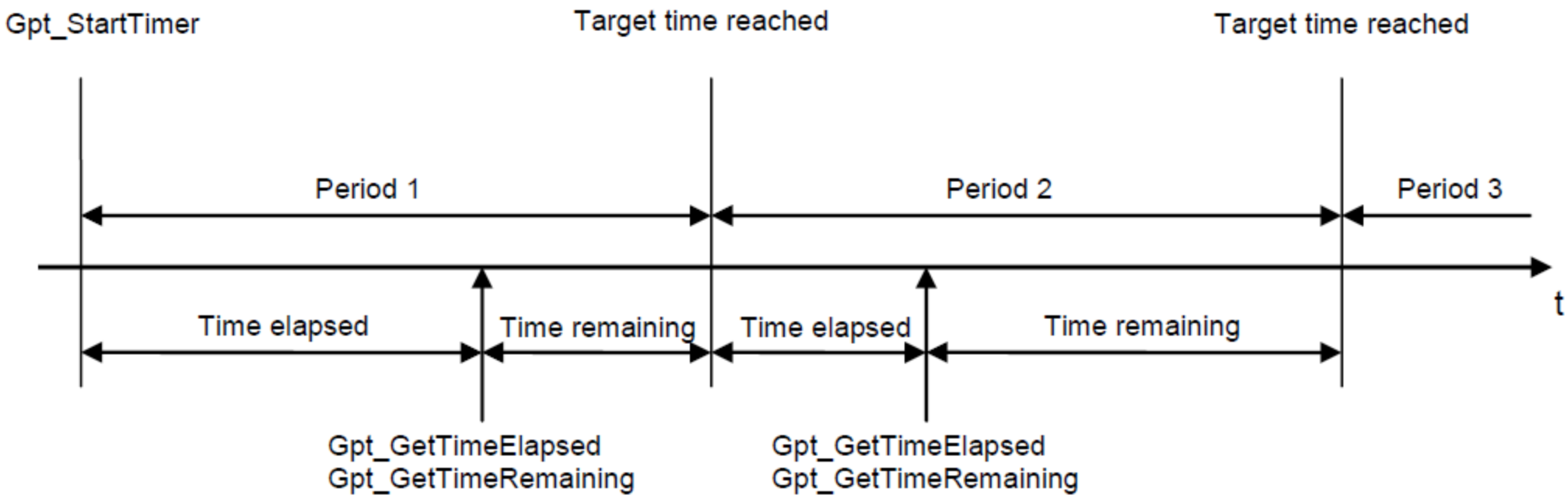
GPT 单脉冲模式


timer value
initialized
running
expired
running
stopped
running
expired
target time 2
target time 3
target time 1
0
t
Gpt_StartTimer 1
Gpt_StartTimer 2
Gpt_StopTimer
Gpt_StartTimer 3

GPT 连续模式


timer value
initialized
running
stopped
running
target time 1
target time 2
0
t
Gpt_StartTimer 1
Gpt_StopTimer
Gpt_StartTimer 2

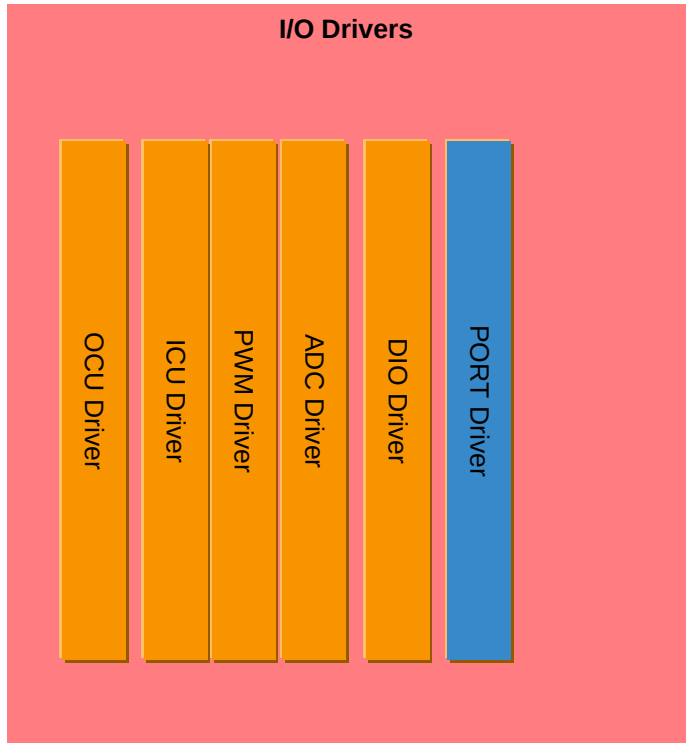
Gpt_GetTimeElapsed/Gpt_GetTimeRemaining



端口及数字输入输出
(PORT & DIO)

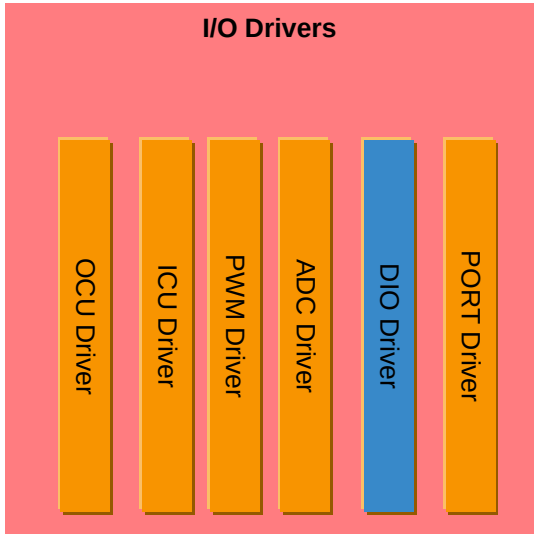
PORT

- ✓ 负责初始化所有端口和Pin脚。
- ✓ 一个端口可能有多个功能（取决于不同硬件平台）。如：
  - GPIO
  - SPI
  - SCI
  - PWM
  - CAN
  - LIN
- ✓ 负责在运行时重新初始化指定端口（如改变功能，但通常不建议这么做）。



DIO

- ✓ 该模块不需要初始化，已由PORT模块完成。
- ✓ 提供APIs获取端口状态或控制端口状态。
- ✓ 该模块的API均为同步操作，没有缓存。
- ✓ 读写操作必须连续，read-modify-write不允许。



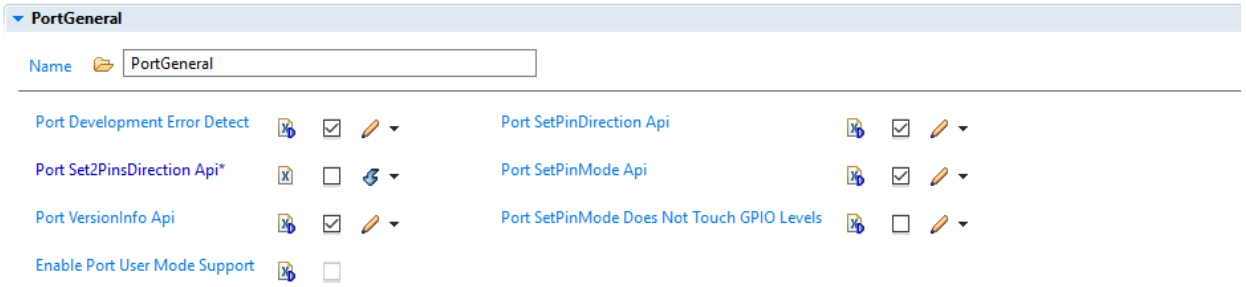
PORT APIs

Port:

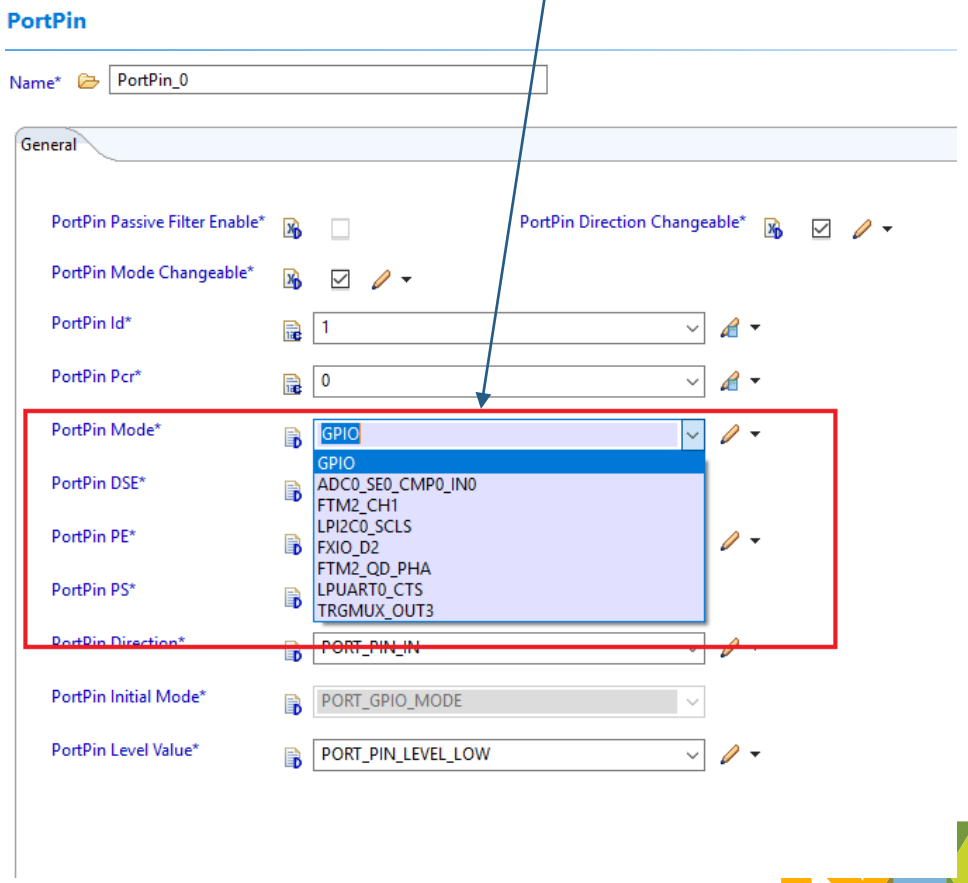
Run time apis

- void Port_Init(const Port_ConfigType* ConfigPtr)
- void Port_SetPinDirection(Port_PinType Pin, Port_PinDirectionType Direction)
- void Port_RefreshPortDirection(void)
- void Port_GetVersionInfo(Std_VersionInfoType* versioninfo)
- void Port_SetPinMode(Port_PinType Pin, Port_PinModeType Mode)
- Void Port_Set2PinsDirection (Port_PinType pin1, Port_PinType pin2, Direction)

Enable APIs by configuration



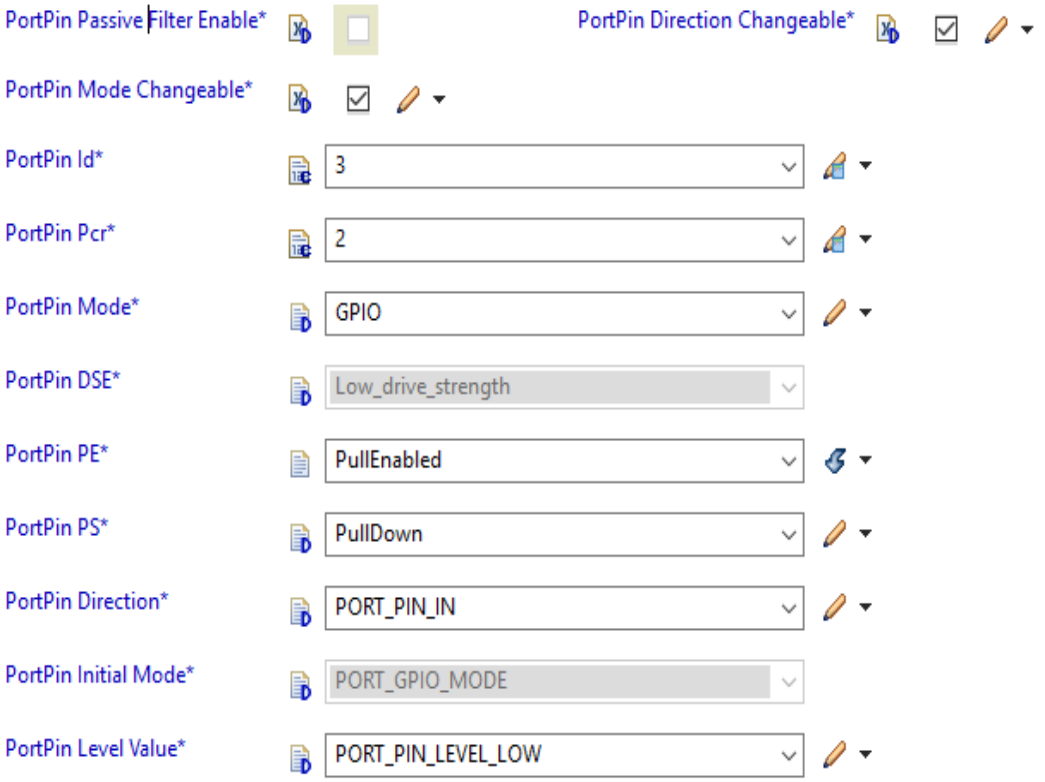
Pin 功能选择



PORT底层数据结构

```
typedef struct
{
  VAR(Port_InternalPinIdType, AUTOMATIC)   Pin;     /**< @brief Pin Defined on PORT */
  VAR(uint32,                 AUTOMATIC)   u32PCR;  /**< @brief Pad Control Register */
  VAR(uint8,                  AUTOMATIC)   u8PDO;   /**< @brief Pad Data Output */
  VAR(Port_PinDirectionType,  AUTOMATIC)   ePDDir; /**< @brief Pad Data Direction */
  VAR(boolean,                AUTOMATIC)   bGPIO;   /**< @brief GPIO initial mode*/
  VAR(boolean,                AUTOMATIC)   bDC;     /**< @brief Direction changebility*/
  VAR(boolean,                AUTOMATIC)   bMC;     /**< @brief Mode changebility*/
} Port_Port_LLD_PinConfigType;
```

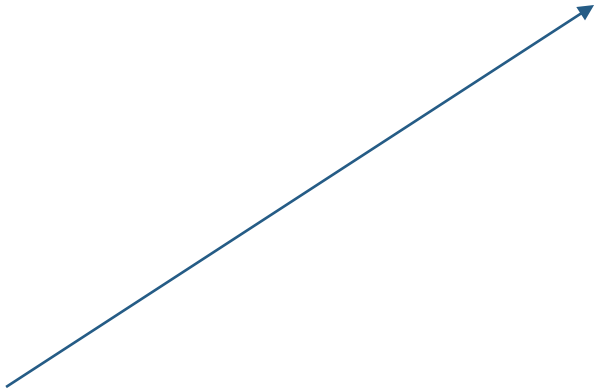
配置界面与代码相对应



Dio

- Dio_LevelType **Dio_ReadChannel**(Dio_ChannelType ChannelId)
- void **Dio_WriteChannel**(Dio_ChannelType ChannelId, Dio_LevelType Level)
- Dio_PortLevelType **Dio_ReadPort**(Dio_PortType PortId)
- void Dio_WritePort(Dio_PortType PortId, Dio_PortLevelType Level)
- Dio_PortLevelType **Dio_ReadChannelGroup**(const Dio_ChannelGroupType* ChannelGroupIdPtr)
- void Dio_WriteChannelGroup(const Dio_ChannelGroupType* ChannelGroupIdPtr, Dio_PortLevelType Level)
- void Dio_GetVersionInfo(Std_VersionInfoType* VersionInfo)
- Dio_LevelType **Dio_FlipChannel**(Dio_ChannelType ChannelId)

Driver	*Name for a Port Pin*	*Name for Subset of Adjacent pins on one port*	*Name for a whole port*
DIO Driver	Channel	Channel Group	Port
PORT Driver	Port pin	--	Port



Dio 通道

一个通道指一个物理引脚 。

Dio 端口

一个端口通常包含多个通道，并且他们属于一个硬件组。

Dio 通道组

一个通道组包含多个相邻的通道，且这些通道属于同一个端口。

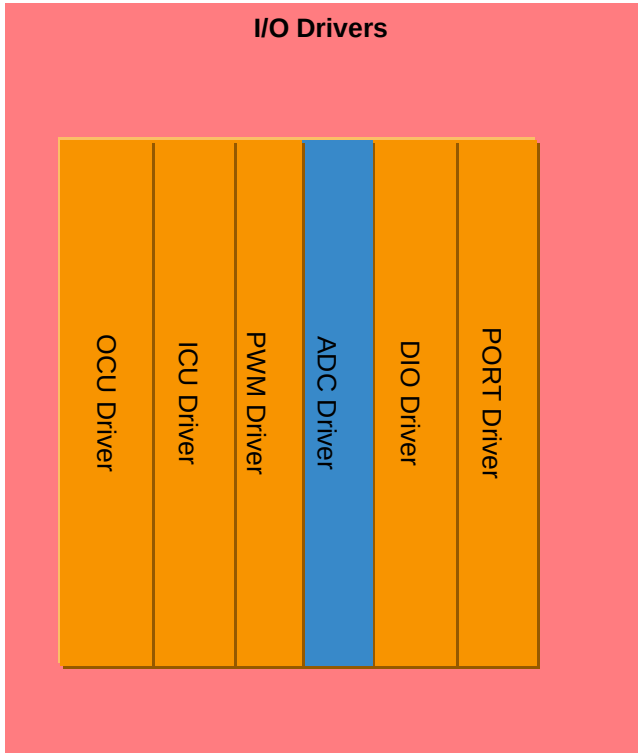
模数转换
(ADC)

ADC

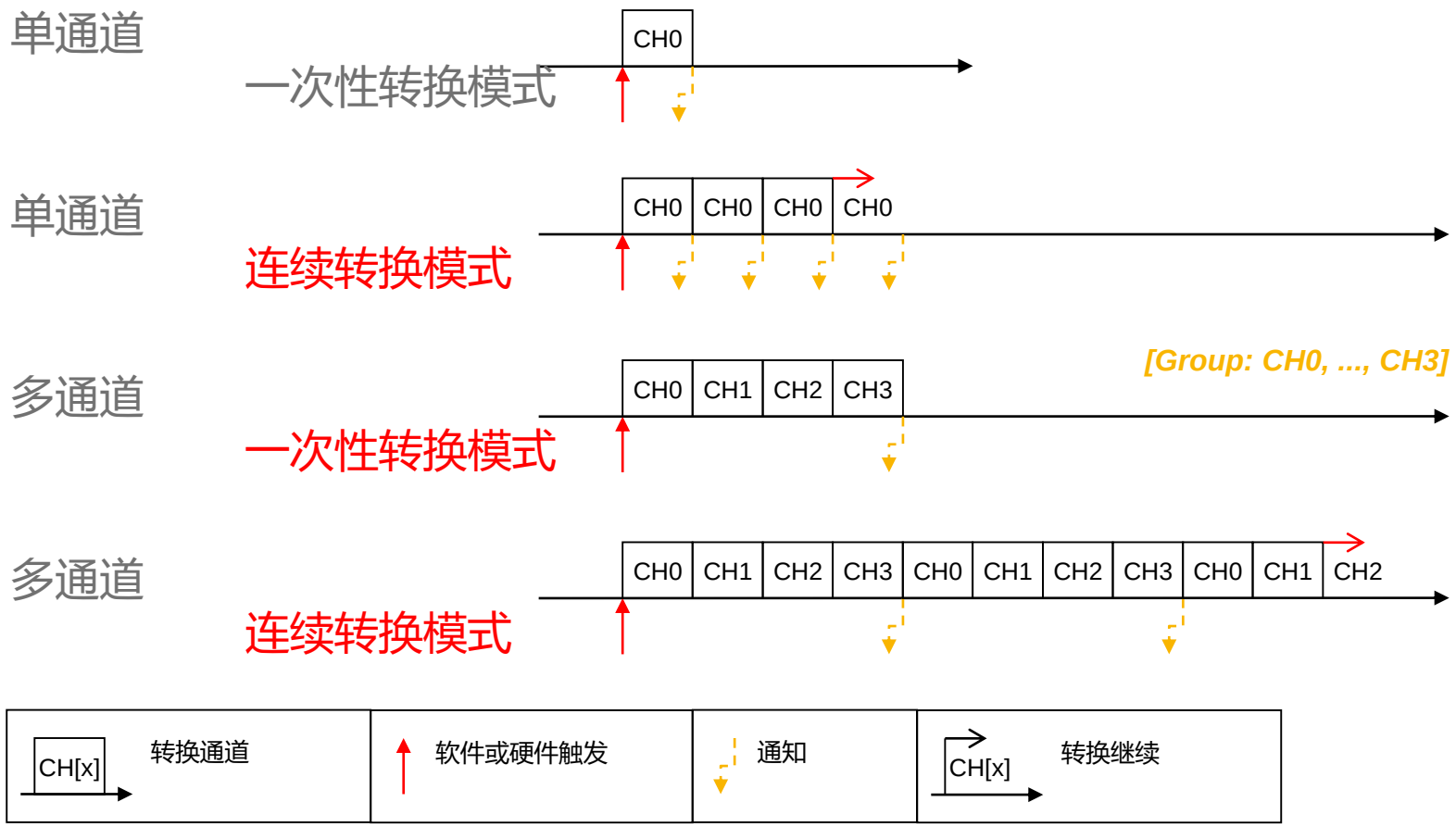
- ✓ 负责初始化及控制ADC硬件模块
- ✓ 还提供以下服务：
  - 启动/停止转换
  - 使能/关闭转换触发源
  - 打开/关闭通知机制
  - 反初始化

- ✓ 具有一次性/连续转换功能
- ✓ 具有软件/硬件触发转换功能

- ✓ 具有单次/流访问模式
- ✓ 具有循环/线性缓存



ADC转换模式


单通道
一次性转换模式
CH0
单通道
连续转换模式
CH0 CH0 CH0 CH0
[Group: CH0, ..., CH3]
多通道
一次性转换模式
CH0 CH1 CH2 CH3
多通道
连续转换模式
CH0 CH1 CH2 CH3 CH0 CH1 CH2 CH3 CH0 CH1 CH2
CH[x]
转换通道
软件或硬件触发
通知
CH[x]
转换继续

输入捕捉
(ICU)

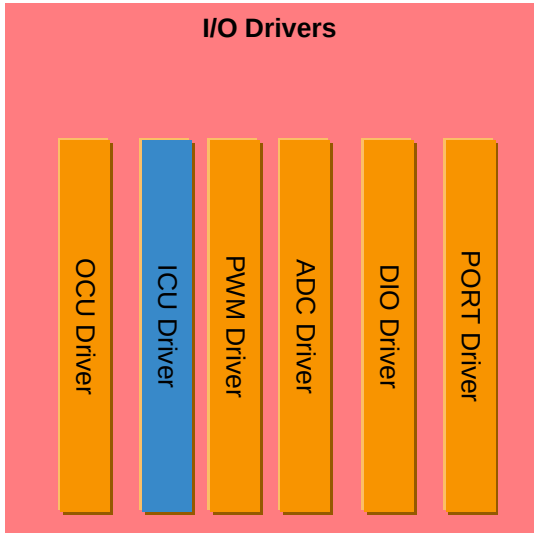
ICU

ICU驱动提供以下服务：

- ✓ 信号边沿检测通知（如外部中断）
- ✓ 唤醒中断控制
- ✓ 周期信号时间测量
- ✓ 非周期信号的边沿时间测量
- ✓ 边沿统计

应用场景：

- ✓ 统计PWM信号脉冲数
- ✓ 统计PWM信号频率和占空比
- ✓ 外部中断或唤醒中断控制



ICU配置

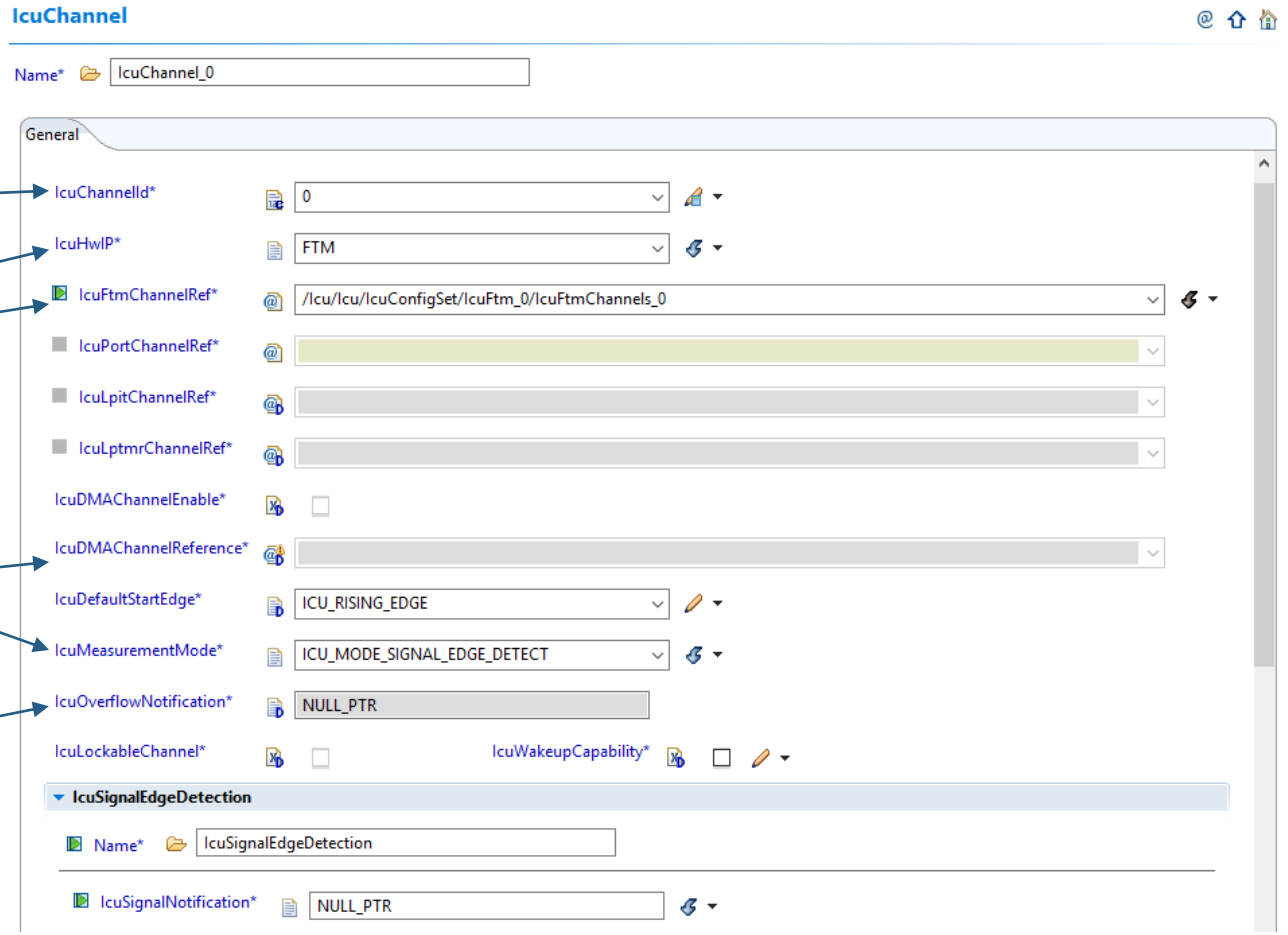
这里主要展示了ICU硬件单元配置

逻辑通道ID

硬件相关配置

时间戳功能

可配置的通知回掉函数



脉冲调制（PWM）

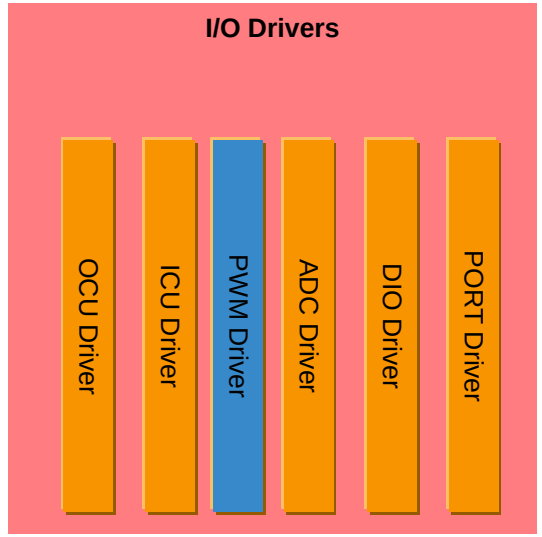
PWM

本模块负责初始化及控制PWM硬件模块

- ✓ 支持产生可变脉冲宽度
- ✓ 支持改变周期和占空比
- ✓ 支持极性配置
- ✓ 支持对其模式选择（边沿对其，中心对其）

应用场景：

- ✓ 电机控制
- ✓ 电源变换控制
- ✓ 无线充电
- ✓ ADC触发等



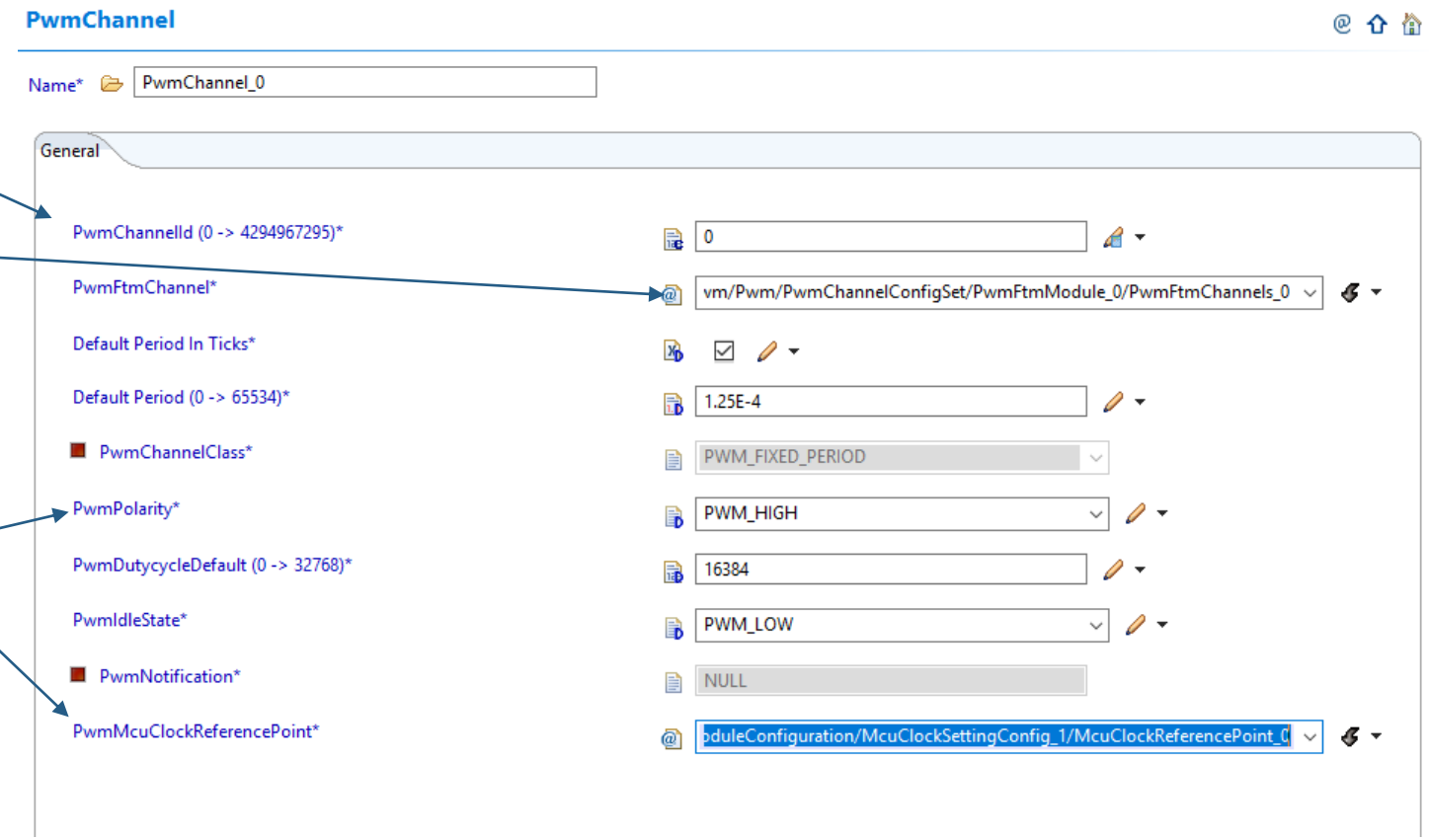
PWM配置

主要配置信息：

逻辑通道ID

参考时钟

极性配置

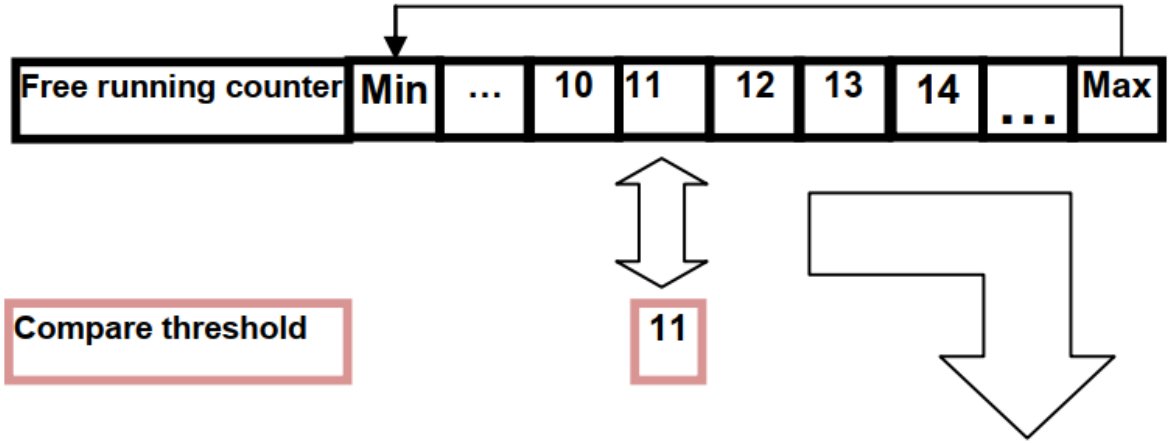


输出比较
(OCU)

OCU

该模块提供输出比较单元的初始化函数和访问接口

工作原理：
启动后，计时器开始计数，然后与预先设置好的阈值进行比较，如果计时器的值到达阈值，则输出信号并调用用户配置的回掉函数通知应用层。



输出端口输出信号以及通知应用层

OCU配置

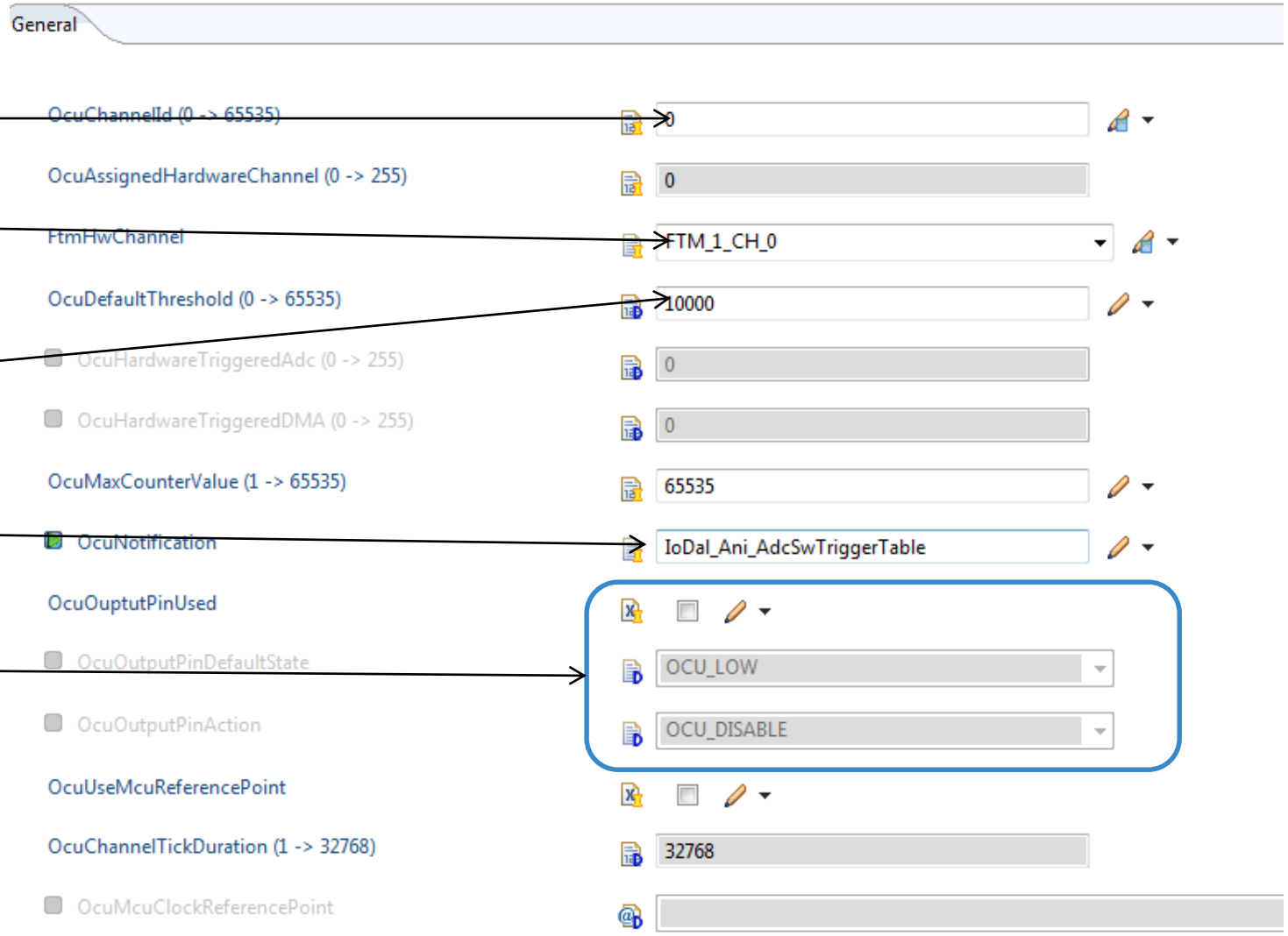
逻辑通道ID

物理通道

默认阈值
该值在没有调用
设置阈值相关函数时使用

回调函数

输出端口配置



串行外设接口（SPI）

SPI

AUTOSAR SPI模块只支持主模式。

NXP将其扩展为主从模式都支持。

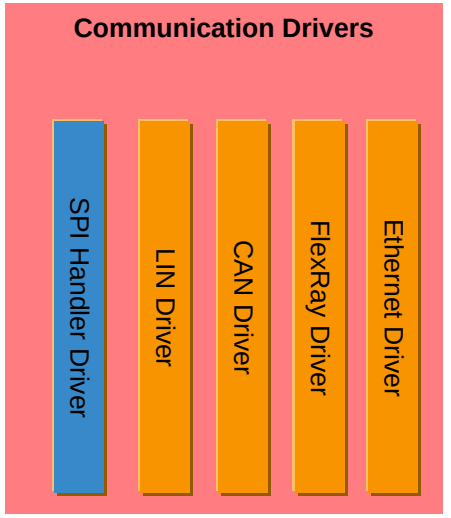
本模块对连接在SPI总线上的设备提供读写操作服务。

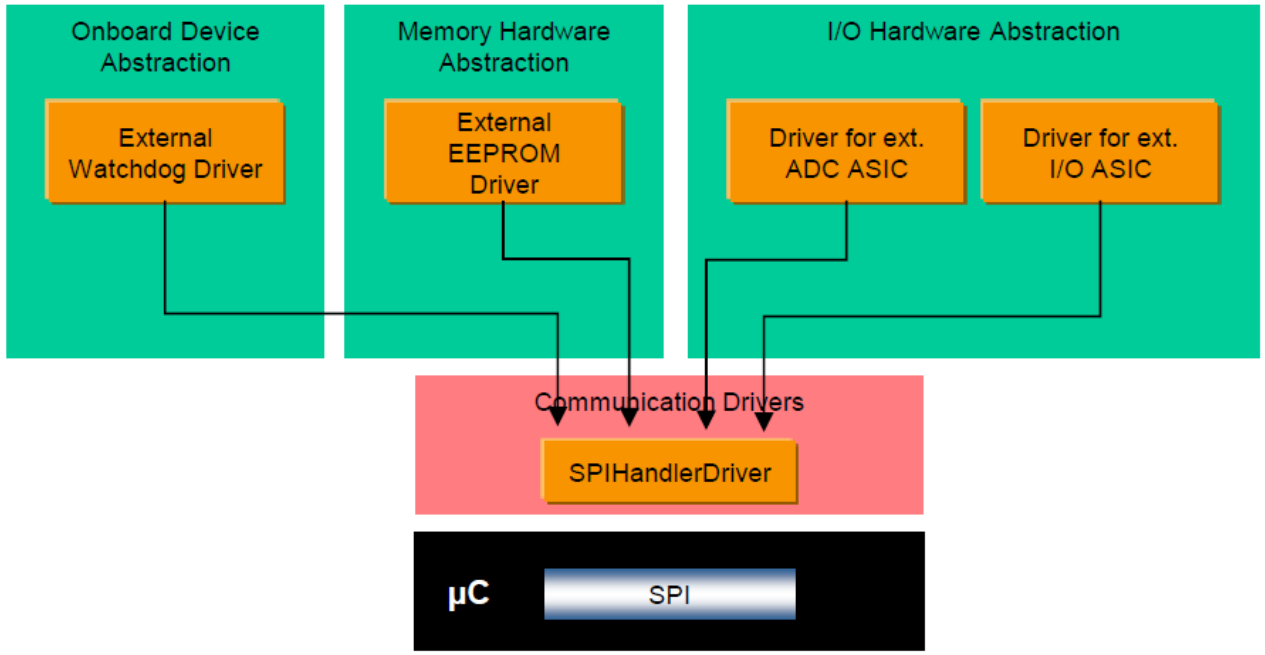
支持多个客服端同时访问一个或多个SPI总线（如：EEPROM, Watchdog等）。

每个用户请求抽象为一个任务，一个任务为一个原子操作。

每个任务由优先级属性。

每个物理通道可以接受多个任务。





SPI功能裁剪

提供3个可配置的功能等级：

- LEVEL 0：简易型，仅支持同步及轮询操作
- LEVEL 1：基础型，支持异步、中断、优先级操作
- LEVEL 2：加强型，支持同步、异步、中断、轮询、优先级

内部带数据缓存，用户也可配置为外部缓存

	Buffer allowed (0=IB, 1=EB, 2=both)	Synchronous	Asynchronous	Priority	IRQ	Polling		
Level 0	x	x	o	o	o	x		
Level 1	x	o	x	x	x	o		
Level 2	x	x	x	x	x	x		

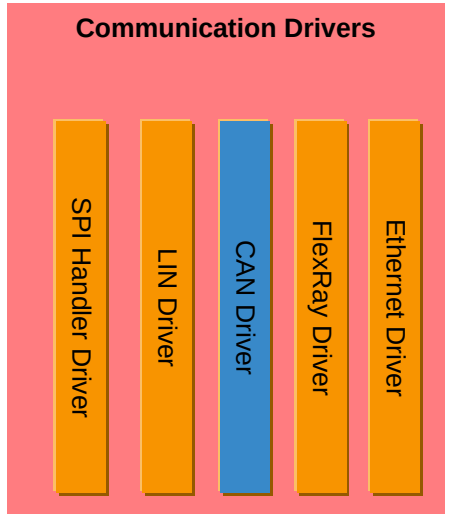
控制器局域网（CAN）

CAN

CAN驱动用于CAN控制器的抽象访问，负责报文发送和接收以及控制器状态（睡眠，唤醒）切换。

本模块支持以下功能：

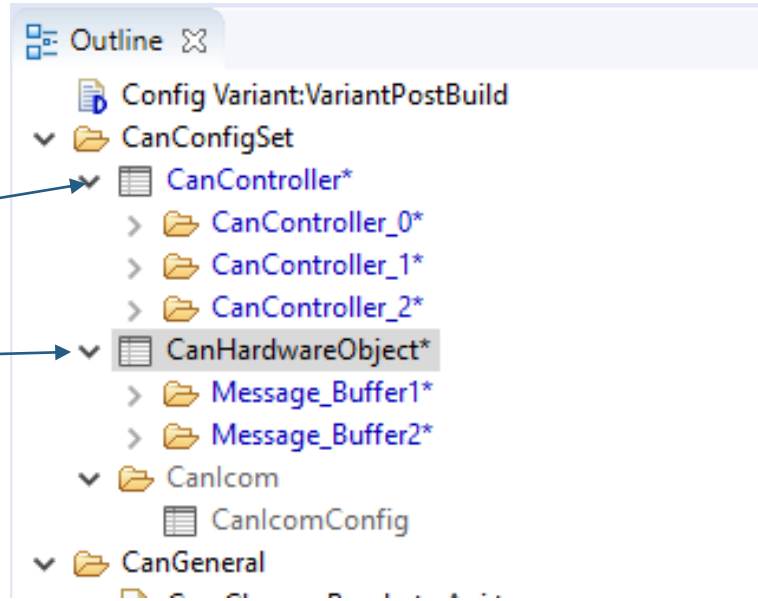
- ✓11位标准帧ID
- ✓29未扩展帧ID
- ✓消息邮箱
- ✓ID硬件过滤
- ✓唤醒ID硬件过滤



CAN 配置

HW abstraction

- CanController
- CanHardwareObject

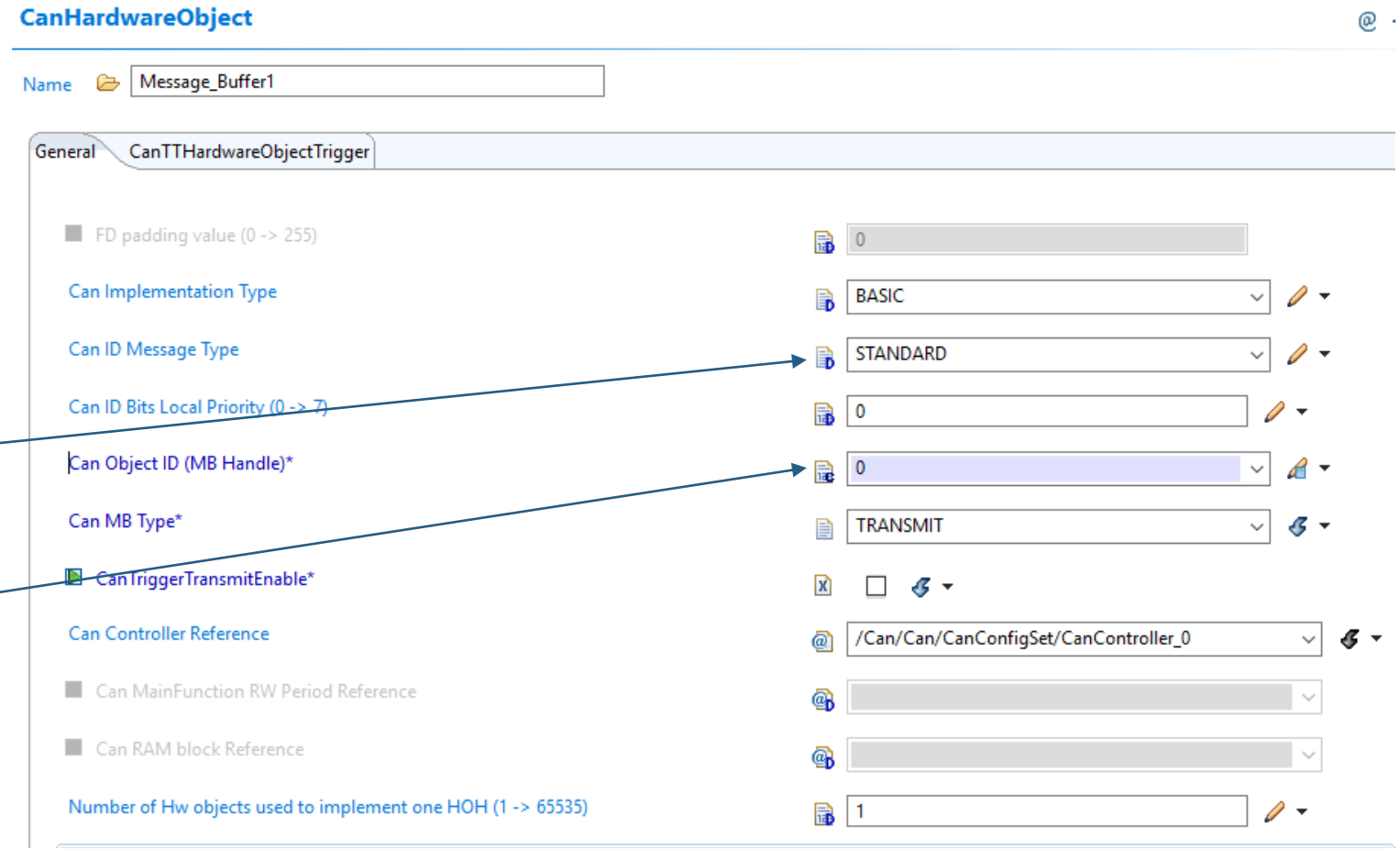


Can 硬件对象句柄(HOH) 配置

硬件对象句柄用于配置CAN接受发送消息邮箱

ID类型

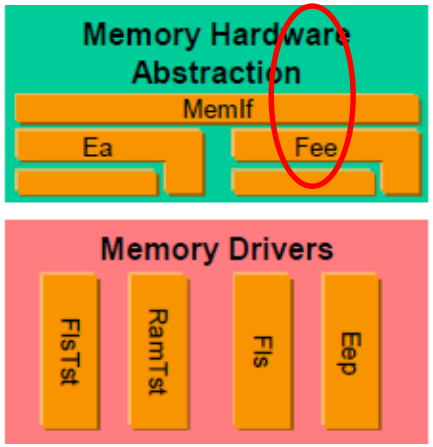
消息邮箱



FLASH 驱动（FLS）

FLS

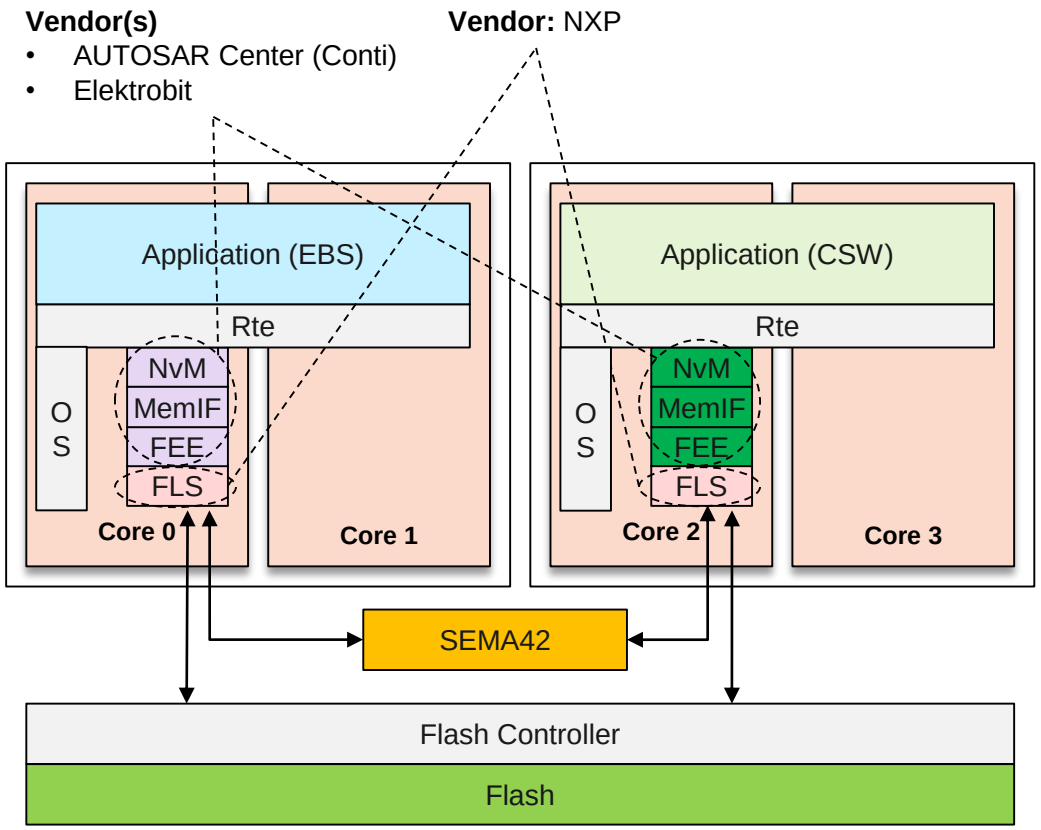
- ✓ 读、写、擦除均为异步操作
- ✓ 驱动内部没有缓存，需要应用层提供
- ✓ 当驱动层在操作缓存时，应用层需要保证数据的一致性
- ✓ 提供写/擦除保护的设置
- ✓ 提供虚拟线性地址到物理地址的映射机制
- ✓ 每一个读/写/擦除请求抽象为一个任务，然后使用异步机制处理
- ✓ 不仅支持内部FLASH，也支持外部FLASH，如QUAD SPI FLASH



FLS 多核支持

Flash 许可

多个应用，一个Flash控制器



NXP 方案:

- 使用平台硬件资源锁(SEMA42) 或者内存共享（可选）
- 硬件资源锁由MCL模块提供
- 每个应用在操作之前需要从MCL模块获取资源访问许可

FLS 机制

异步机制:

- 在同一时刻只允许一个异步请求被挂起，否则就拒绝请求
- 请求被保存在全局变量里
- 所有请求由主函数统一调度执行(Fls_MainFunction())

内存分配

- 区（Sector）: 最小擦除单元
- 页（Page）: 最小写单元
- 多个区被组装为一个连续的存储区间给上层使用
- NXP 方案: 所有区都可组装且任意顺序皆可

FLS 同步/异步功能

同步模式
- Fls_MainFunction() 初始化写/擦除并等待到其完成
- 每调用一次主函数（ Fls_MainFunction()）将写FlsMaxWriteNormalMode/FlsMaxWriteFastMode 字节
- 每调用一次主函数将擦除一个区

异步模式
- Fls_MainFunction()仅调度写/擦除任务并启动它，但是不会等待其完成
- 下一次检查上一次是否完成，如果完成则调度新的请求
- FlsMaxWriteNormalMode/FlsMaxWriteFastMode 将被忽略， 每次写FlsPageSize/FlsProgSize字节
- 每个区都需要单独配置

举例:
FLS Max Write = 16 bytes
FLS Page Size = 8 bytes
Total write size = 32 bytes

Sync: 2 x Fls_MainFunction() calls
Async: ≥ 5 x Fls_MainFunction() calls

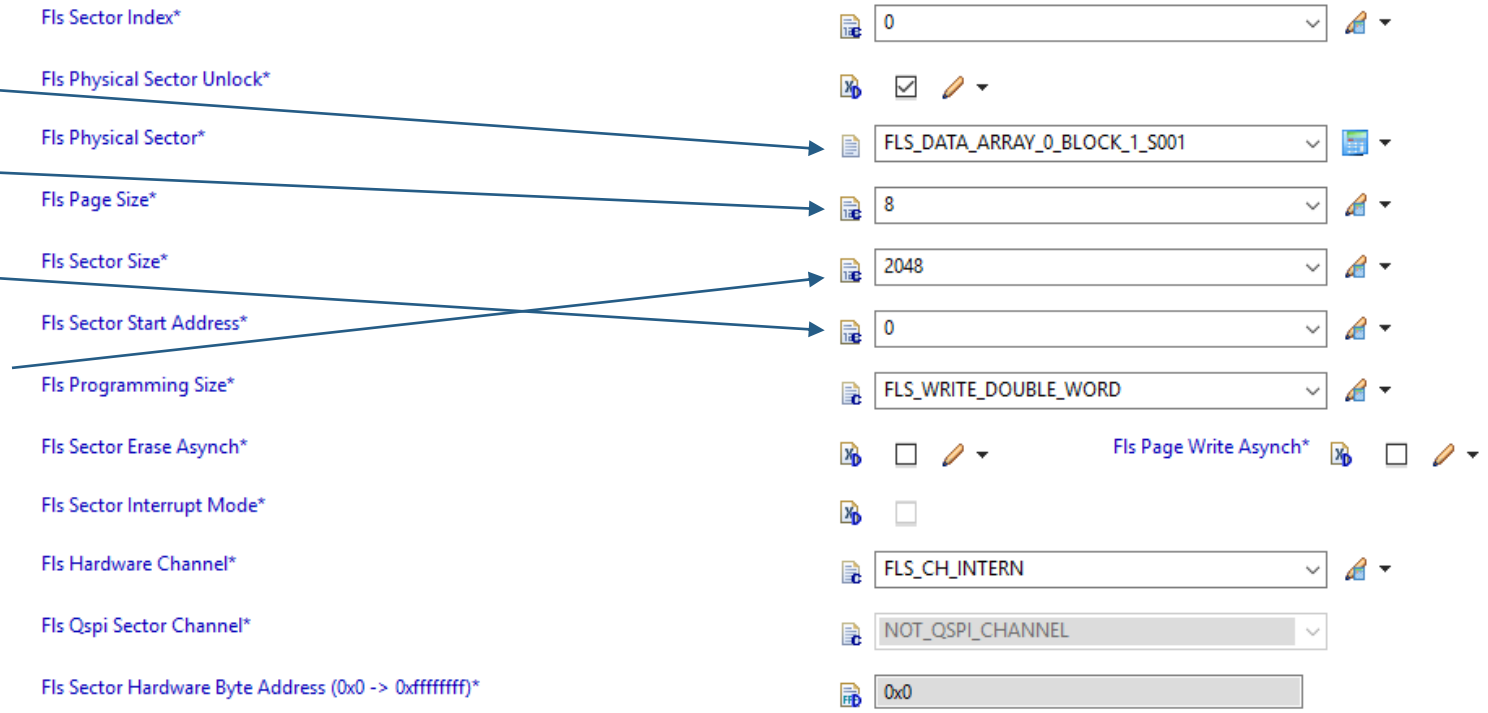
FLS 配置

物理区名字

页大小

逻辑开始地址

区大小，用于计算下一个逻辑区的起始地址

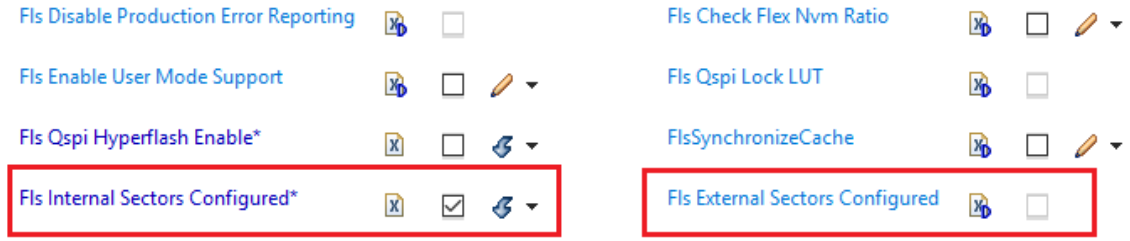


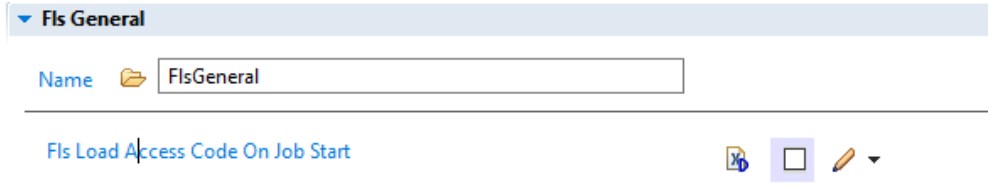
FlsSector*

Index	Name	Fls Sector...	Fls Physic...	Fls Physical Sector	Fls Page Size	Fls Sector...	Fls Sector...	Fls Progr...
0	FlsSector_0	0	☑	FLS_DATA_ARRAY_0_BLOCK_1_S001	8	2048	0	FLS_WRITE...
1	FlsSector_1	1	☑	FLS_DATA_ARRAY_0_BLOCK_1_S000	8	2048	2048	FLS_WRITE...
2	FlsSector_2	2	☑	FLS_DATA_ARRAY_0_BLOCK_1_S003	8	2048	4096	FLS_WRITE...

FLS 配置

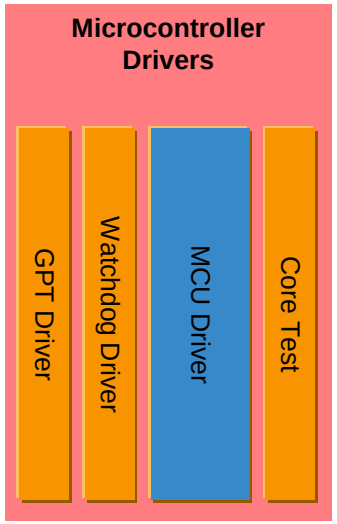
- 由于FLS同时支持内部及外部FLASH设备 。所以你需要对每个配置明确指明是内部还是外部，见右侧配置项。
- 如果配置为内部FLASH，为了防止 RWW (Read While Write) 问题，FLS驱动会将访问设备的相关代码拷贝到RAM区域执行。当任务执行完成或取消后，将相关RAM区域代码卸载。
- 为了和FEE兼容，FLS支持任务结束回调及任务错误通知机制。





微处理器单元（MCU）

MCU



该模块提供以下服务及参数配置:

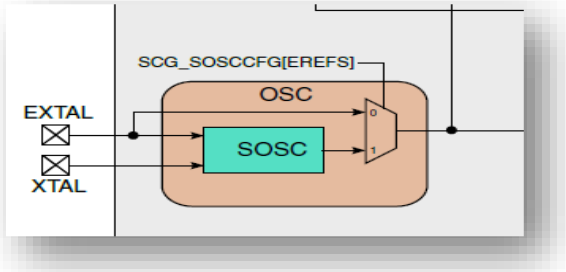
- ✓ 初始化时钟, PLL, 时钟预分频及MCU时钟分配
- ✓ 初始化RAM区域
- ✓ 激活省电模式Activation of microcontroller reduced power modes
- ✓ 激活MCU复位
- ✓ 获取系统复位原因
- ✓ 获取RAM状态
- ✓ 可配置多个时钟模式及功耗模式，供用户切换

MCU时钟初始化

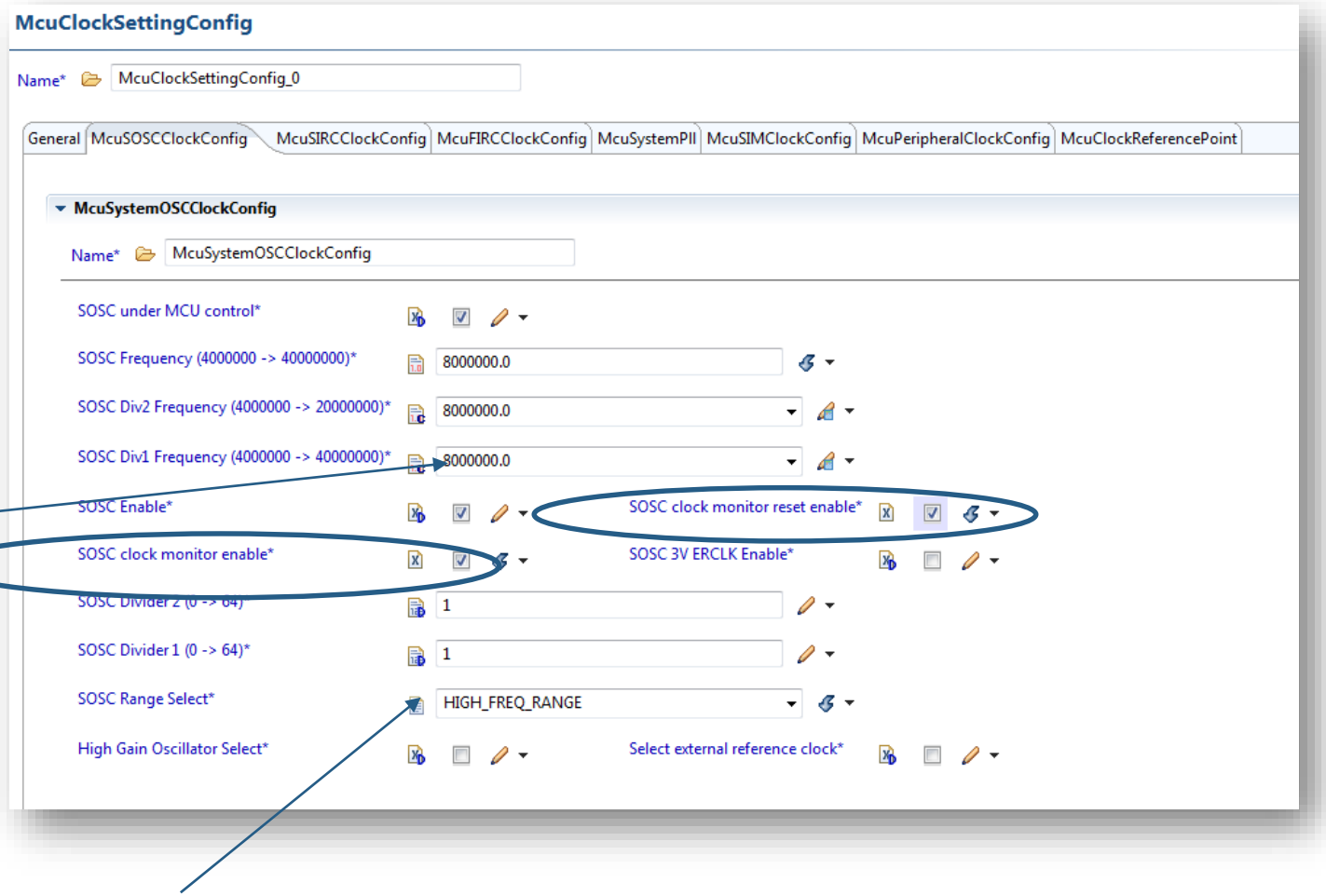
- 通过Mcu_InitClock接口配置所有时钟源，如系统时钟，外设时钟，时钟监控单元，PLL等
- 所有（预）分频器和时钟源都可配置，频率自动计算，通过ClockReferencePoint导出给其他模块使用

- 时钟源导出给其他模块使用，比如:
  - CAN 模块使用Mcu ClockReferencePoint 来计算波特率
  - GPT 模块使用Mcu_ClockReferencePoints来处理每个硬件计时器的频率

MCU晶振配置

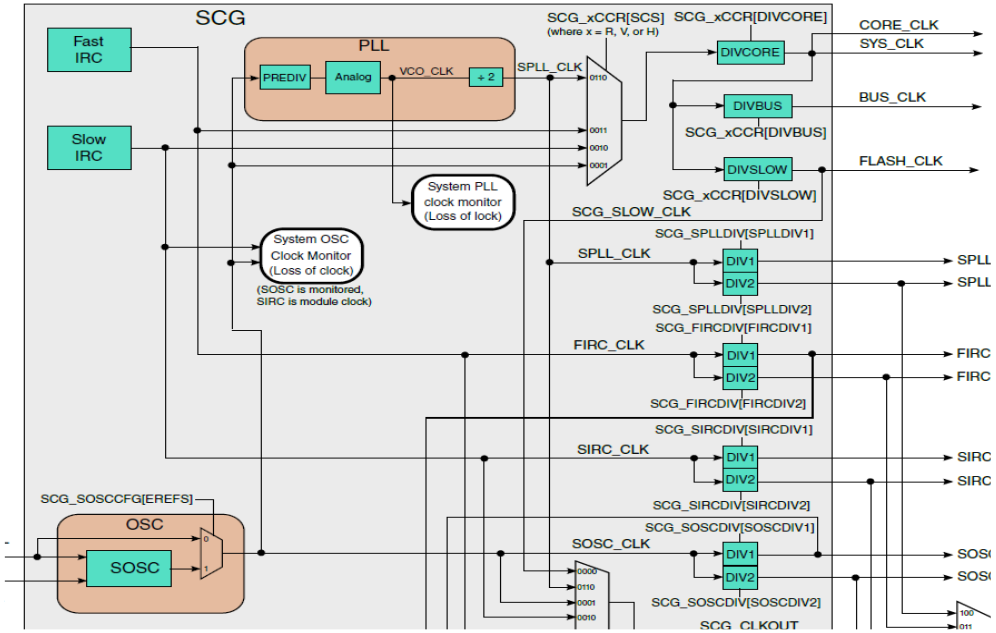


选择合适的晶振监控源



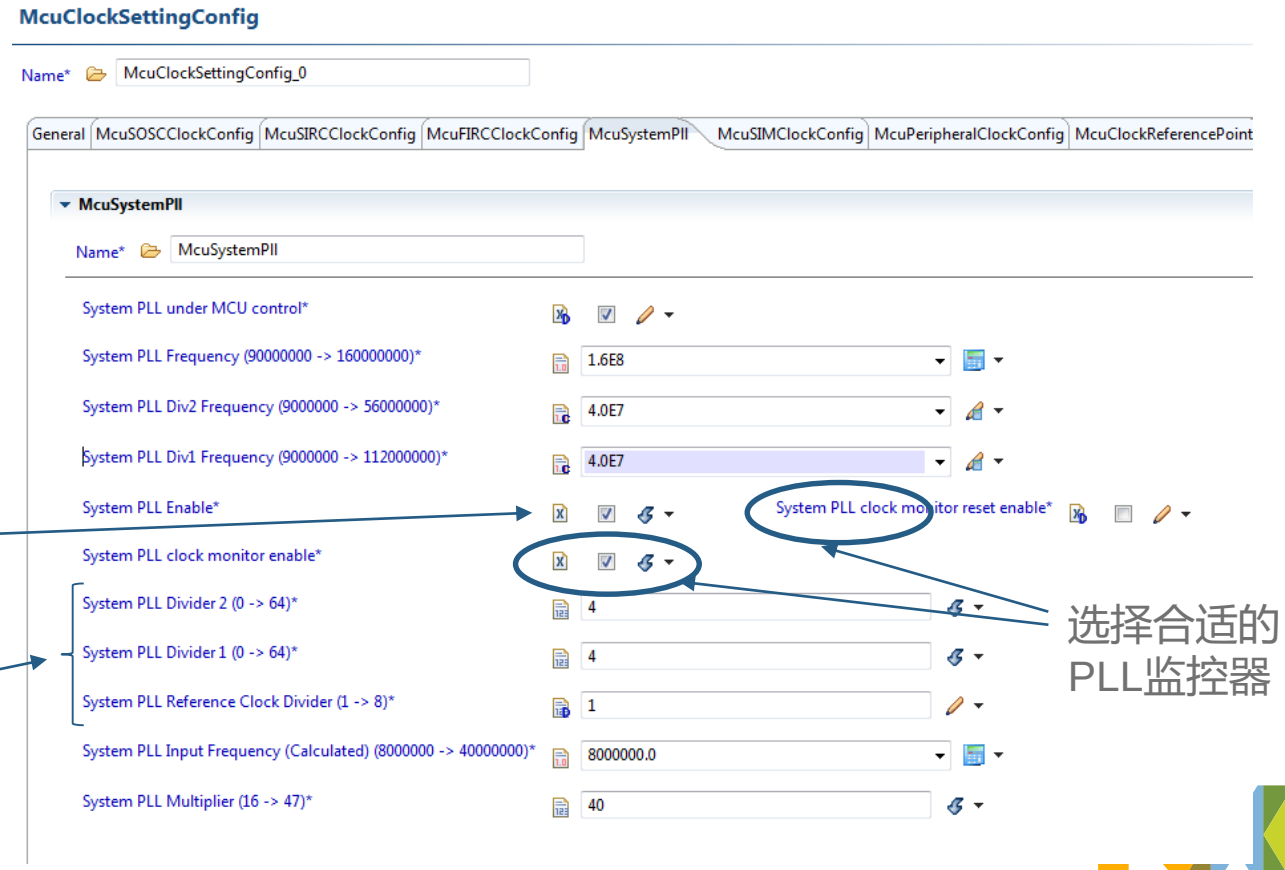
中等频率范围：选择4Mhz 到 8 Mhz。
高频率范围：选择8 Mhz 到 40 Mhz。

MCU PLL配置



使能PLL

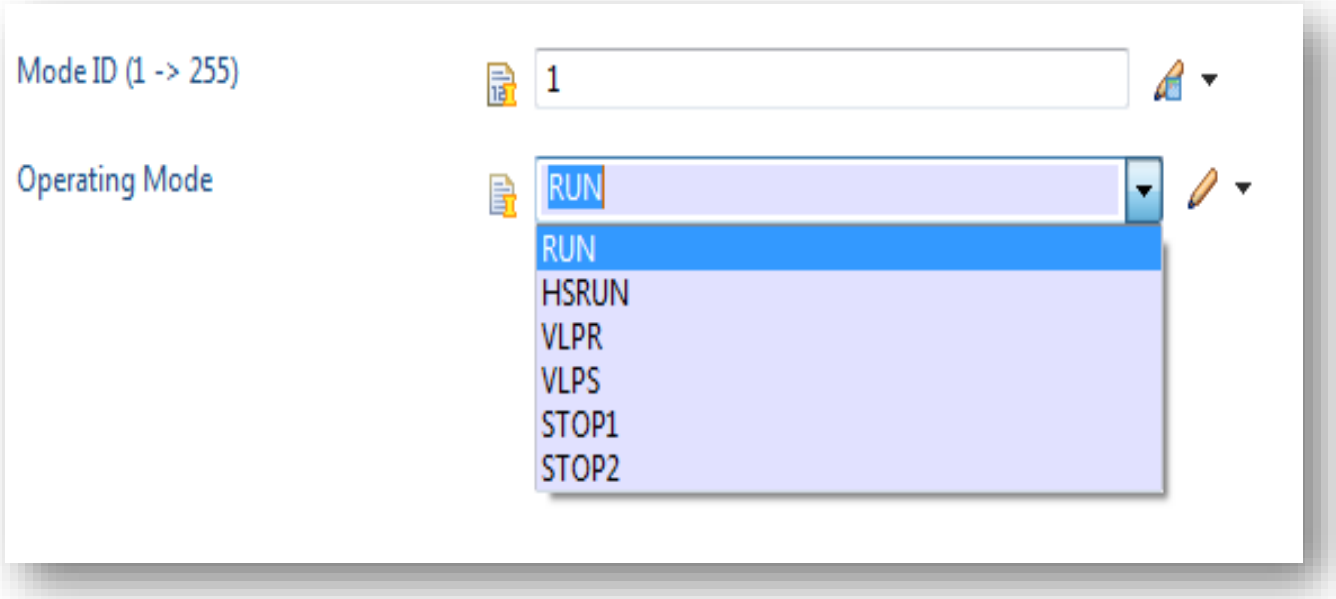
设置分频器参数
需要注意主频最大限制



选择合适的
PLL监控器

MCU 电源模式

使用*Mcu_SetMode*函数，可以将系统从正常运行模块切换到低功耗模式，但是这里不负责从低功耗模式唤醒。



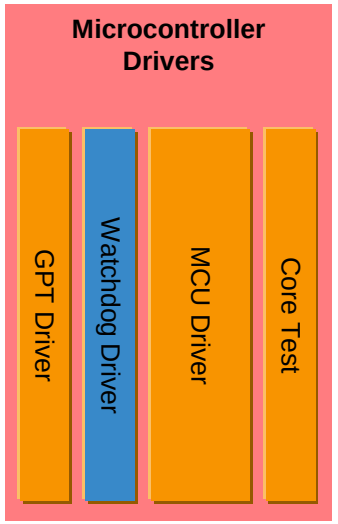
看门狗
(WDG)

WDG

初始化及控制看门狗硬件单元。

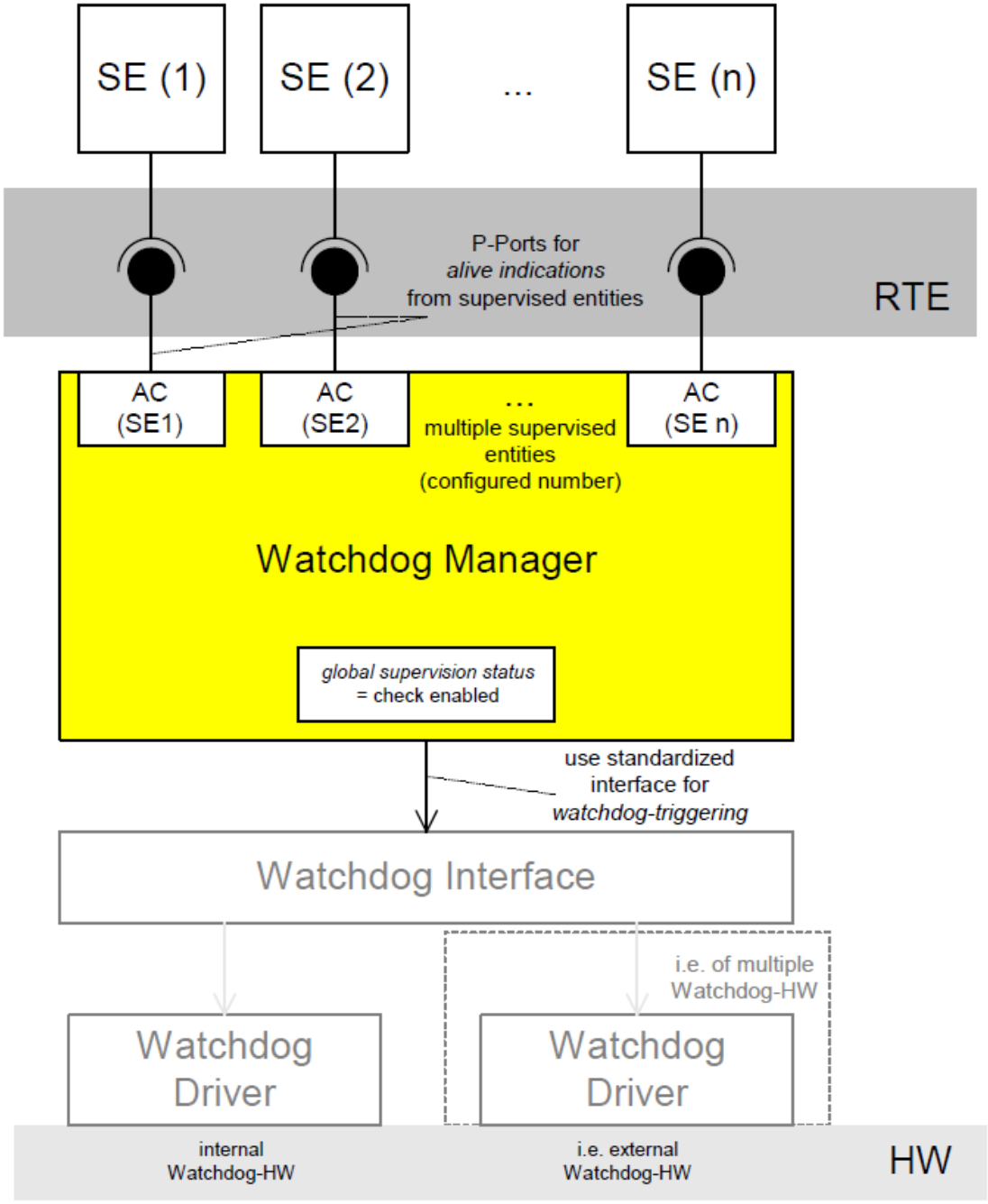
该模块提供以下功能及参数配置：

- ✓ 初始化
- ✓ 改变操作模式
- ✓ 设置触发条件



WDG 上层

- WdgIf
  - 管理多个硬件看门狗（如内部看门狗，外部看门狗）
- WdgManager
  - 程序流监控
  - 监控多个SWC或者说有多个SE(supervised entities)。每个应用会持续向监控模块报告当前状态，只有当所有应用都保持在线才会认为系统正常，才不会触发看门狗



其他桩模块

桩模块

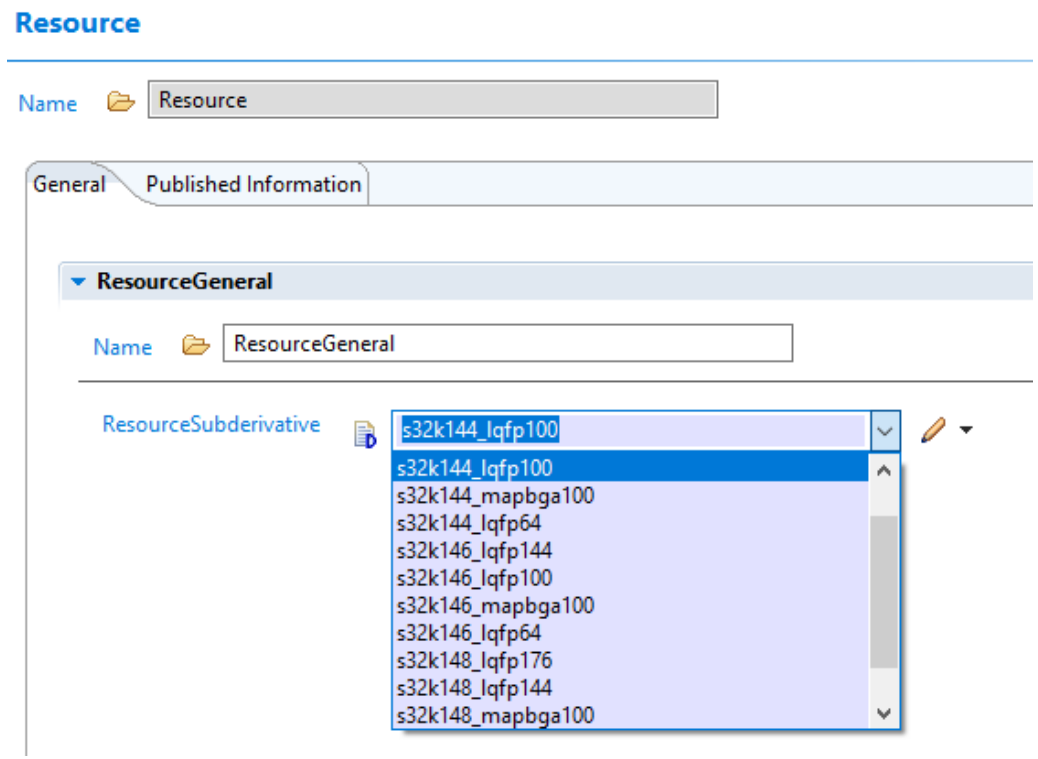
这里的桩模块指一些空模块，常用于占位使用，为了解决由于该模块缺失导致的编译或运行问题；或者一些不能被模块化的基础服务，如类型定义等。

主要包含以下模块：

Dem,Det,WdgIf,MemIf,Ea,Fee,EcuM,Base

Resource

该模块不包含任何执行代码， EB Tresos Studio 用来选择多个不同版本配置（不同芯片型号）。



微处理器库（MCL）

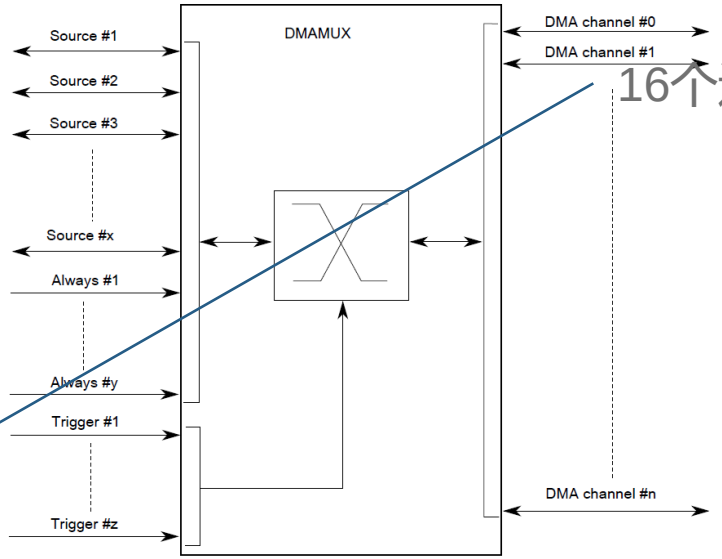
MCL 功能列表

MCL

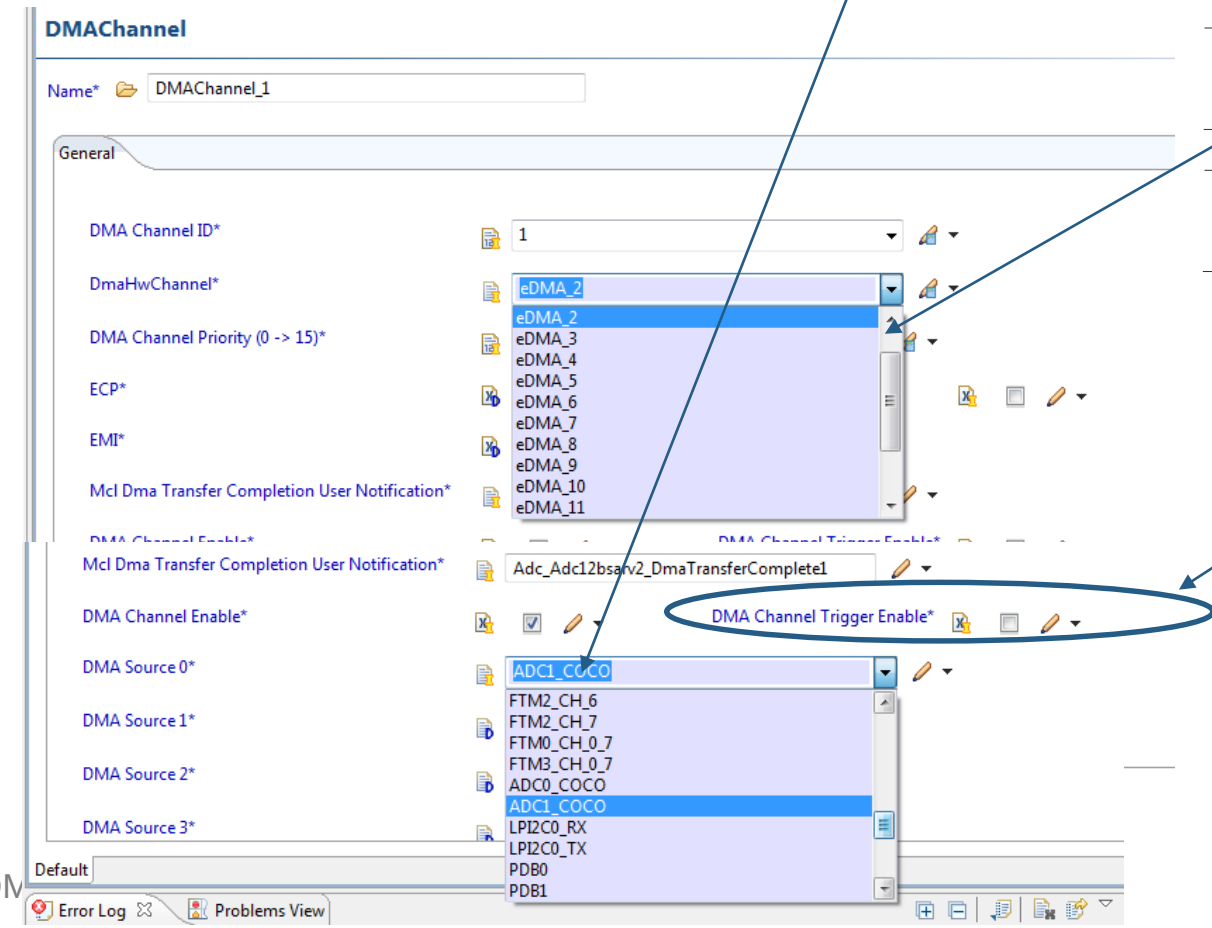
- ✓ DMA
- ✓ DMAMUX
- ✓ TRGMUX
- ✓ Crossbar(AXBS, AXBS lite)
- ✓ Logical Address (支持系统使用MMU)
- ✓ FLEXIO
- ✓ FTM, eTimer, eMIOS 基础功能
- ✓ Etc…

DMA 配置

选择DMA源



16个通道中选择一个



前4个通道可选择为周期触发

MCL CACHE

MCL 提供支持多种CACH操作。

内部硬件外设: LMEM。

MCL 提供以下服务：

- ✓打开/关闭CACHE
- ✓基于实例的刷新/使无效/清空
- ✓基于地址的刷新/使无效/清空

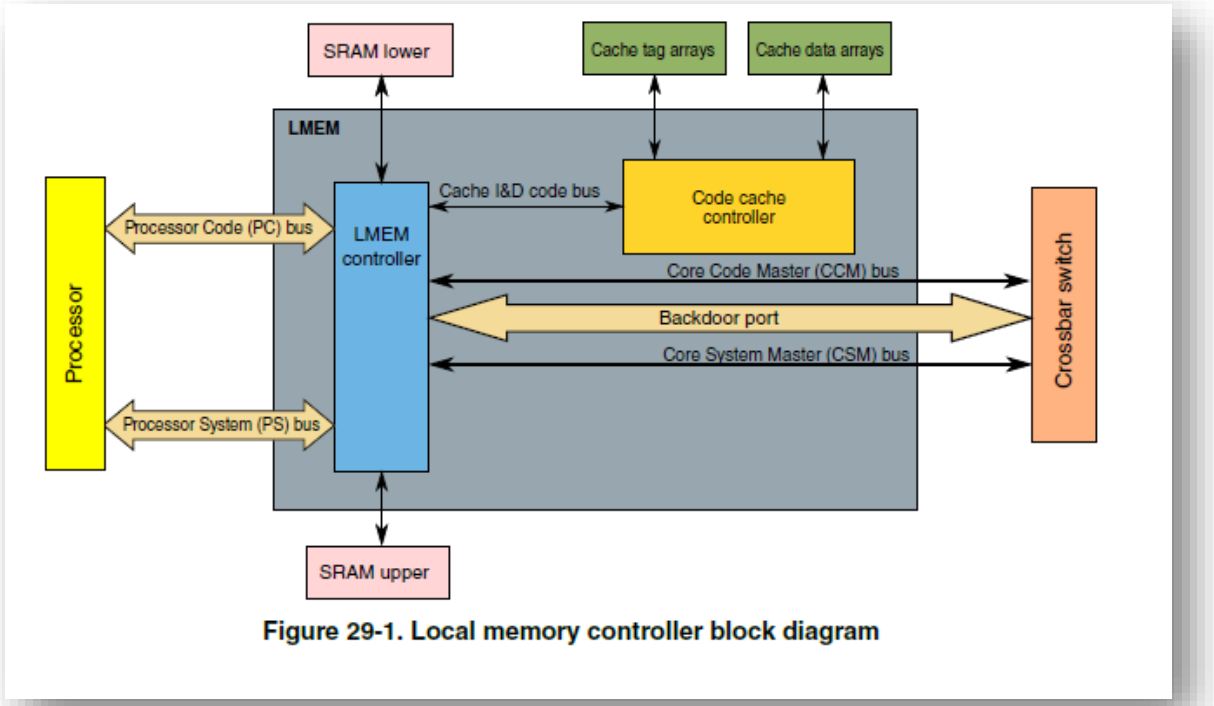
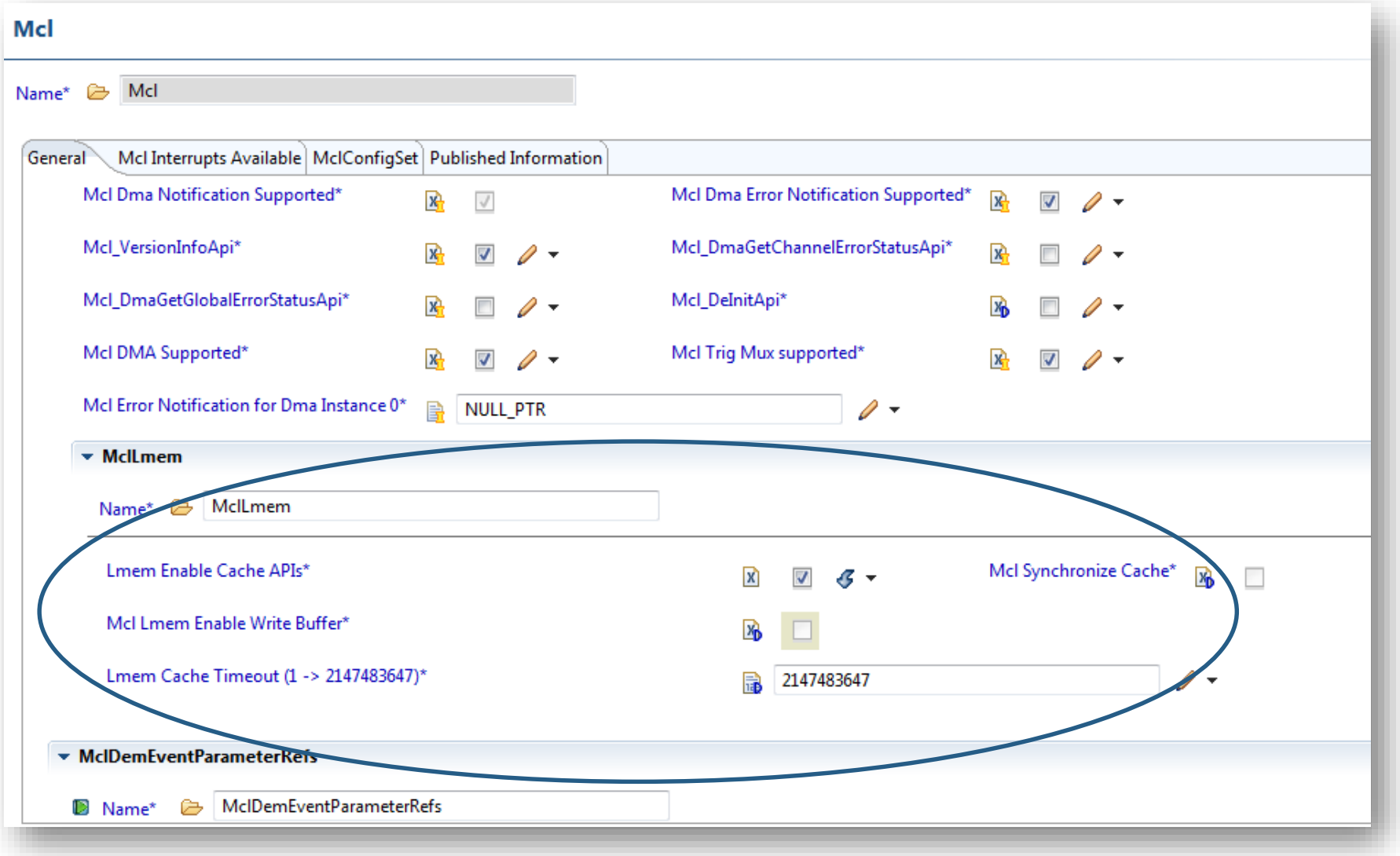


Figure 29-1. Local memory controller block diagram

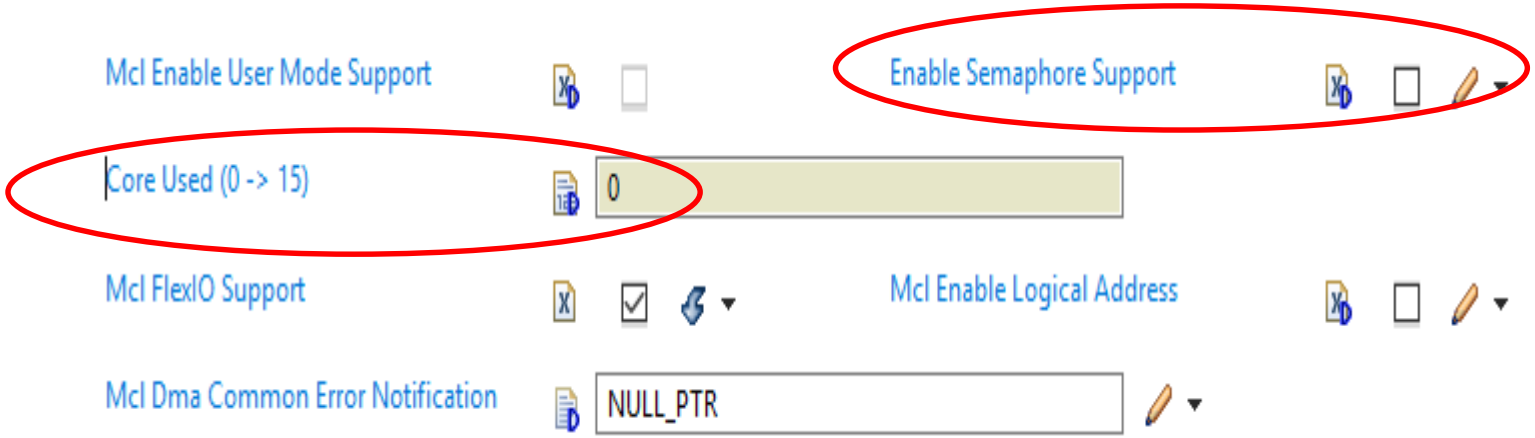
MCL CACHE配置界面



MCL 信号量

在多线程/多核系统里面，多个系统软件经常共享数据结构，共享硬件单元，因此需要一个可安全使用且友好的互锁机制。

要使用该功能必须首先在配置界面使能，然后指定在哪个内核上工作，如下图：



总结

Q&A

SECURE CONNECTIONS
FOR A SMARTER WORLD