FTF FREESCALE TECHNOLOGY FORUM 2015

Hands-On Workshop: AUTOSAR Training (Reserved Seat Required)

FTF-ACC-F1243

Marius Rotaru | Technical Leader - Automotive Software

J U N E . 2 0 1 5

External Use

Freescale, the Freescale logo, AltiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Magniv, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Qorivva, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid and Xtrinsic are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Airfast, BeeKit, BeeStack, CoreNet, Flexis, Layerscape, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink and UMEMS are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. © 2015 Freescale Semiconductor, Inc.

Agenda

- AUTOSAR Motivation and Principles
  - Vision and Objectives
  - Development Cooperation
  - Architecture of the Standard
  - Migration of the Standard
- AUTOSAR Configuration Methodology & Tools
- AUTOSAR MCAL
- AUTOSAR OS
- Examples: Hands-on Training
  - LAB1: Blinking LED
  - LAB2: Dimming LED

AUTOSAR Motivation and Principles

Embedded Software

Mars Curiosity Rover
5MLoC

Android
11.8 MLoC

F-35 Joint Strike Fighter
23.5 MLoC

Mercedes S Class
~**100MLoC**

There is A LOT of Embedded Software in **Automotive**!

Source:

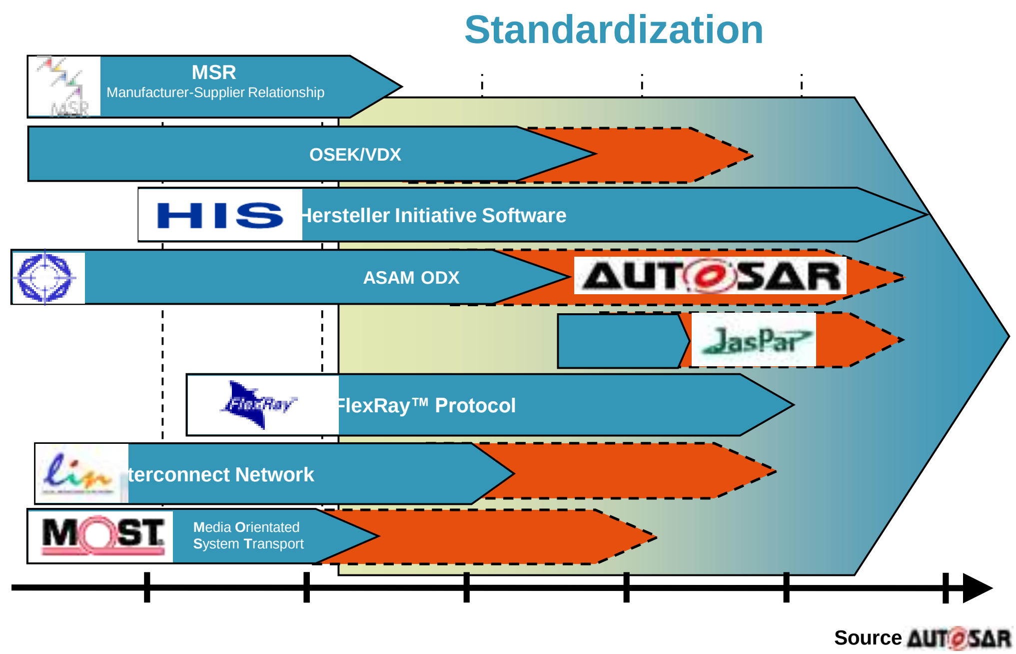
http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code
http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

AUTOSAR Standardization

Technology partnerships and open standards encouraging “plug-and-play” approach

Freescale, a reliable partner for automotive software and hardware innovation:

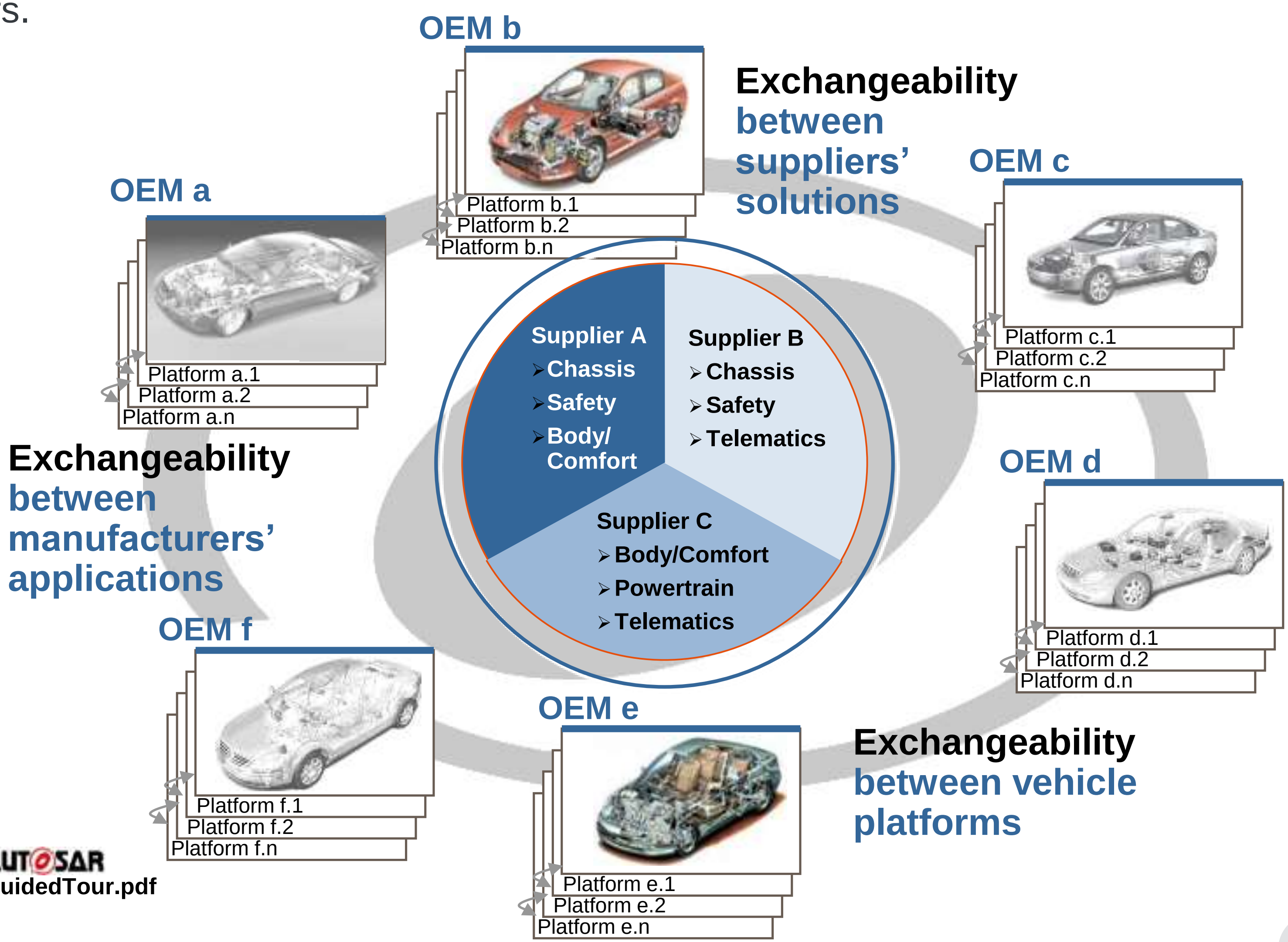
- Driving member of the **OSEK/VDX™** consortium, with own operating system implementation
- Founding member of the **LIN™** consortium
- Founding member of **FLEXRAY™** partnership
- First semiconductor vendor to join **AUTOSAR™** partnership



(AUTomotive Open System ARchitecture)

AUTOSAR Vision

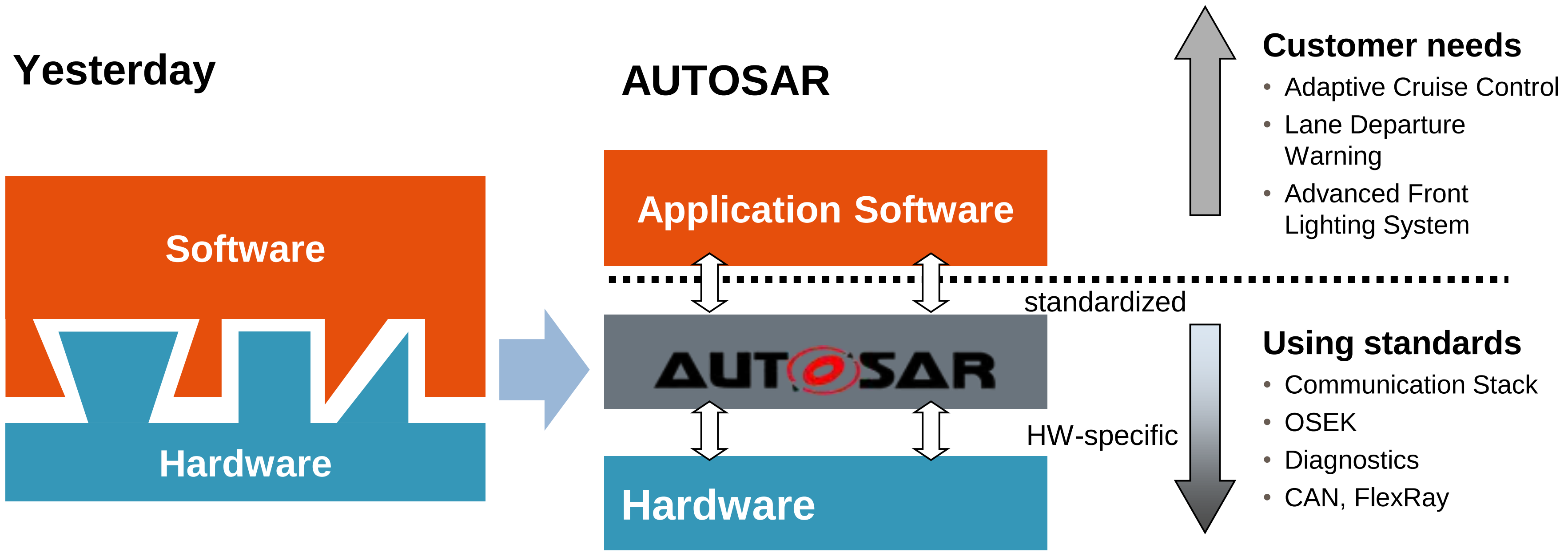
AUTOSAR aims to improve the complexity management of integrated E/E architectures through increased reuse and exchangeability of software modules between OEMs and suppliers.



Source: AUTOSAR Autosar_GuidedTour.pdf

AUTOSAR Vision

AUTOSAR aims to standardize the software architecture of ECUs. AUTOSAR paves the way for innovative electronic systems that further improve performance, safety and environmental friendliness.



- Hardware and software will be widely independent of each other
- Development can be de-coupled by horizontal layers. This reduces development time and costs
- The reuse of software increases at OEM as well as at suppliers. This enhances quality and efficiency

Source: AUTOSAR Autosar_GuidedTour.pdf

AUTOSAR Objectives

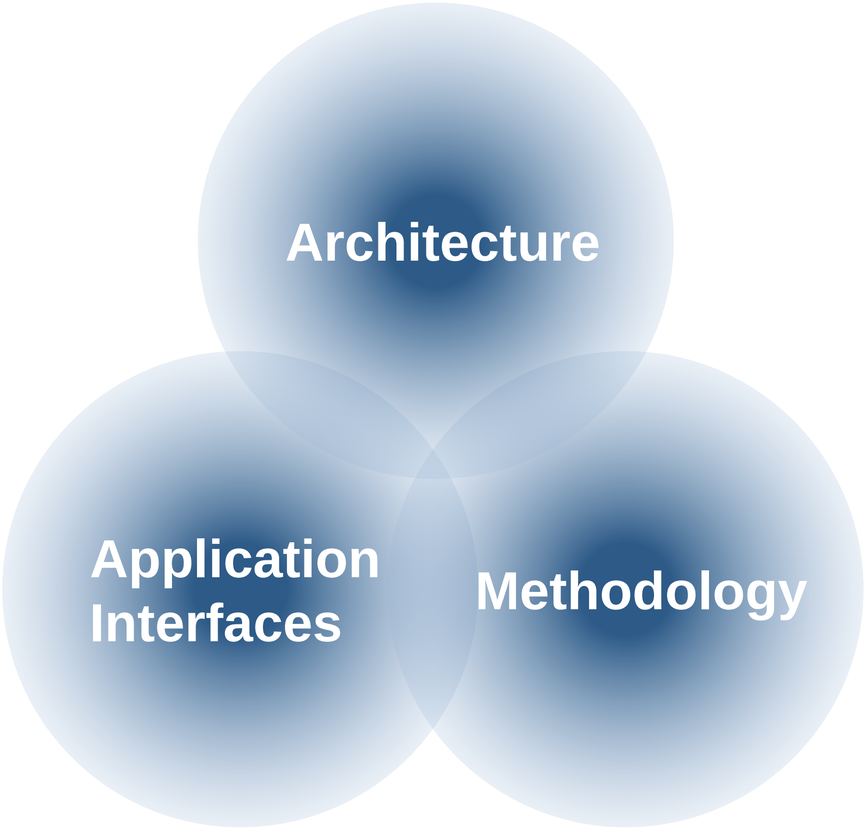
PO1:
Transferability of software

PO2:
Scalability to different vehicle and platform variants

PO3:
Different functional domains

PO4:
Definition of an open architecture

PO5:
Dependable systems



PO6:
Sustainable utilization of natural resources

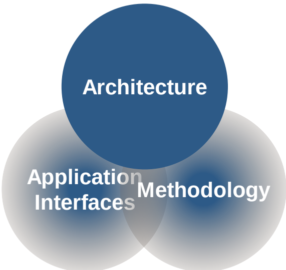
PO7:
Collaboration between various partners

PO8:
Standardization of basic software functionality of automotive ECUs

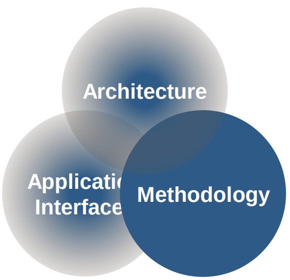
PO9:
Applicable automotive international standards and state-of-the-art technologies

Source AUTOSAR RS_ProjectObjectives.pdf
Autosar_GuidedTour.pdf

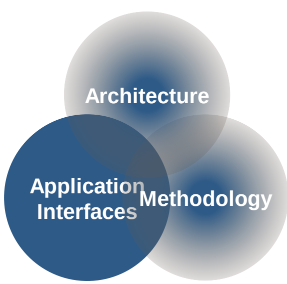
AUTOSAR Main Working Topics



- **Architecture:**
  Software architecture including a complete basic software stack for ECUs — the so called AUTOSAR Basic Software — as an integration platform for hardware independent software applications.



- **Methodology:**
  Defines exchange formats and description templates to enable a seamless configuration process of the basic software stack and the integration of application software in ECUs. It includes even the methodology how to use this framework.



- **Application Interfaces:**
  Specification of interfaces of typical automotive applications from all domains in terms of syntax and semantics, which should serve as a standard for application software.

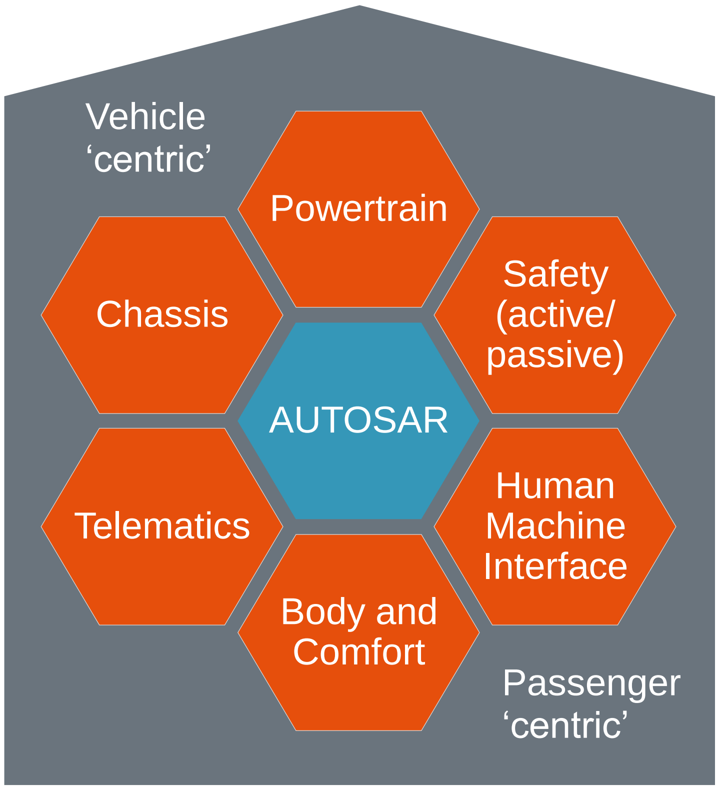
Source: AUTOSAR
Autosar_GuidedTour.pdf

AUTOSAR — Concept and Functional Domain

The AUTOSAR project objectives will be met by specifying and standardizing the central architectural elements across functional domains, allowing industry competition to focus on implementation.

Cooperate on standards, compete on implementation.

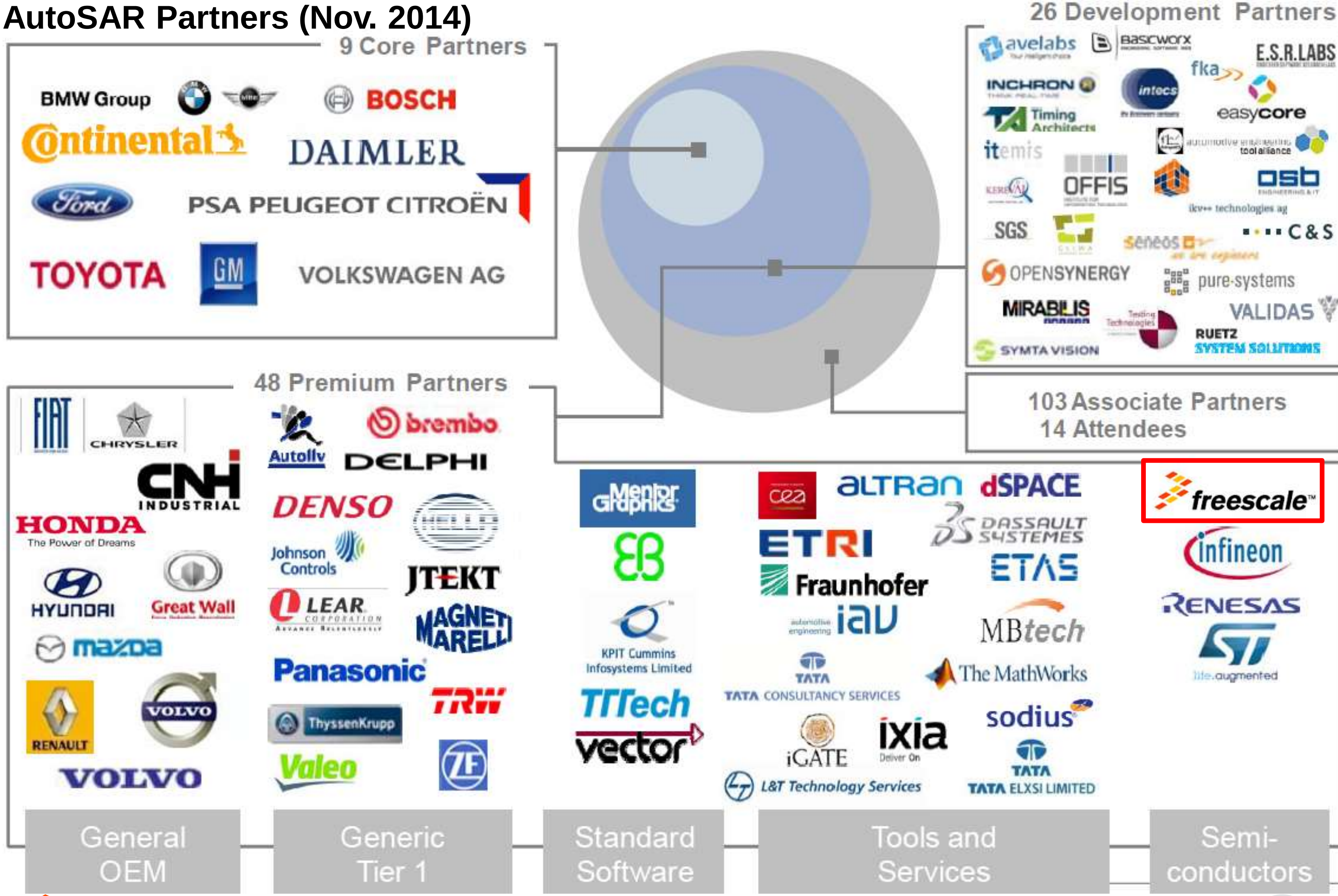
Functional Domains



Source: AUTOSAR
Autosar_GuidedTour.pdf

AUTOSAR — Cooperation Structure and Partners

AutoSAR Partners (Nov. 2014)



Source: AUTOSAR us at http://www.AUTOSAR.org

Basic AUTOSAR Approach

Virtual Integration

Independent of hardware

Virtual Functional Bus.

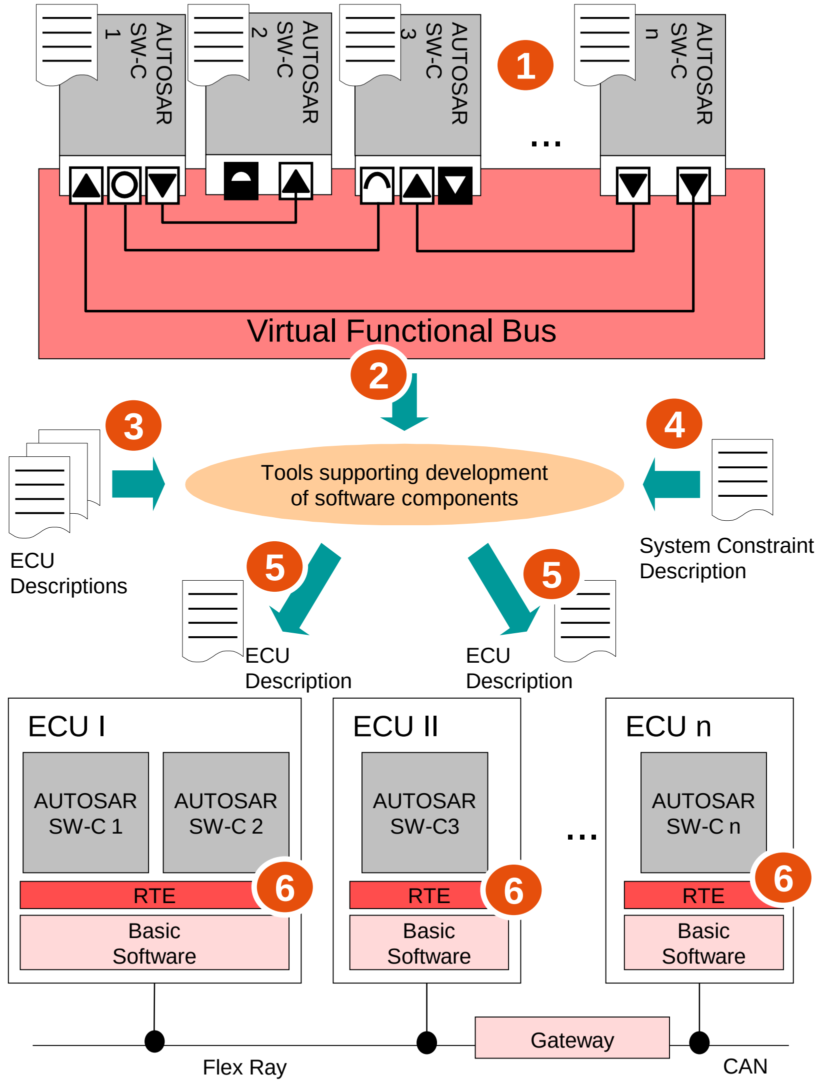
Introduction of Hardware Attributes

Holistic view of the entire system, both software and hardware.

ECU Configuration

Run-Time Environment

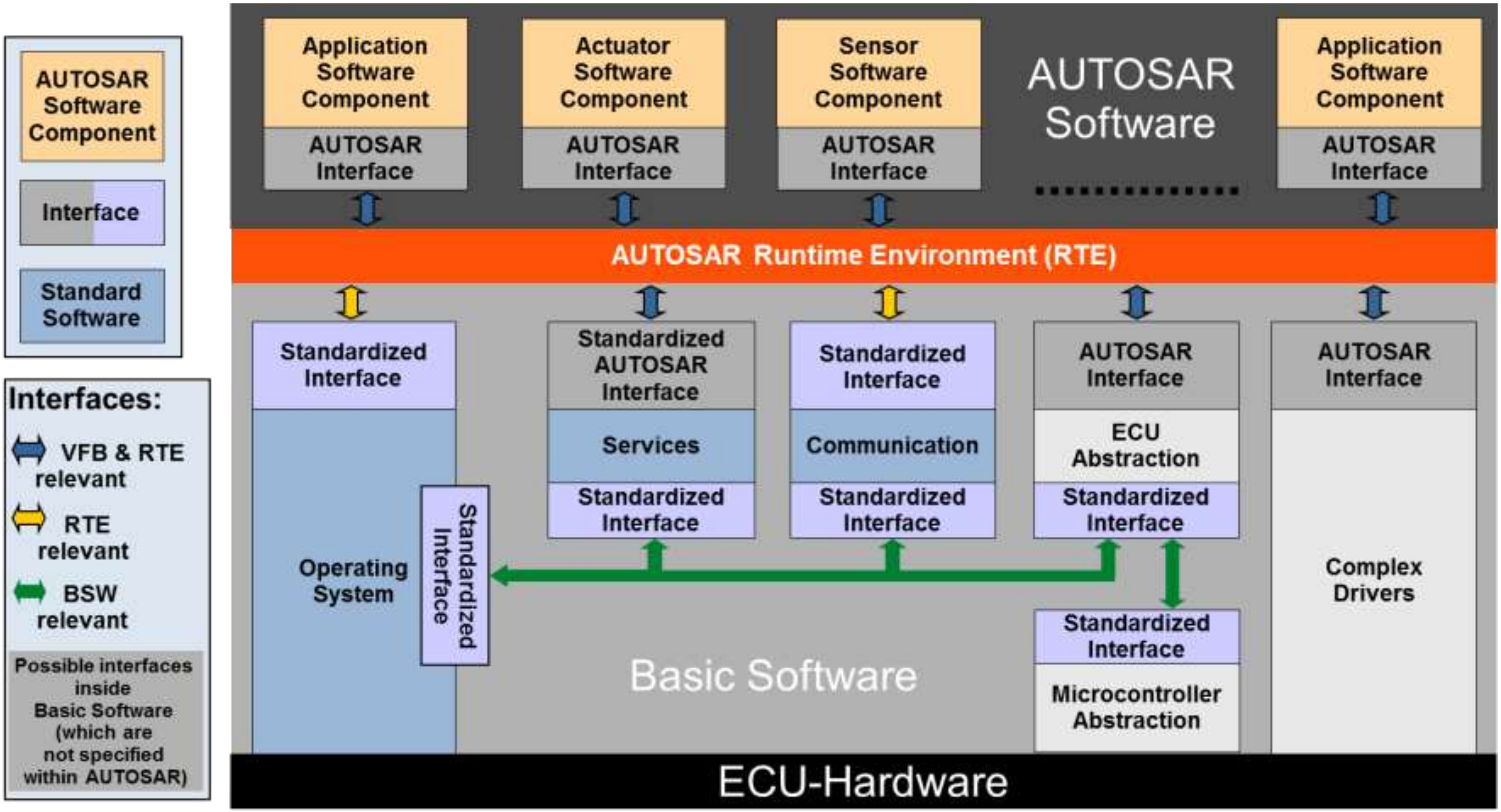
Separation of system into its ECU (plus common infrastructure).



1. Software Component (SW-C) description
2. Integration of SW-C via Virtual Functional Bus (VFB)
3. ECU Description
4. System Constraints
5. Mapping of SW-C on specific ECU
6. Configuration of Basic Software Modules (BSW) and Run-Time Environment (RTE)

Source: AUTOSAR

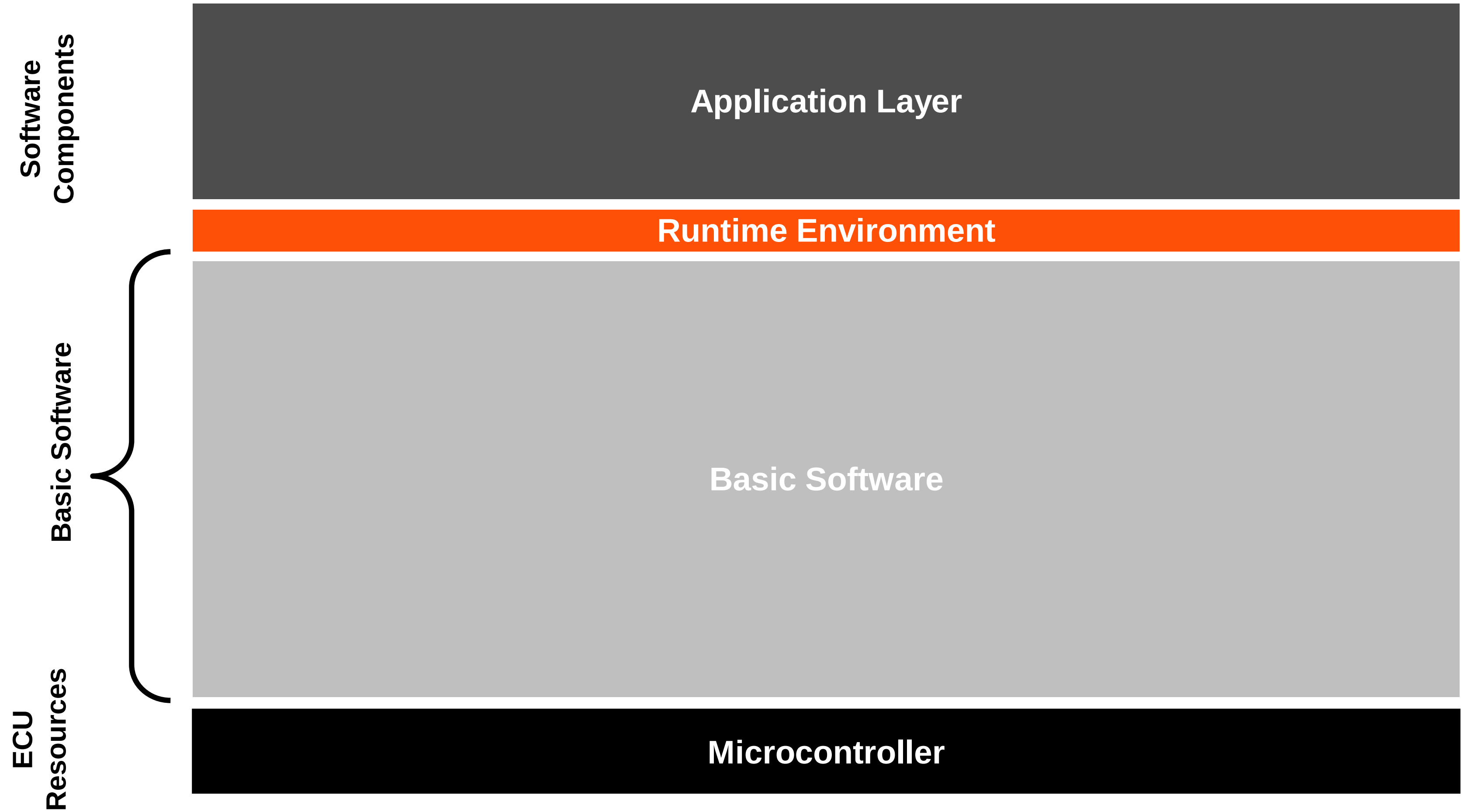
AUTOSAR Architecture — Components and Interface View



Source: AUTOSAR

AUTOSAR Layered Software Architecture

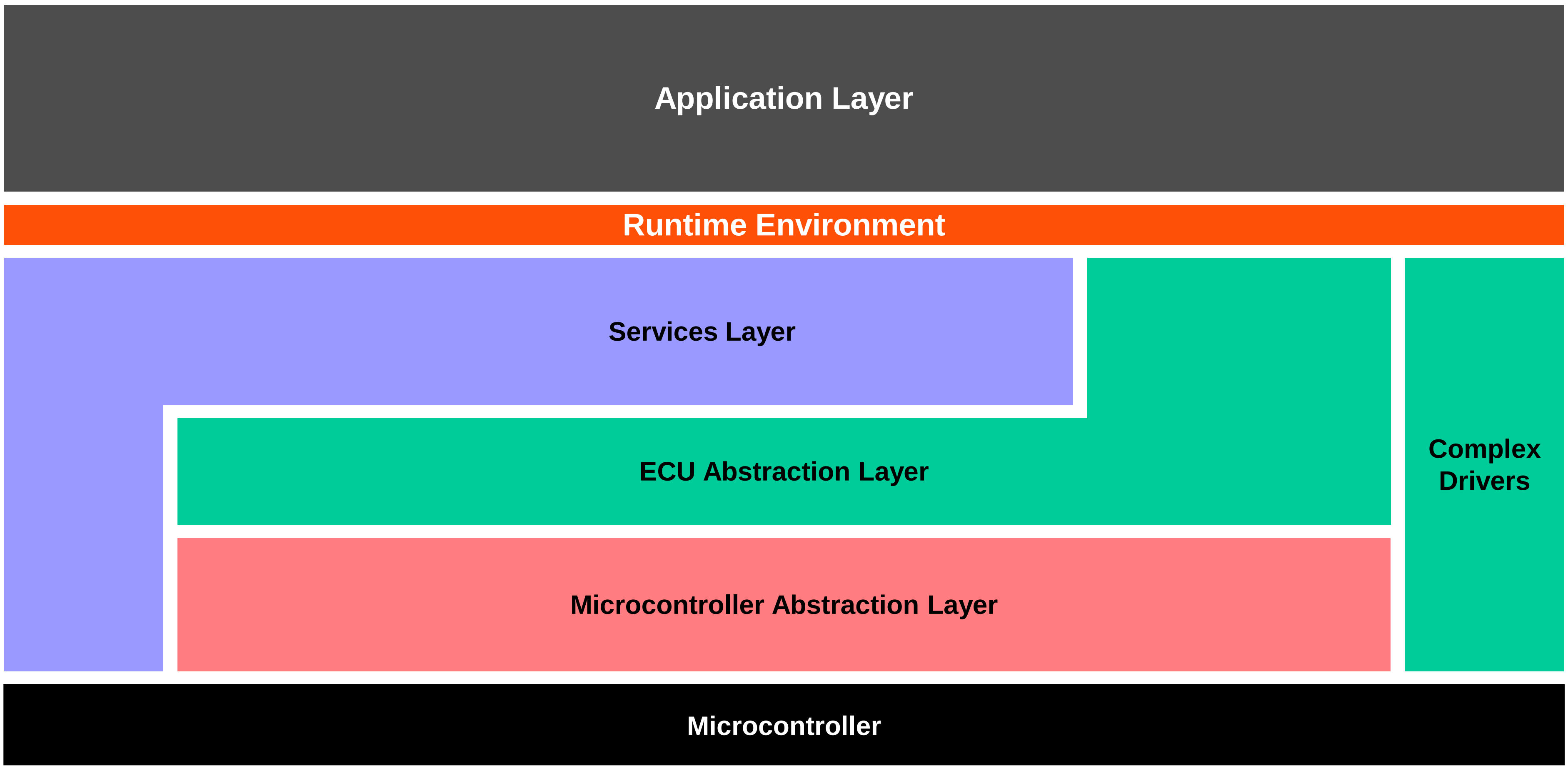
Basic structure distinguishes four basic layers.



Source: AUTOSAR

AUTOSAR Layered Architecture

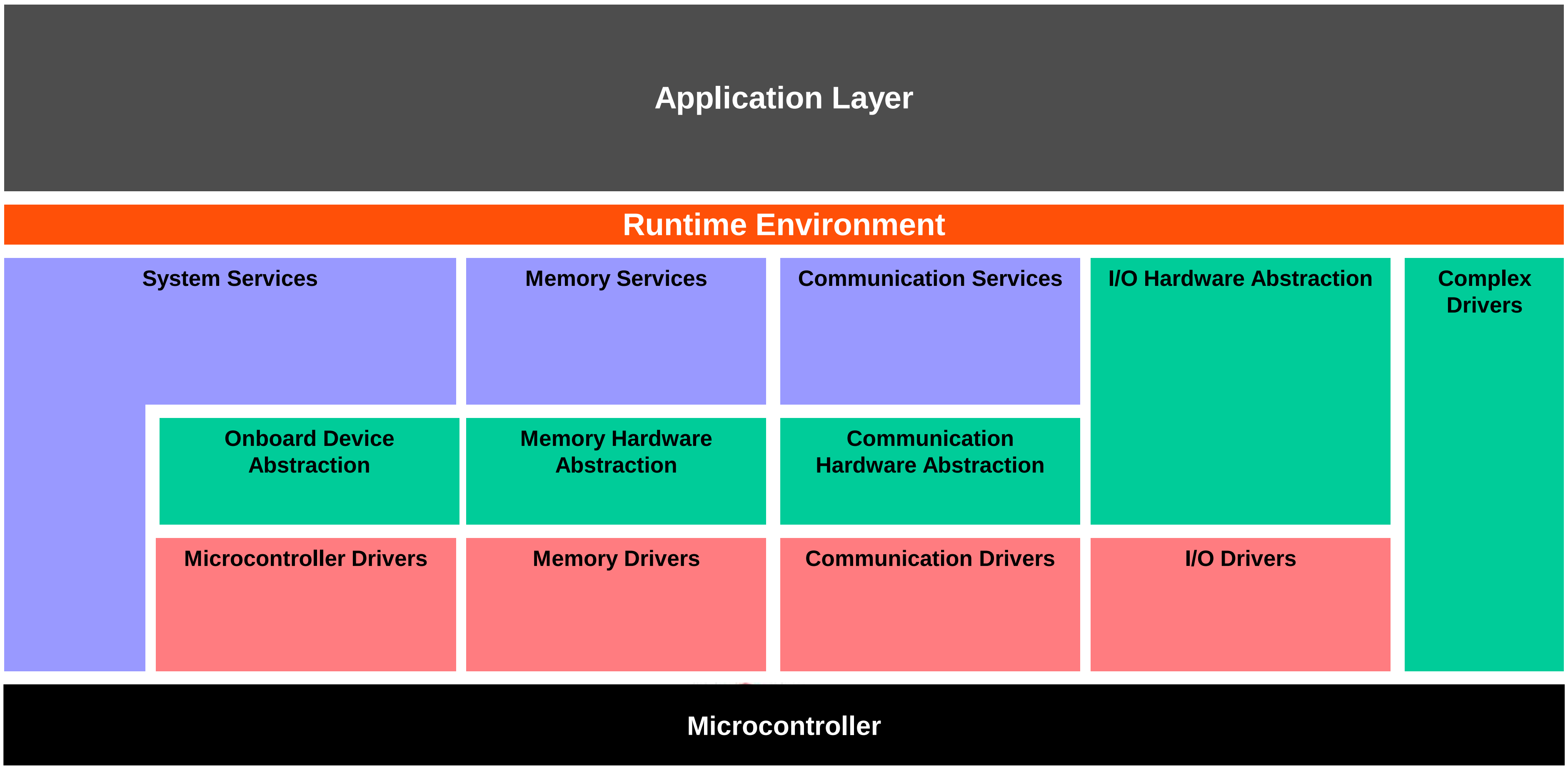
The **AUTOSAR Basic Software** is further divided in the layers: Services, ECU Abstraction, Microcontroller Abstraction and Complex Drivers**.**



Source: AUTOSAR

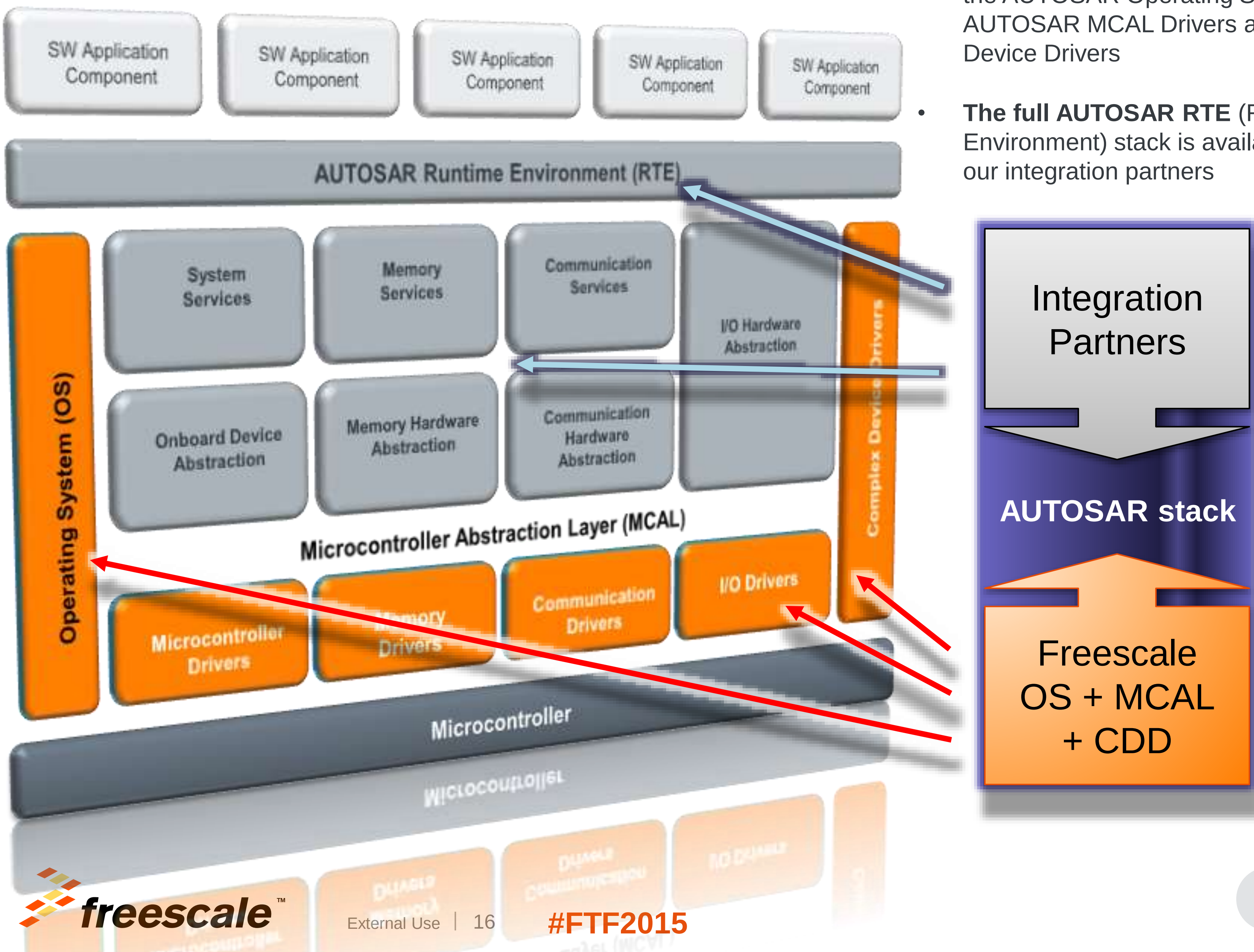
AUTOSAR Layered Architecture

The **Basic Software Layers** are further divided into functional groups. Examples of Services are System, Memory and Communication Services.



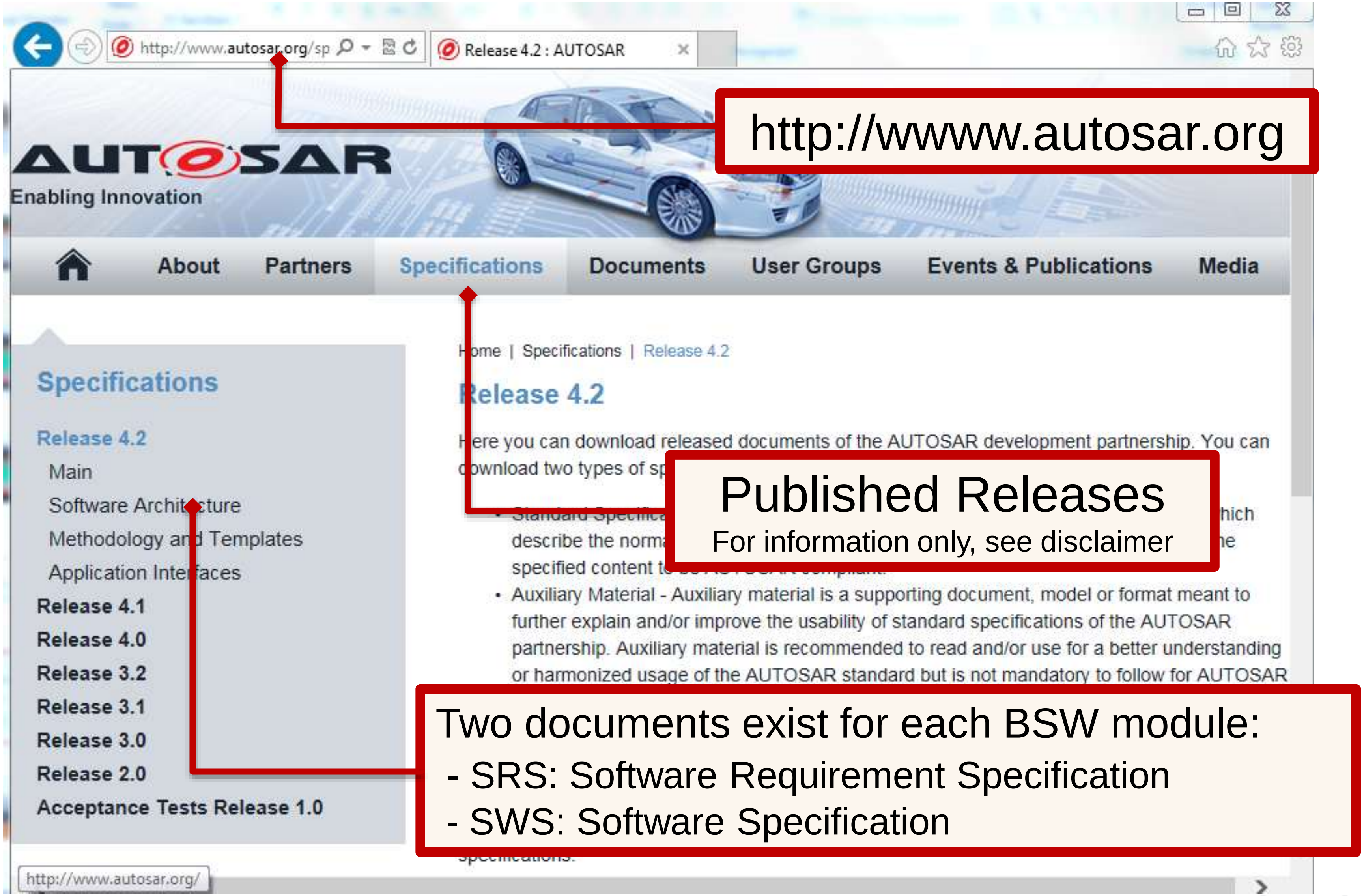
Source: AUTOSAR

AUTOSAR Freescale Solution

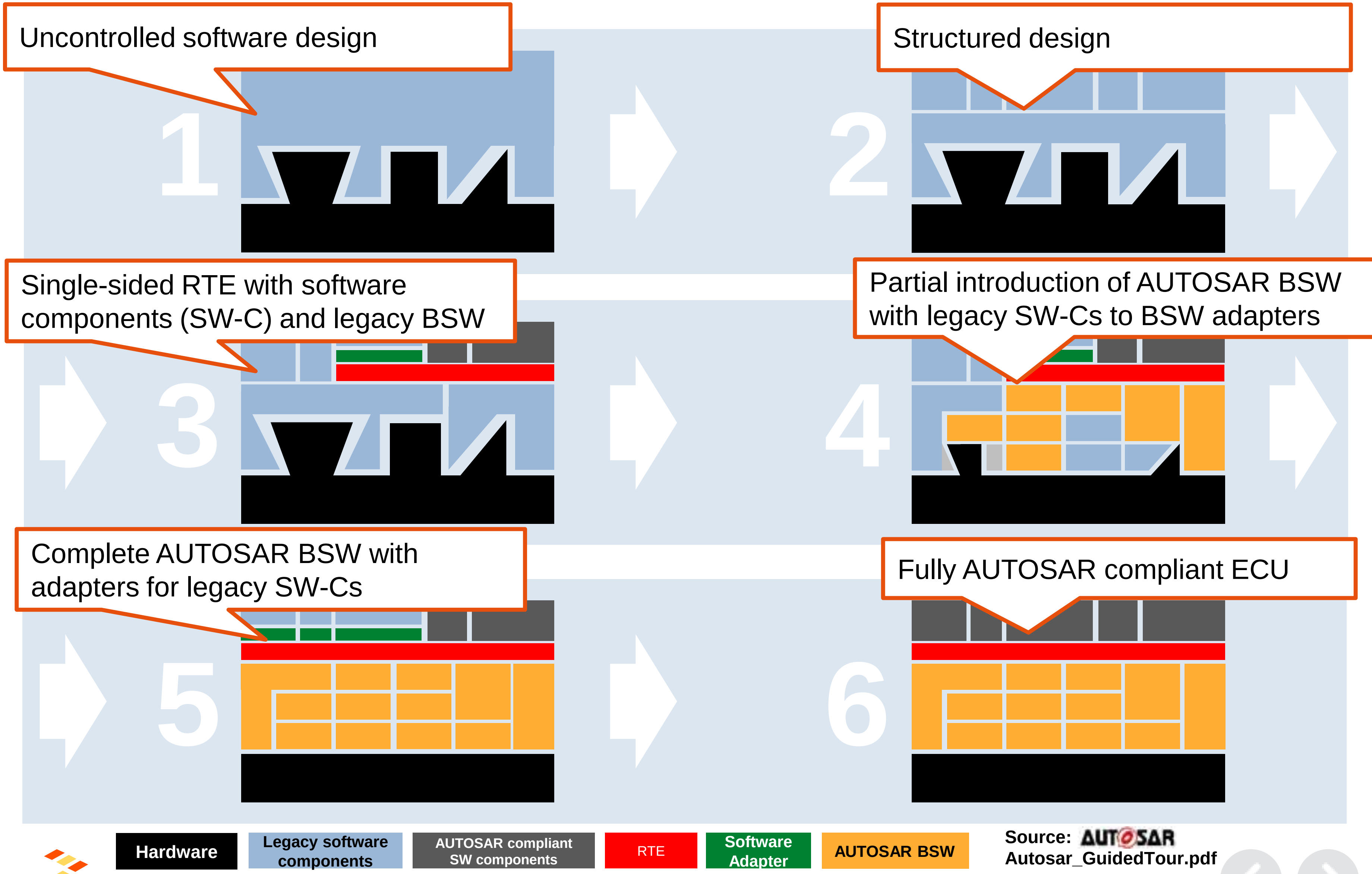


- **Freescale Software Products** include the AUTOSAR Operating System, AUTOSAR MCAL Drivers and Complex Device Drivers
- **The full AUTOSAR RTE** (Runtime Environment) stack is available through our integration partners

AUTOSAR Documents



AUTOSAR — Application Migration



Source: AUTOSAR Autosar_GuidedTour.pdf

AUTOSAR Configuration Methodology and Tools

Basic Software Configuration Process

Freescale AUTOSAR Integration Partners receive Freescale MCAL and OS releases for pre-integration into their proprietary AUTOSAR BSW products.

EB

vector

Mecel

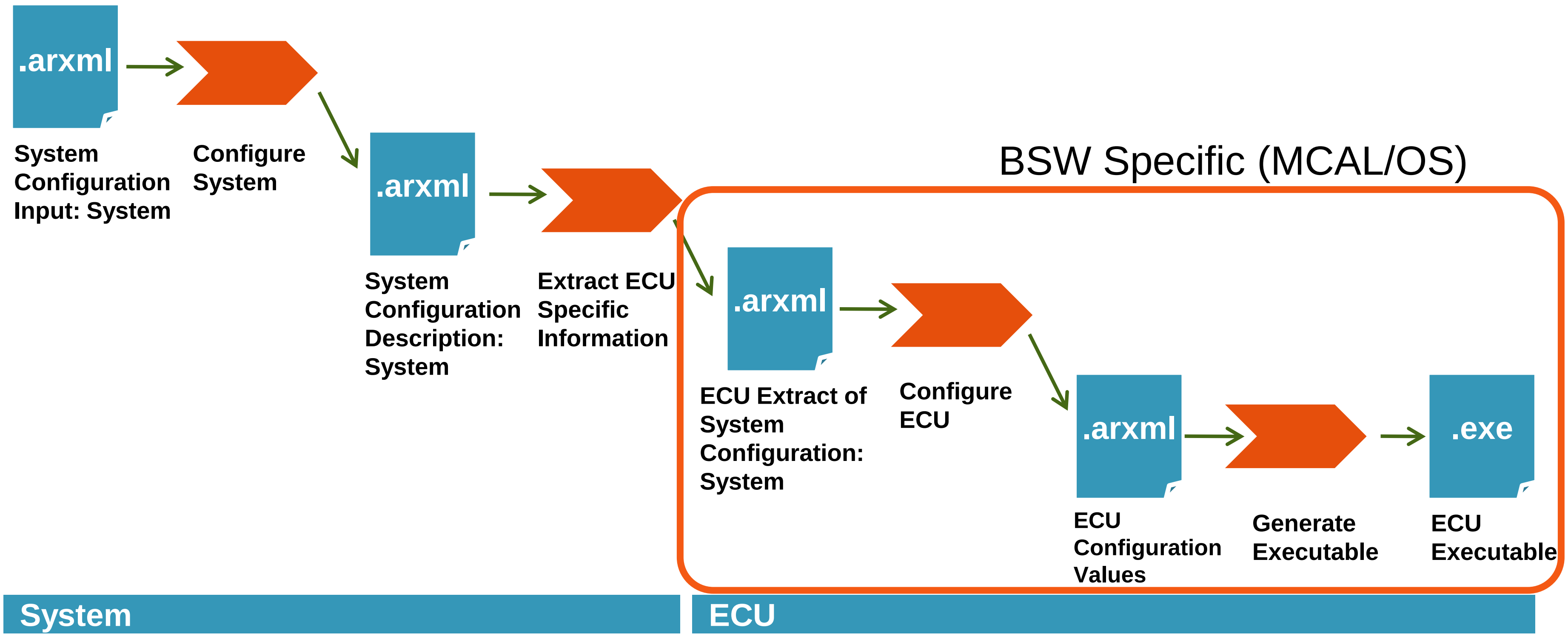
ETAS

Mentor Graphics

KPIT Cummins

see4sys

AUTOSAR Methodology and Templates — Waterfall View



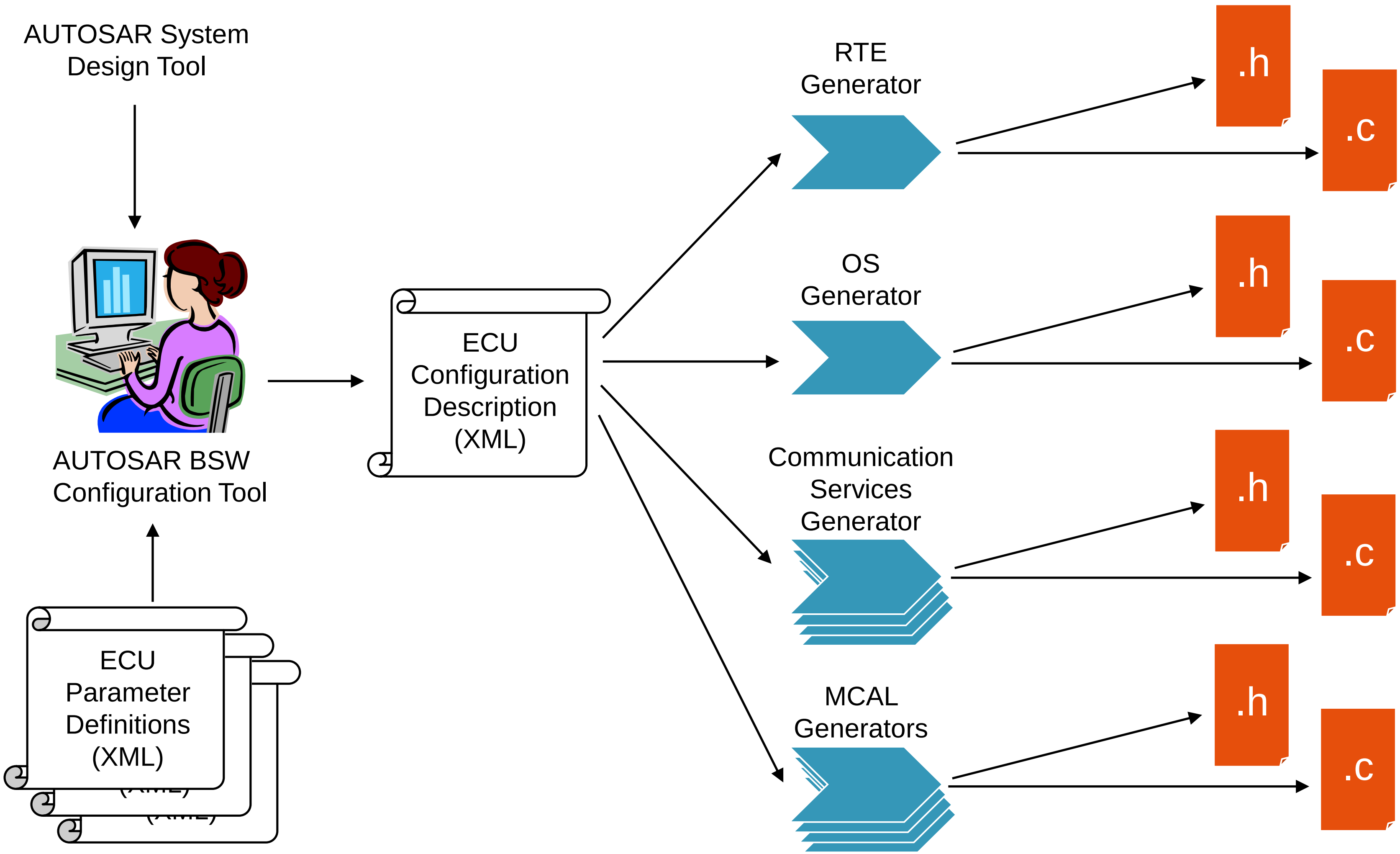
- The AUTOSAR Methodology is foreseen to support activities, descriptions and use of tools in AUTOSAR
  - The notation of the Software Process Engineering meta-model (SPEM) is used
- The AUTOSAR methodology is not a complete process description but rather a common technical approach for some steps of system development
- Outside the scope of the AUTOSAR standard is:
  - Description of tools (which add value to the ‘Activities’ in the methodology)
  - Definition of roles and responsibilities

Software Module Static/Generated Parts

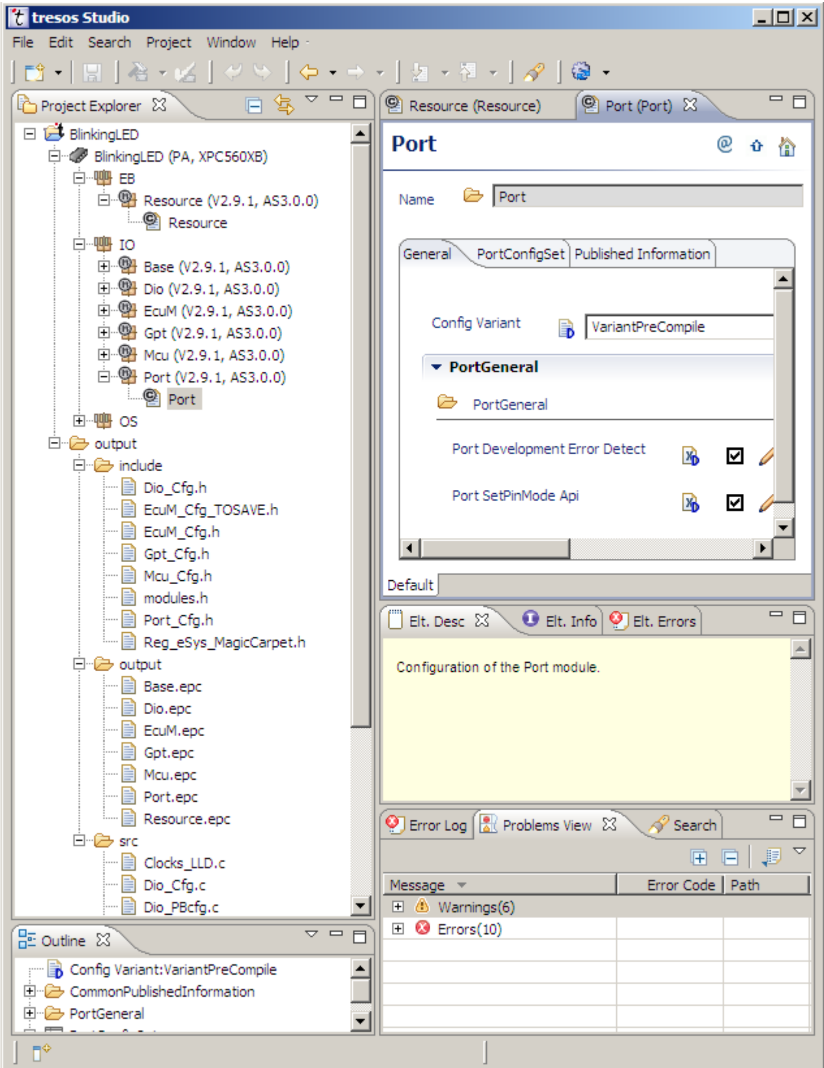
One AUTOSAR BSW module normally consists of three main pieces:

- Software module **source code**:
  - it is a static part of software module, which is not ECU configuration dependent
- Software module **VSMD** (Vendor Specific Module Definition):
  - an XML file that describes software module configuration capabilities (EPD)
- Software module **generator**:
  - process ECU configuration (also an XML file but different to VSMD) (EPC) and generates software module(s)

Basic Software Configuration Process



ElektroBit (EB) Tresos Studio

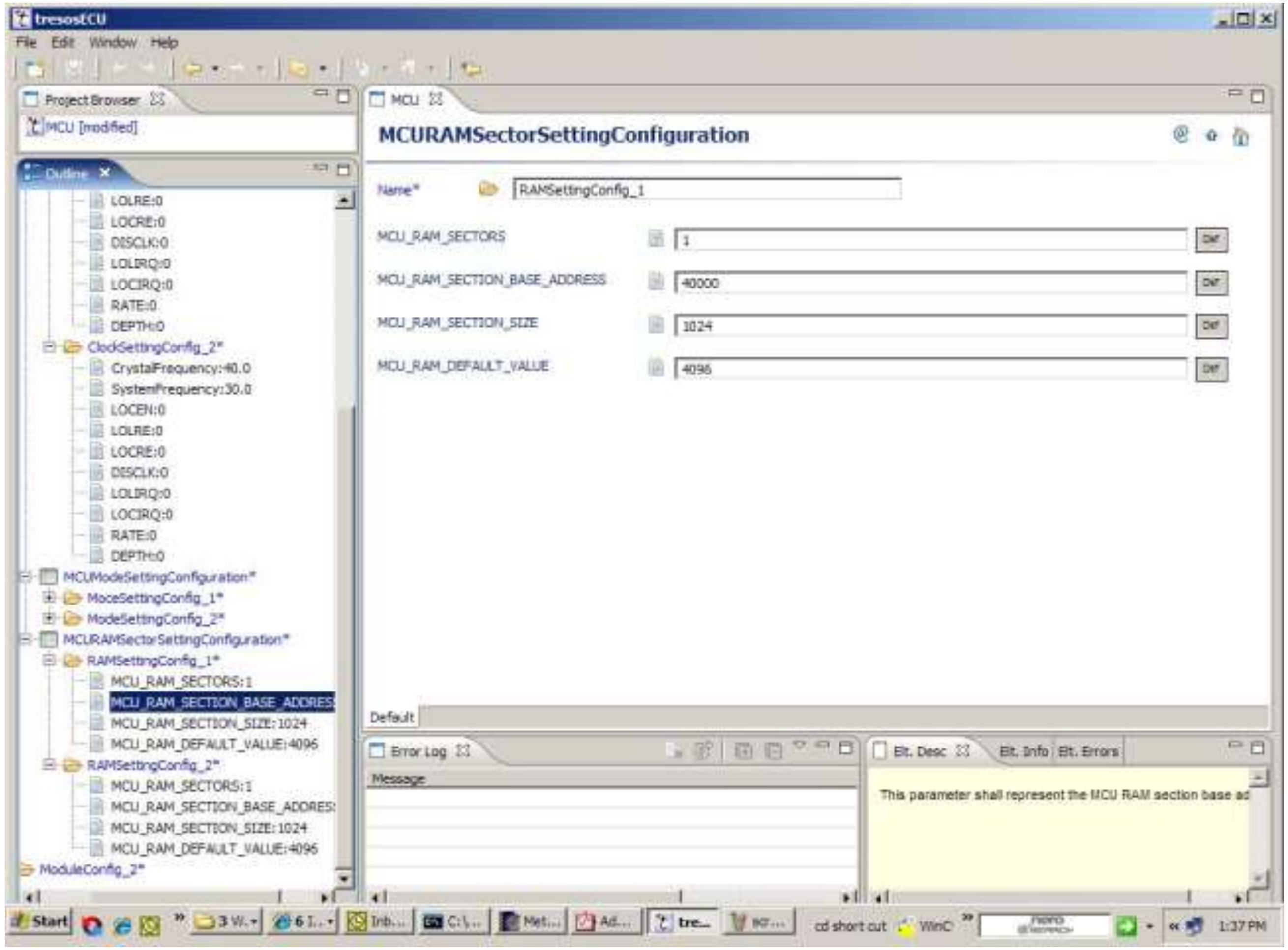


- EB tresos Studio is an easy-to-use tool for ECU standard software configuration, validation and code generation
- Full support for the AUTOSAR standard
- Full support for the Freescale AUTOSAR software and the EB tresos AutoCore
  - Integrated, graphical user interface
  - Based upon Eclipse and open standards
  - Online-help and parameter-specific help

AUTOSAR BSW Configuration Tool

Example: Tresos® ECU

- Graphical representation of ECU configuration description (ECD)
- Import/export of ECD
- Easy configuration of AUTOSAR BSW using pre-compile methodology

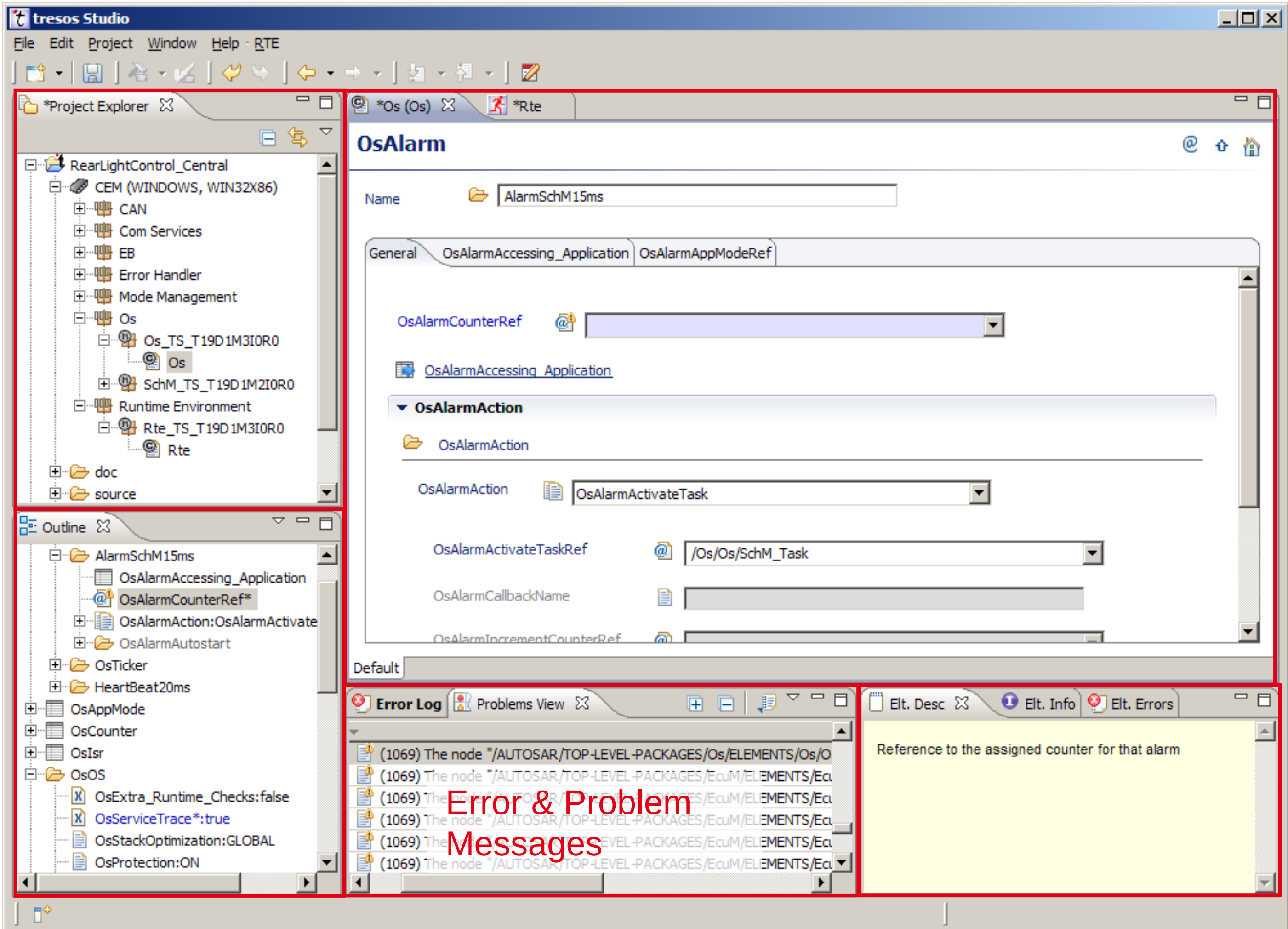


Source: Elektrobit Automotive

Main Window

Project Browser

Node Outline

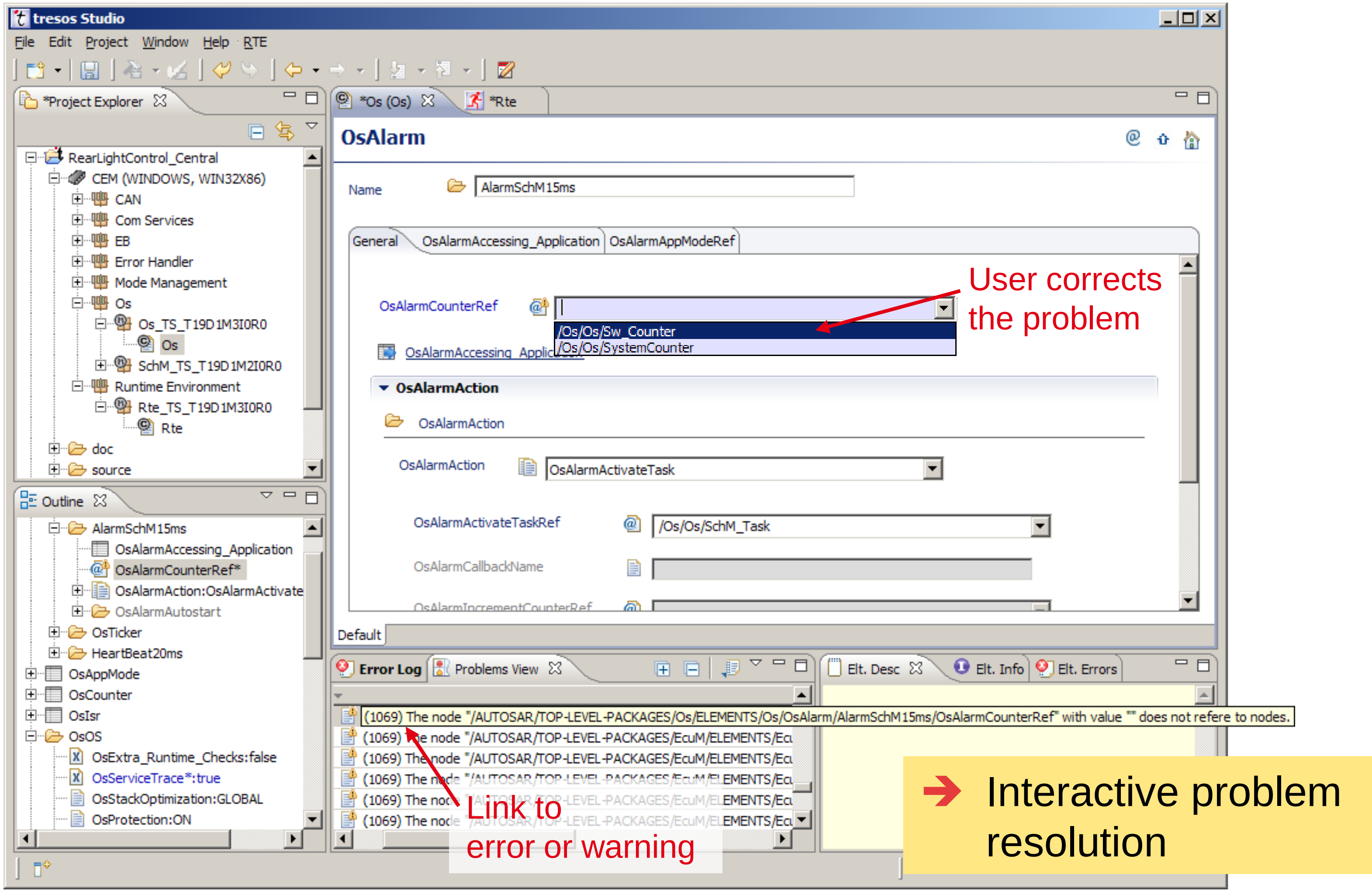


Editor

Parameter Information

Source: Elektrobit

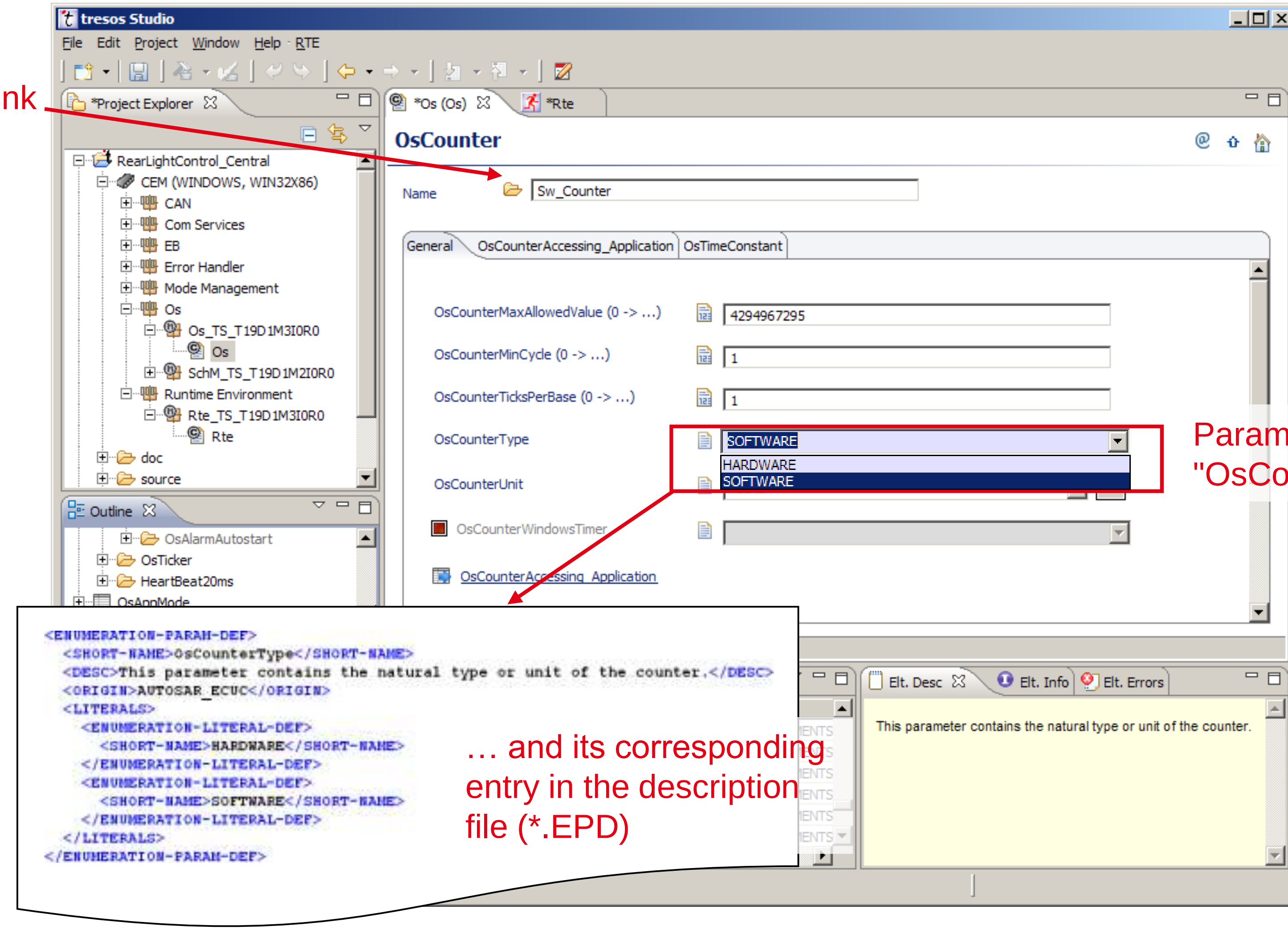
Errors & Warnings



Source: Elektrobit

Parameter Definition

Jump to link

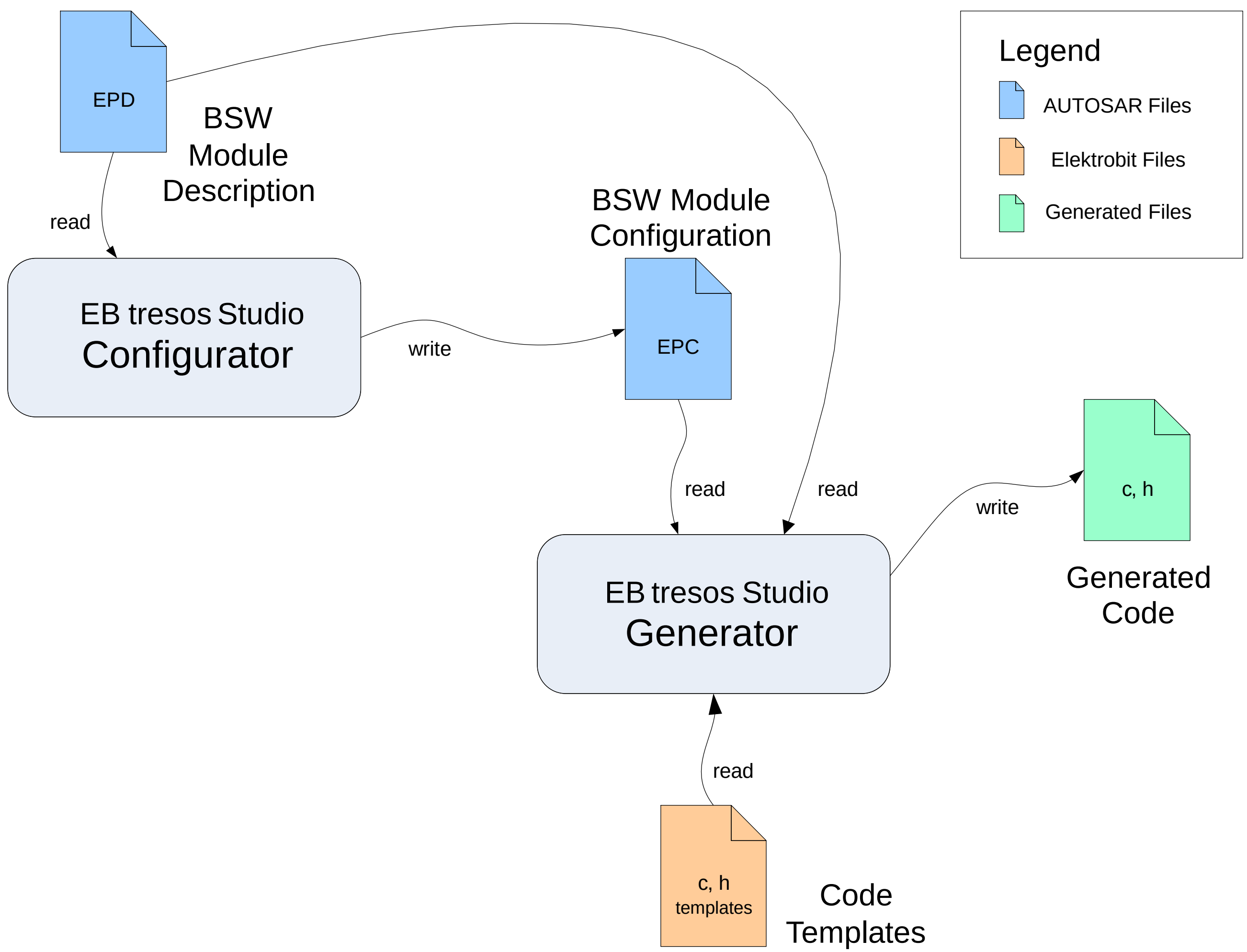


Parameter "OsCounterType"

… and its corresponding entry in the description file (*.EPD)

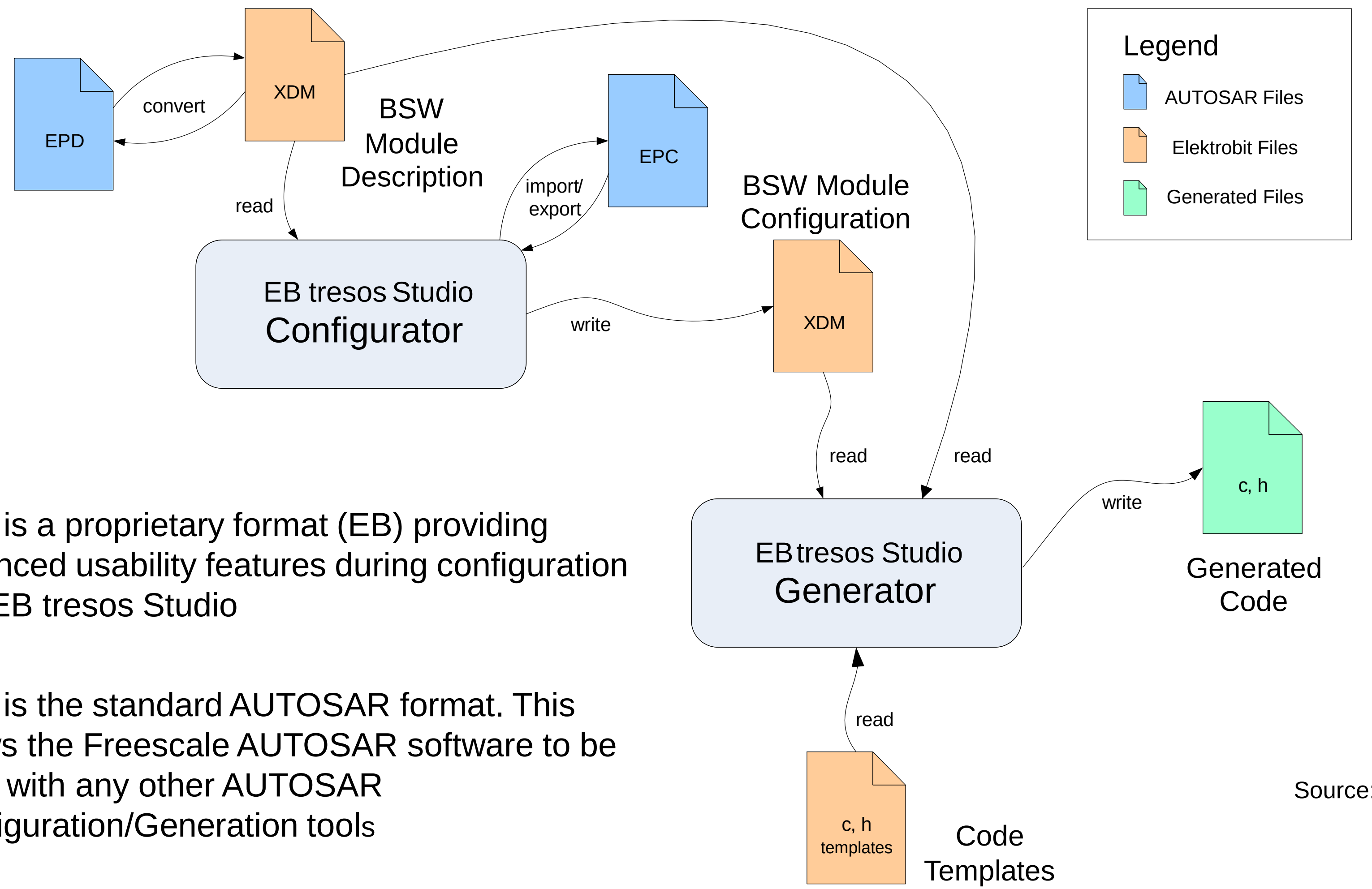
Source: Elektrobit

Parameter Description Files — EPD/EPC



Source: Elektrobit

Parameter Description Files — XDM



- **XDM** is a proprietary format (EB) providing enhanced usability features during configuration with EB tresos Studio

- **EPD** is the standard AUTOSAR format. This allows the Freescale AUTOSAR software to be used with any other AUTOSAR Configuration/Generation tools

Source: Elektrobit

AUTOSAR Configuration Classes

- Configuration classes (for parameters):
  - The development of BSW modules involve the following development cycles: compiling, linking and downloading of the executable to ECU memory
  - Configuration of parameters can be done in any of these process-steps: pre-compile time, link time and post-build time

AUTOSAR Configuration Classes

The AUTOSAR Basic Software supports the following configuration classes (for parameters):

1. **Pre-compile time**
   - Preprocessor instructions
   - Code generation (selection or synthetization)

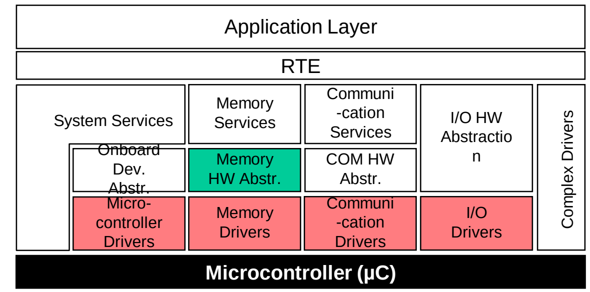
2. **Link time**
   - Constant data outside the module; the data can be configured after the module has been compiled

3. **Post-build time**
   - Loadable constant data outside the module. Very similar to [2], but the data is located in a specific memory segment that allows reloading (e.g. reflashing in ECU production line)

Independent of the configuration class, single or multiple configuration sets can be provided by means of variation points. In case that multiple configuration sets are provided, the actual used configuration set is to be chosen at runtime in case the variation points are bound at runtime.

AUTOSAR MCAL

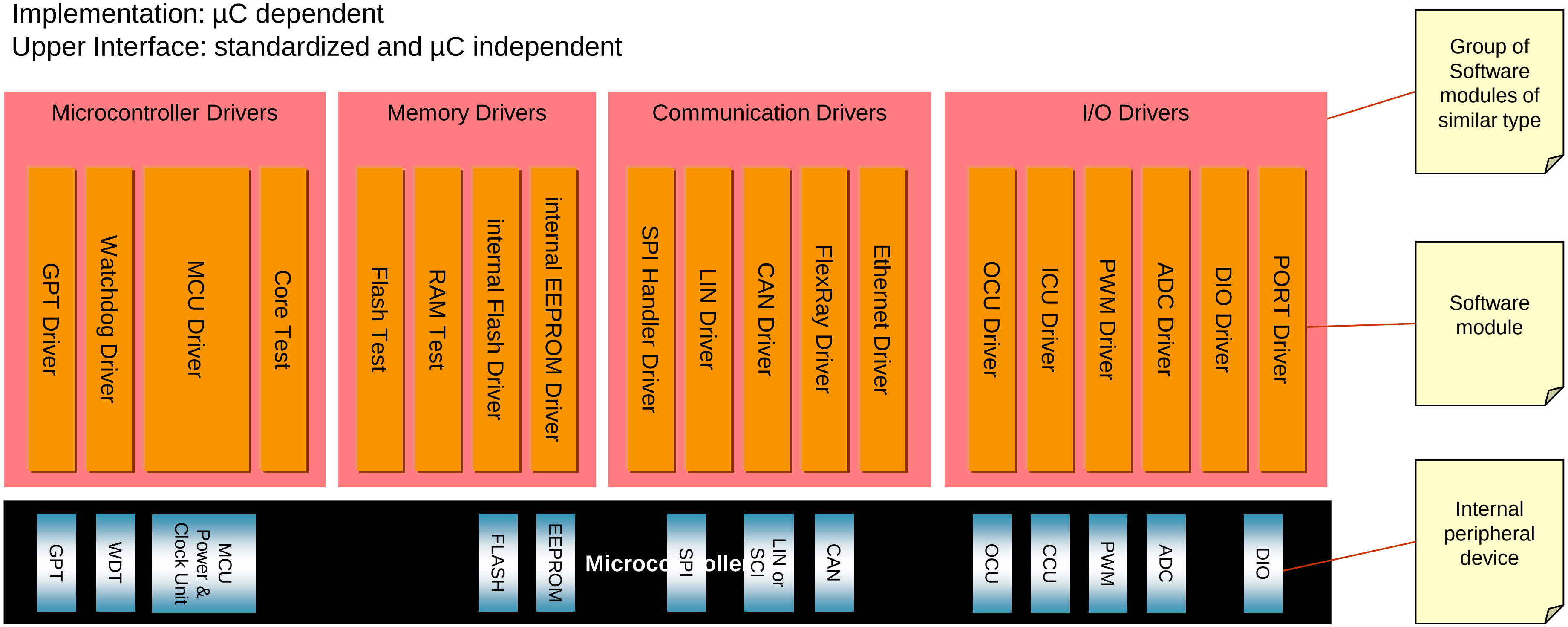
AUTOSAR — Microcontroller Abstaction Layer



The **Microcontroller Abstraction Layer** is the lowest software layer of the Basic Software.
It contains internal drivers, which are software modules with direct access to the µC and internal peripherals.

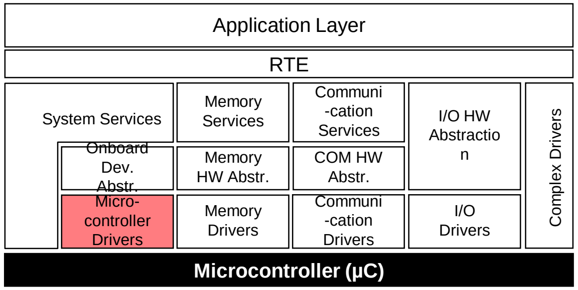
Task
Make higher software layers independent of µC

Properties
Implementation: µC dependent
Upper Interface: standardized and µC independent

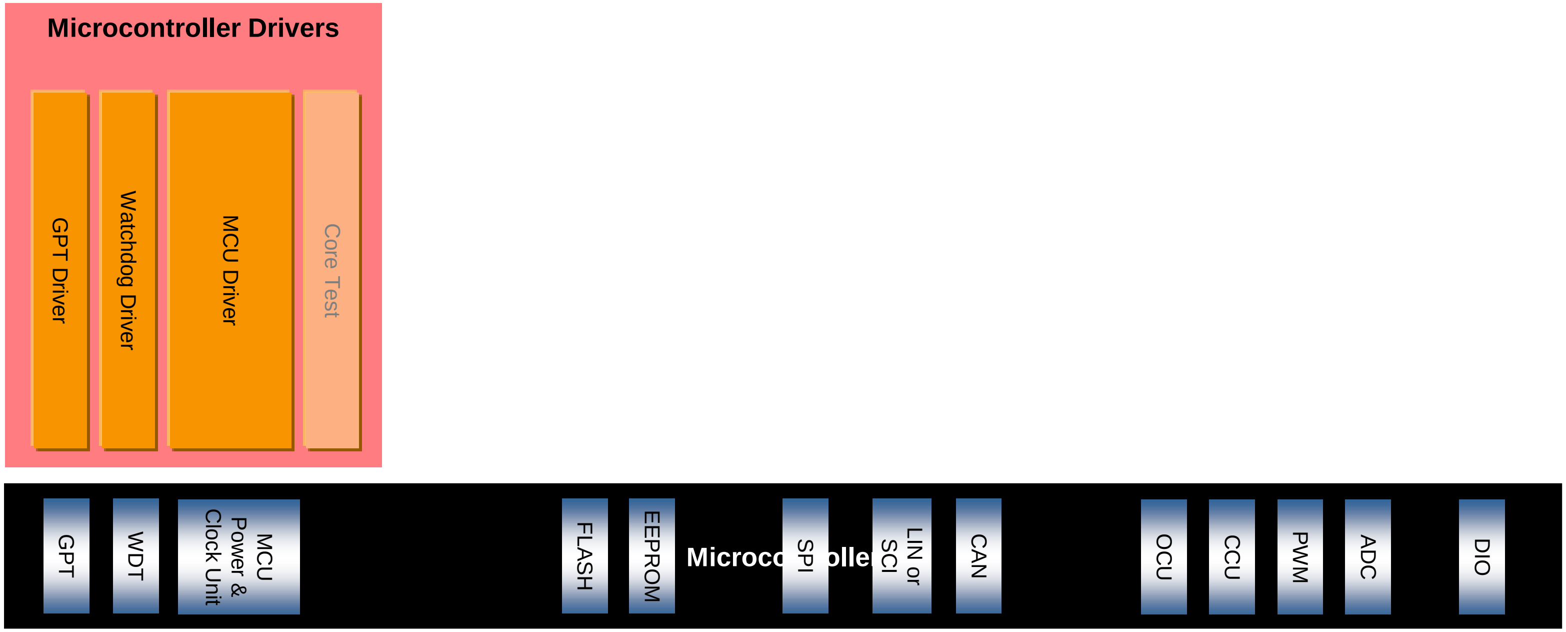


Source: AUTOSAR

AUTOSAR — Microcontroller Abstaction Layer

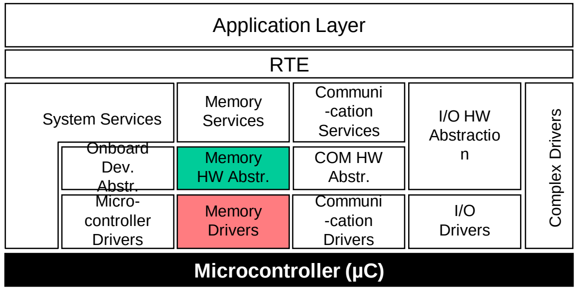


- **Microcontroller Drivers**
  - Drivers for internal peripherals (e.g. Watchdog, General Purpose Timer)
  - Functions with direct µC access

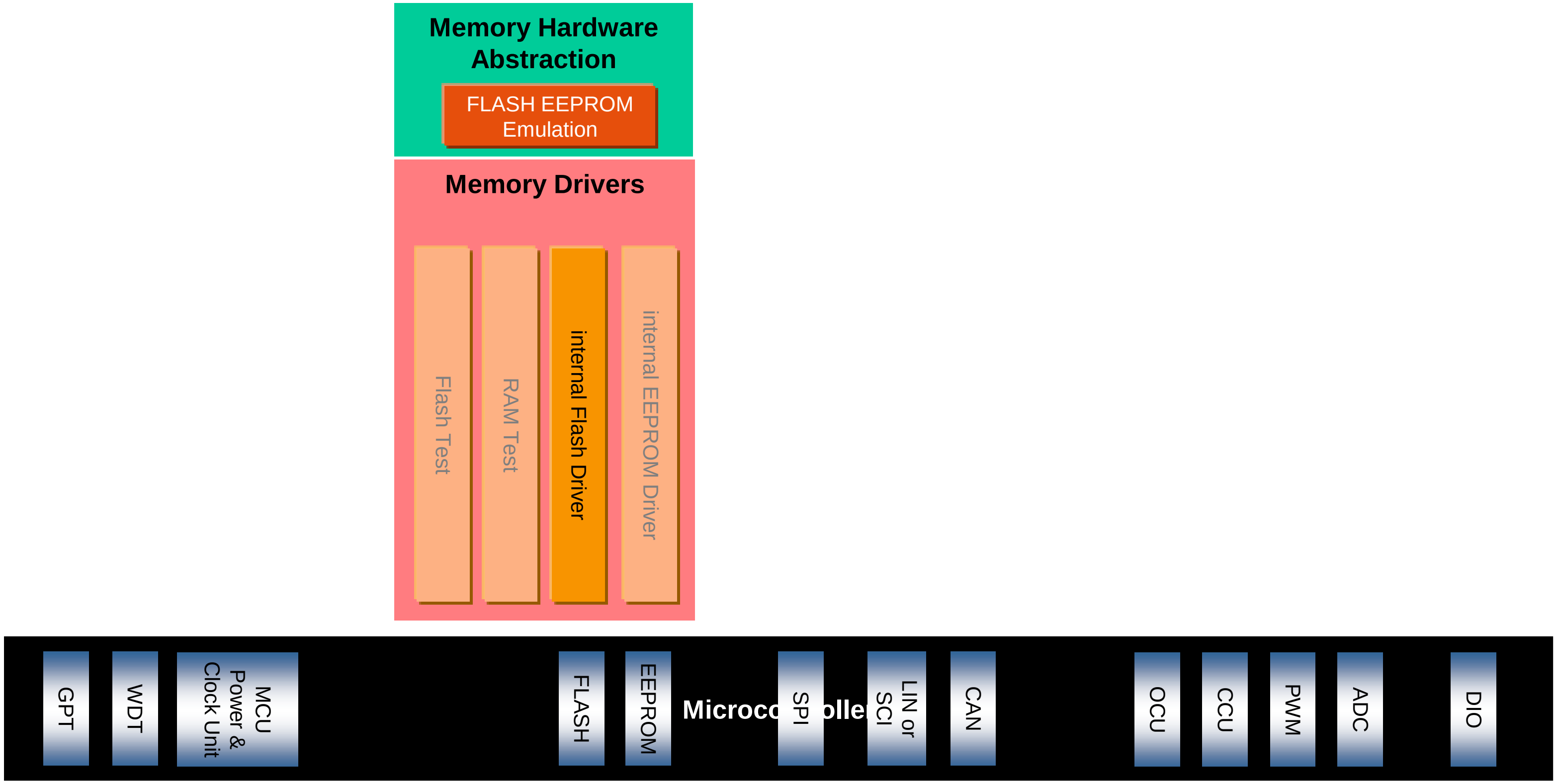


Source: AUTOSAR

AUTOSAR — Microcontroller Abstaction Layer

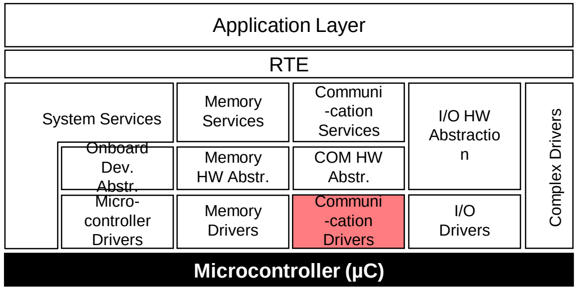


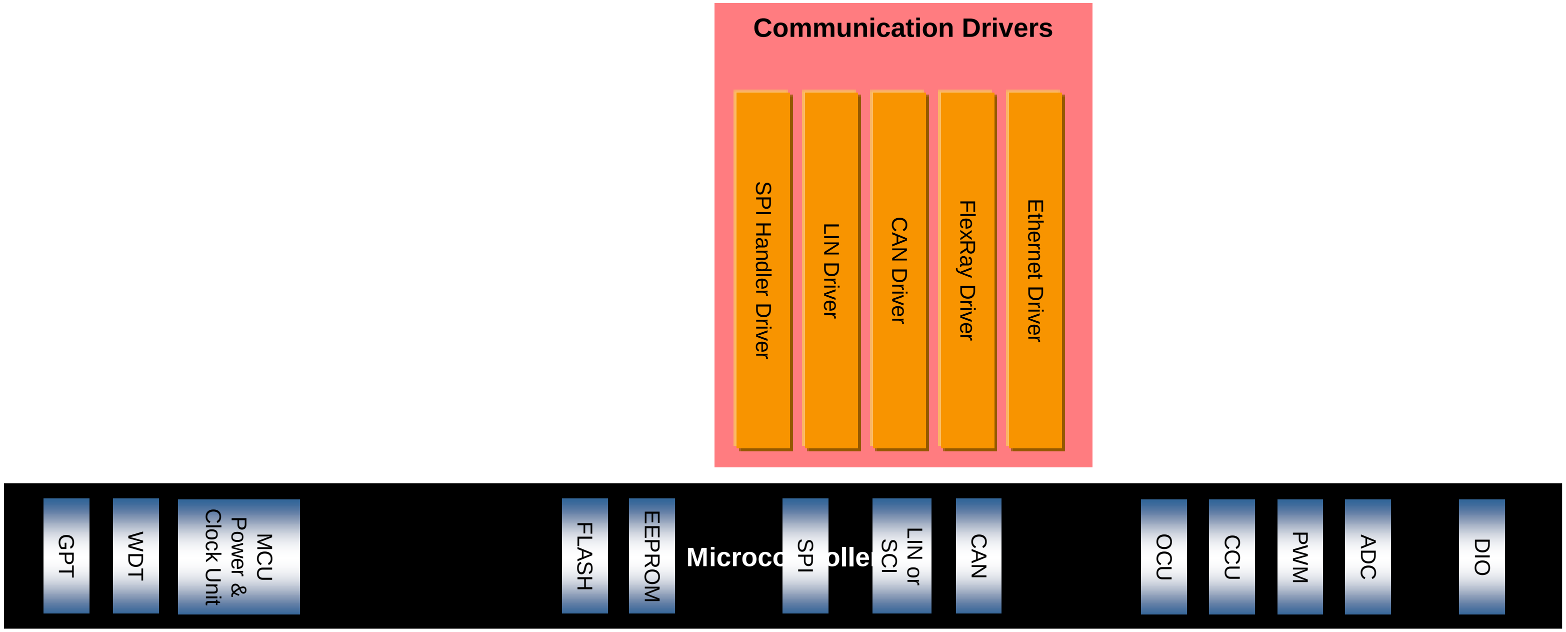
- **Memory Drivers**
  - The **Memory Hardware Abstraction** is a group of modules which abstracts from the location of peripheral memory devices (on-chip or on-board) and the ECU hardware layout
  - Example: on-chip EEPROM and external EEPROM devices are accessible via the same mechanism
  - The **Memory Drivers** are accessed via memory specific abstraction/emulation modules (e.g. EEPROM Abstraction)



AUTOSAR — Microcontroller Abstaction Layer

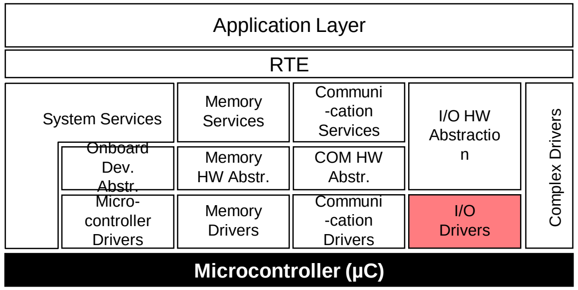
- **Communication Drivers**
  - Drivers for ECU onboard (e.g. SPI) and vehicle communication (e.g. CAN)
  - OSI-Layer: Part of Data Link Layer



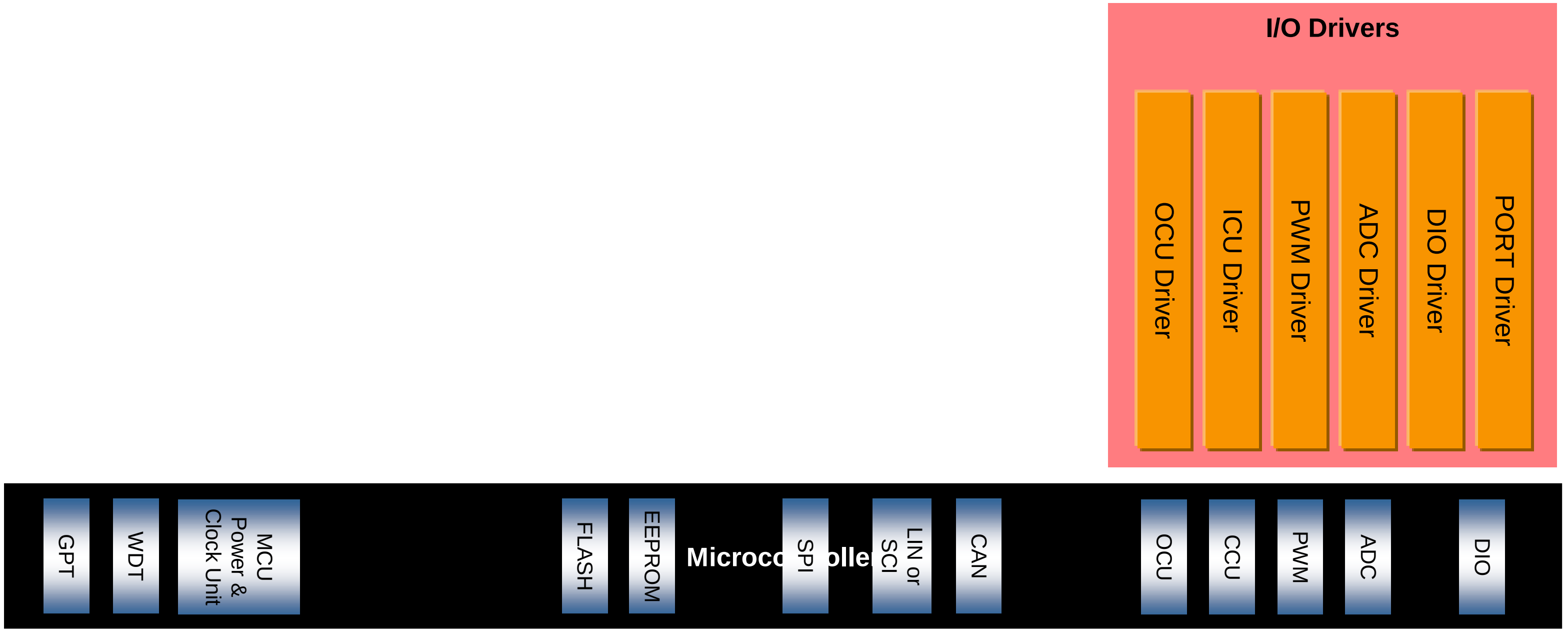


Source: AUTOSAR

AUTOSAR — Microcontroller Abstaction Layer

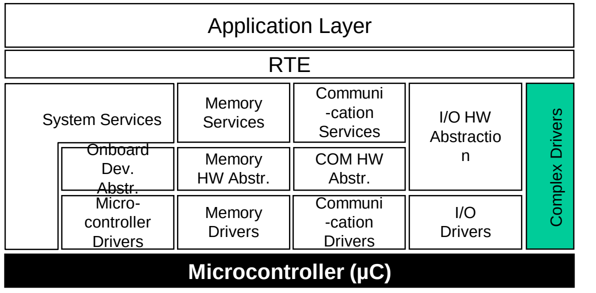


- **I/O Drivers**
  - Drivers for analog and digital I/O (e.g. ADC, PWM, DIO)



Source: AUTOSAR

AUTOSAR — Complex Device Drivers

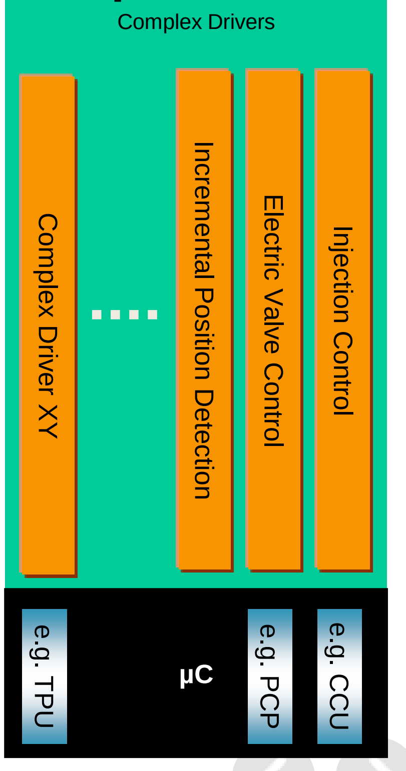


A **Complex Driver** is a module which implements non-standardized functionality within the basic software stack.

An example is to implement complex sensor evaluation and actuator control with direct access to the µC using specific interrupts and/or complex µC peripherals e.g.

- *Fault Monitoring Drivers*
- *Core and Peripheral Self Tests*
- *MicroController Library* (MCL)
- *CRC Driver*

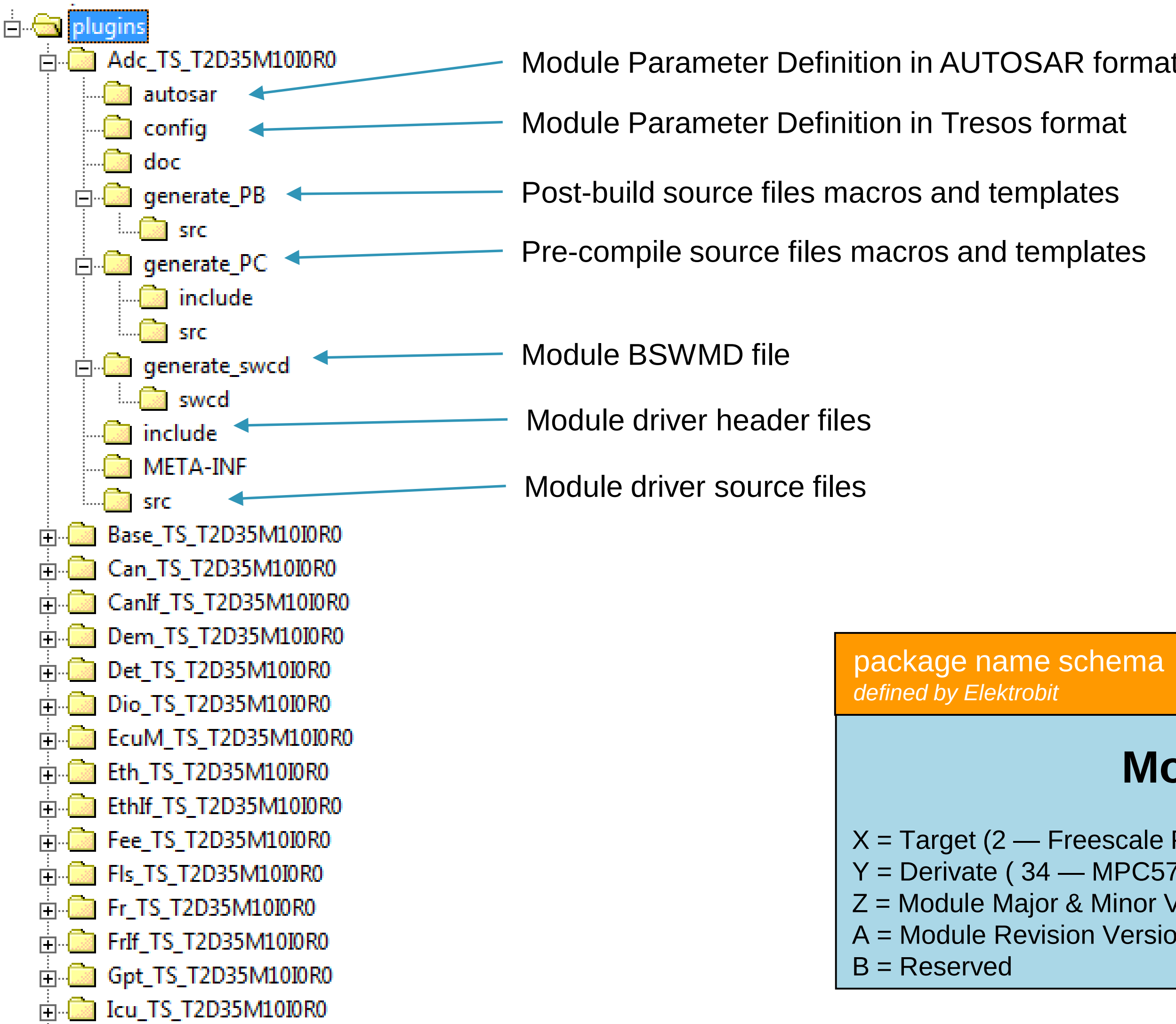
Example:



Properties:
- *Implementation:* highly µC, ECU and application dependent
- *Upper Interface to SW-Cs*: specified and implemented according to AUTOSAR (AUTOSAR interface)
- *Lower interface:* restricted access to Standardized Interfaces

Source: AUTOSAR

Freescale AUTOSAR MCAL Product



package name schema
defined by Elektrobit

Module_TS_TxDyMzIaRb

X = Target (2 — Freescale PPC)
Y = Derivate (34 — MPC5748G)
Z = Module Major & Minor Version
A = Module Revision Version
B = Reserved

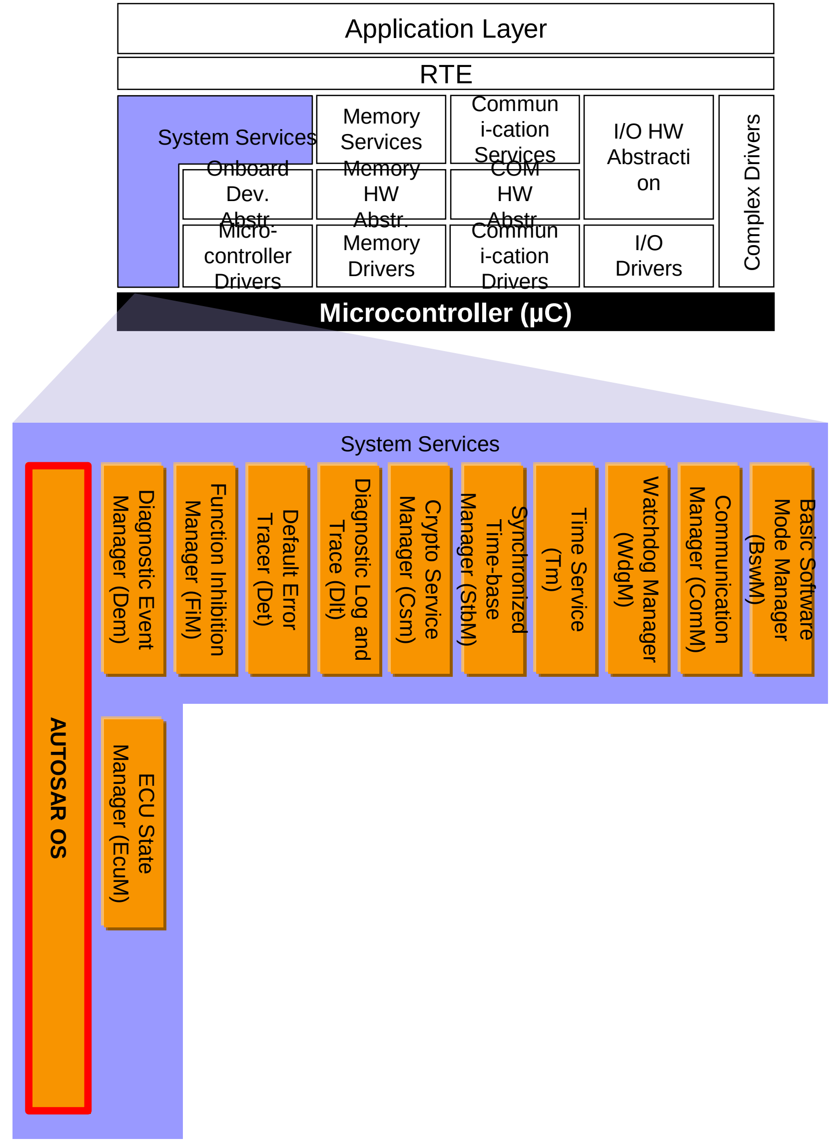
AUTOSAR OS

History: OSEK/VDX

- May 1993
  - Funded by a German company consortium BMW, Robert Bosch GmbH, DaimlerChrysler, Opel, Siemens, and Volkswagen Group in order to create an open standard for the automotive industry
  - Open Systems and their Interfaces for the Electronics in Motor Vehicles
- 1994
  - French cars manufacturers Renault and PSA Peugeot Citroën, which had a similar project called VDX (Vehicle Distributed eXecutive), joined the consortium
- Oct 1997
  - 2nd release of specification package
- Feb 2005
  - Specification 2.2.3 of OSEK OS
- **Goals**: portability and reusability

AUTOSAR OS

- AUTOSAR OS is OSEK/VDX™ OS plus:
  - **New core features**
    - Software and hardware counters
    - Schedule tables with time synchronisation
    - Stack monitoring
  - **Protection features**
    - Timing protection, memory protection and service protection
    - OS applications, trusted and non-trusted code
    - Protection hook

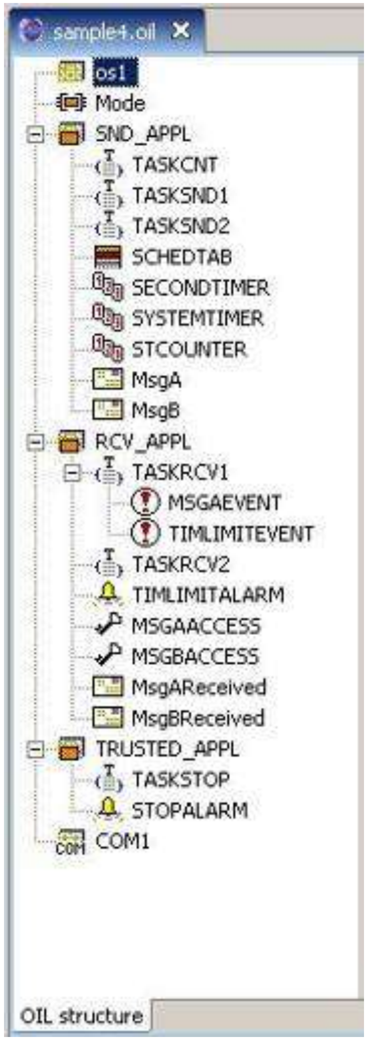


AutoSAR OS — Application and Trusted and Non-Trusted Code

- **Integrity level**: trusted and non-trusted code

- **OS application**
  - A block of software including tasks, ISRs, hooks and trusted functions
  - **Trusted**: An OS application that has unrestricted access
  - **Non-trusted**: An OS application that has restricted access

- **Trusted function**
  - A service function with unrestricted access
  - Provided by a trusted OS application



AUTOSAR OS — Usage of Memory Protection

- A Non-trusted OS application task
  - Can only access the configured resources (i.e. Memory, peripherals, ...)
  - Therefore this task is unabled to interfere with other components in the system

- **Memory protection** can be used, e.g.,
  - To separarate different applications on one MCU
  - For isolating controller functionality from independent sub-suppliers
  - To fulfill safety constraints
  - As a debug feature (faulty memory access is prevented, stack overflow is prevented, protection hook is called)

- Memory protection MUST be supported by on-chip hardware resources (i.e. MPU)

AUTOSAR OS — Usage of Service Protection

- **Service Protection**
  - Protection against faulty/corrupted OS service calls by an OS Application
  - Examples
    - OS Application calls ShutDownOS()
    - OS Application tries to execute ActivateTask() on a task belonging to another OS Application
  - Protection Hook is called upon detection of a service protection error

AUTOSAR OS — Usage of Timing Protection & Global Time

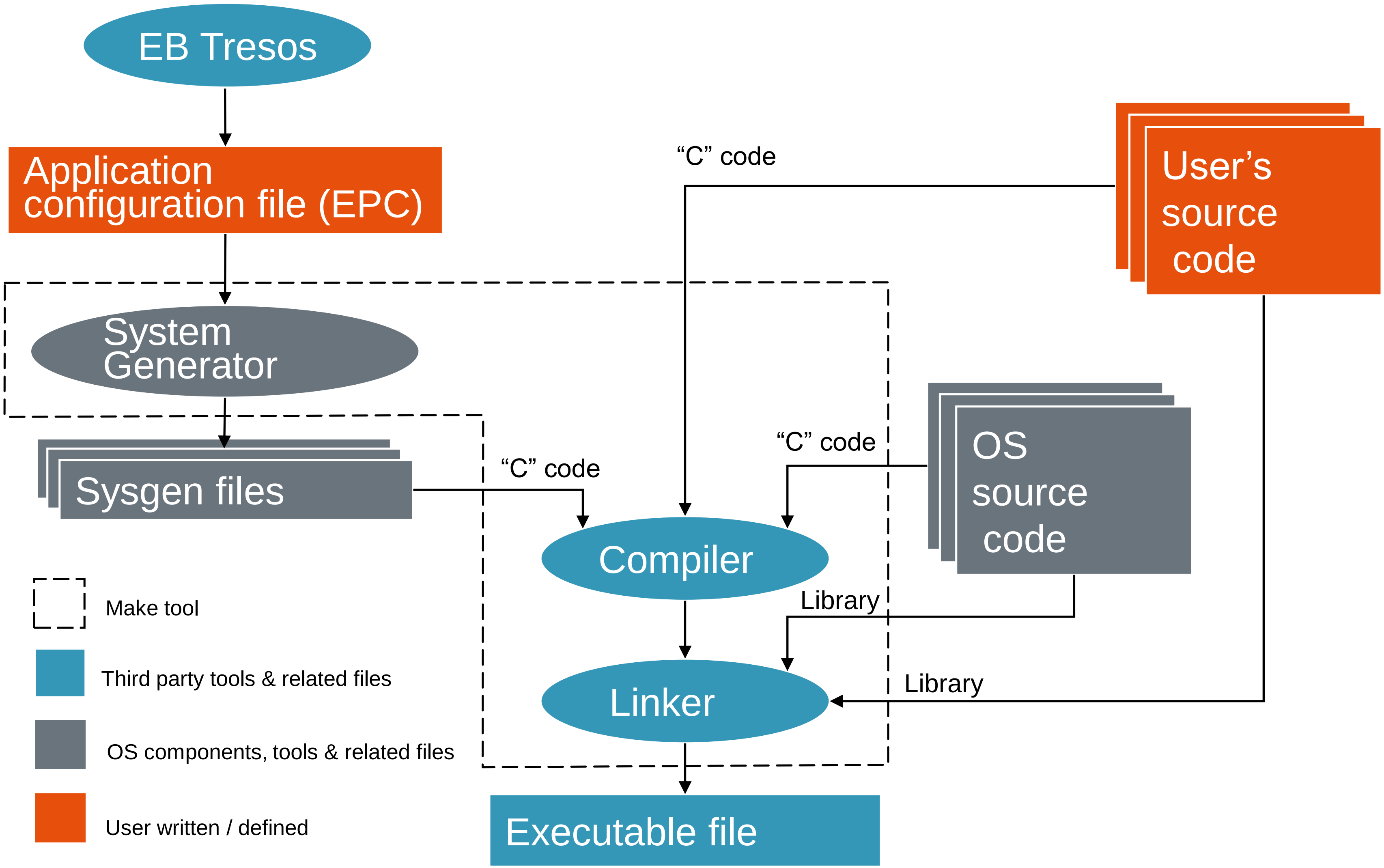
- **Timing Protection**
  - Execution time enforcement
    - Bounds the execution of ISRs, resource locks and interrupt disabled sections at runtime to a statically configured value (“time budget“)
  - Arrival rate enforcement
    - Bounds the number of times that an ISR can execute in a given timeframe to a statically configured limit
  - Protection Hook is called upon detection of a timing protection violation

- **Global Time / Synchronization Support**
  - Requires a global time source, e.g. the FlexRay network time
  - This feature allows schedule tables to be synchronized with a global time through special OS service calls

AUTOSAR OS Scalability Classes 1–4

	Scalability Class 1	Scalability Class 2	Scalability Class 3	Scalability Class 4
OSEK OS (all conformance classes)	✓	✓	✓	✓
Counter Interface	✓	✓	✓	✓
Schedule Tables	✓	✓	✓	✓
Stack Monitoring	✓	✓	✓	✓
Protection Hook		✓	✓	✓
Timing Protection		✓		✓
Global Time/Synchronization Support		✓		✓
Memory Protection			✓	✓
OS Applications			✓	✓
Service Protection			✓	✓
CallTrustedFunction			✓	✓

Freescale AUTOSAR OS Application Architecture

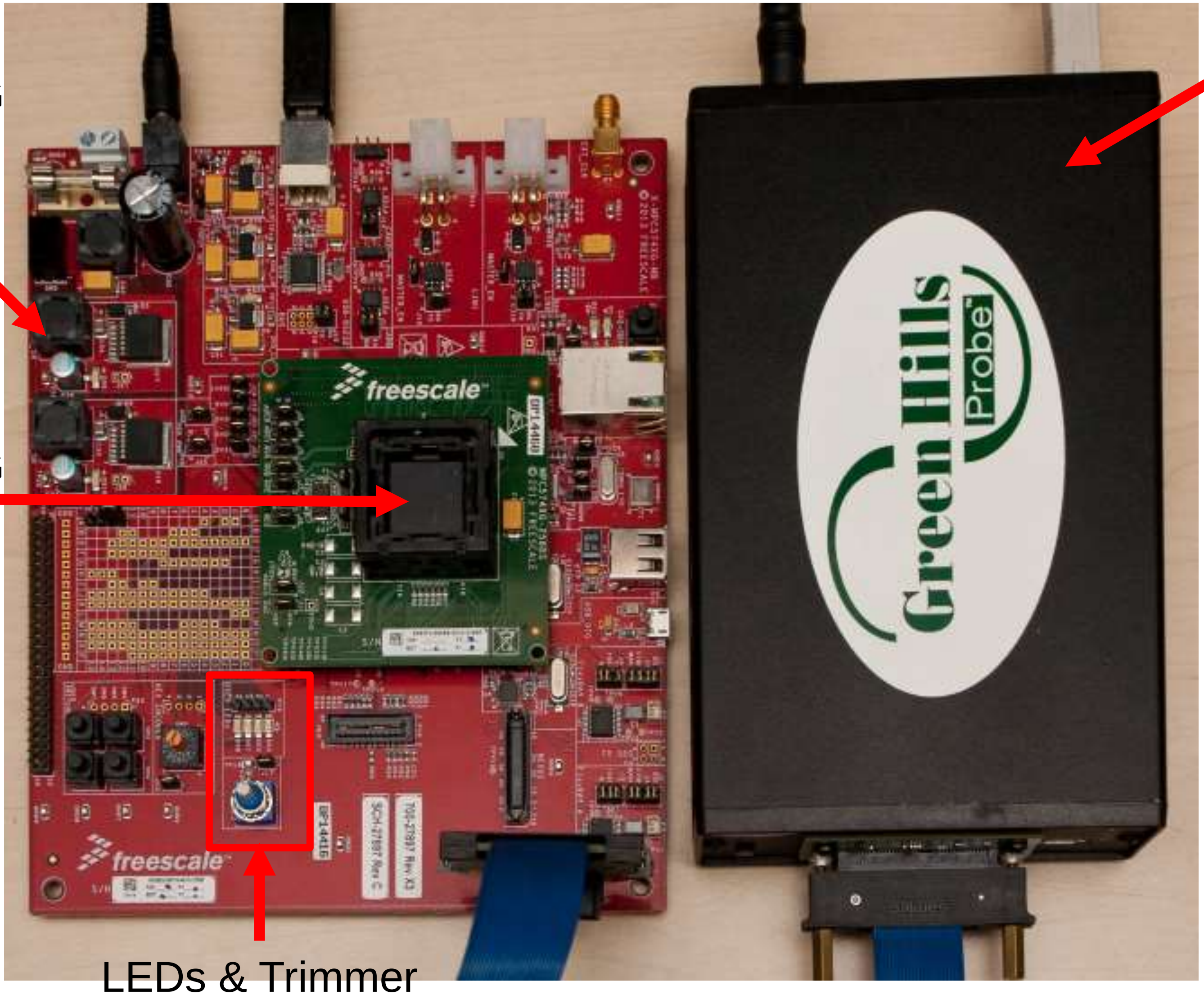


AUTOSAR Hands-On Training

What's on Your Desk

MPC5748G Board

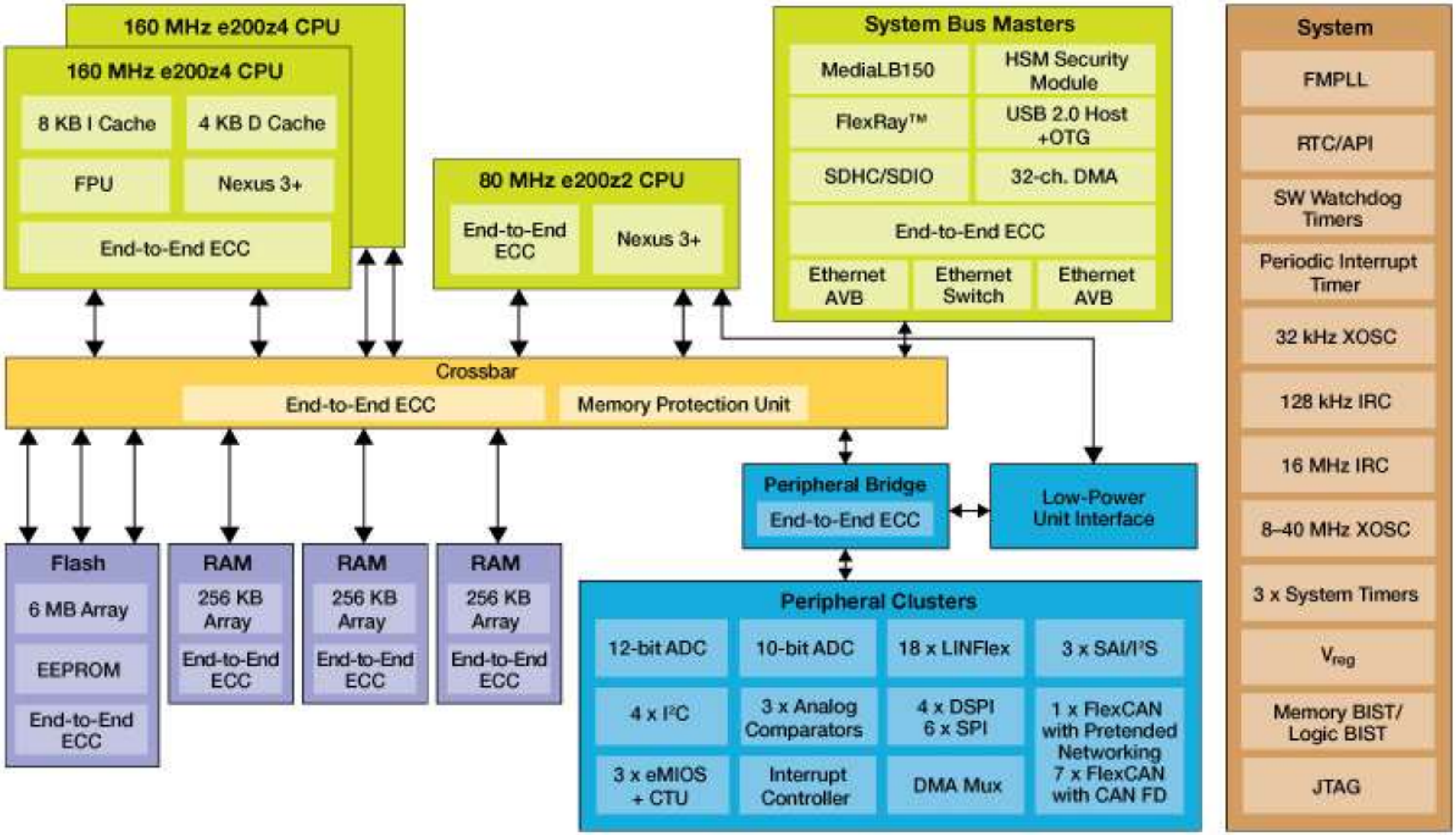
MPC5748G SoC



GreenHills Probe

LEDs & Trimmer

MPC5748G — Block Diagram



LAB1 Blinking LED

- **Objective**
  - Get started with AutoSAR and Blinking LED

- **Environment**
  - AutoSAR MCAL and AutoSAR OS v4.0
  - Tool: Elektrobit tresos Studio 2014.2.1
  - Compiler: GreenHills for PPC
  - Debugger: GreenHills Probes
  - Hardware: MPC5748G Evaluation Board

- **Functional description**
  - The AutoSAR BSW modules Mcu, Dio, Port, Os, EcuM, Rte are applied to build an application which toggles an LED every second.

PORT/DIO Modules — *Functional Overview*

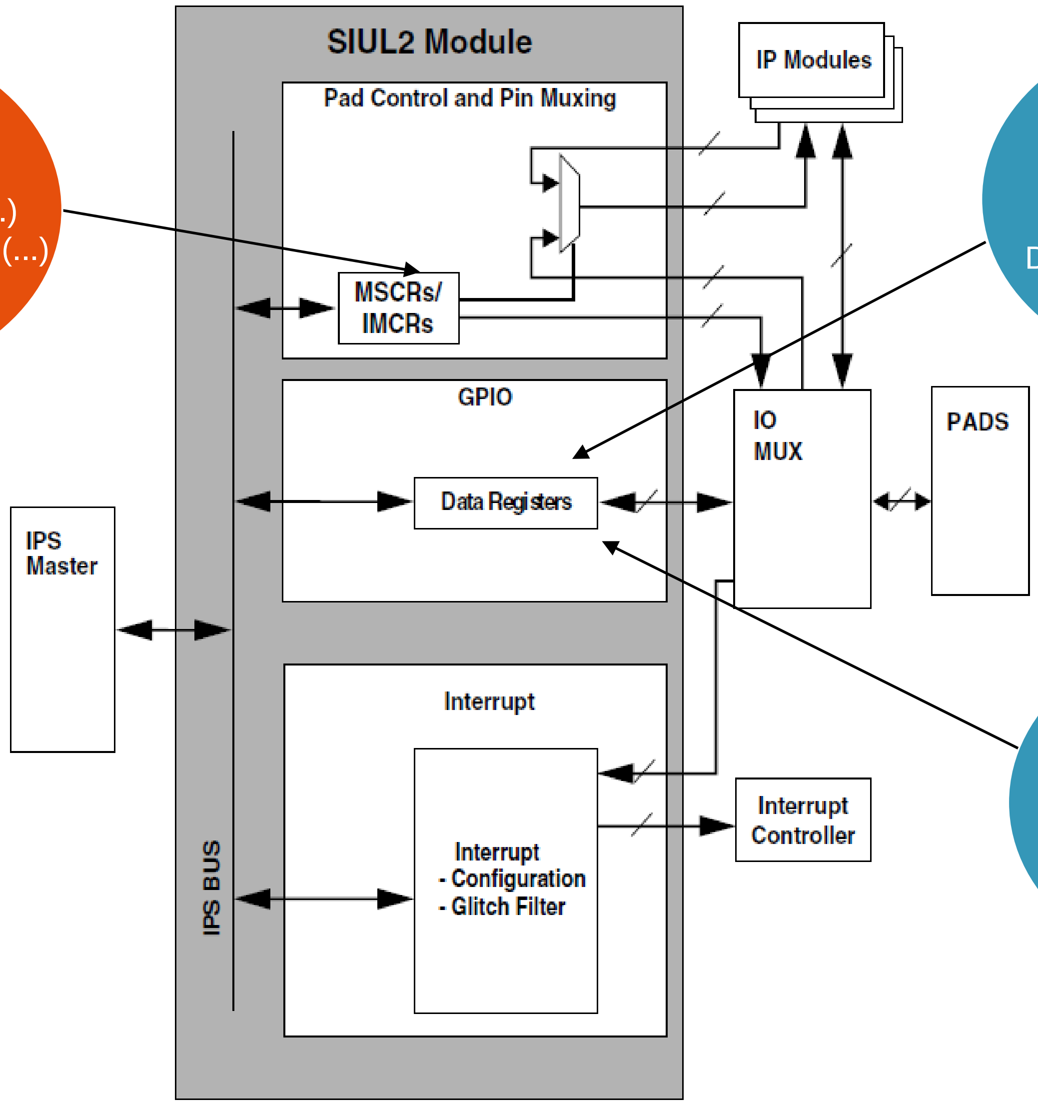
- **Port**
  - Initialization of all pins and ports of the MCU
  - Reinitialization with alternate configurations at runtime possible
  - Reconfiguration of pins at runtime
  - Port Pin Function Assignment (GPIO, Adc, SPI, PWM, ...)
  - PadSelection implicitly via hardware assignment
  - PortPin is the only structural element

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**
  - Provides APIs to read and write GPIO ports/pins
  - Requires an initialized Port module
    - Pins/ports need to be initialized via Port module
  - API synchronous and unbuffered
  - Structural Elements:
    - Channel (single pin)
    - ChannelGroup (adjacent pins in the same port)
    - Port (aggregates Channels and ChannelGroups)

PORT/DIO Modules — *Freescale Implementation*

Port Access:
Port_Init(...)
Port_SetPinDirection(...)
Port_RefreshPinDirection(...)
Port_SetPinMode(...)



Dio Write Accesses:
Dio_WriteChannel
Dio_WritePort
Dio_WriteChannelGroup

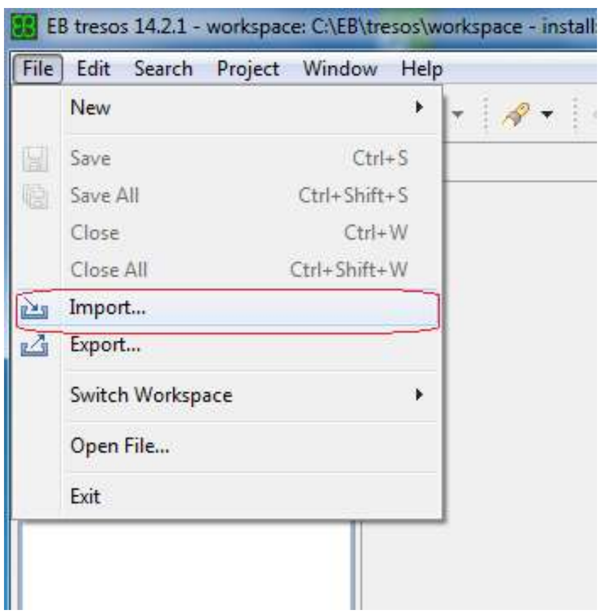
Dio Read Accesses:
Dio_ReadChannel
Dio_ReadPort
Dio_ReadChannelGroup

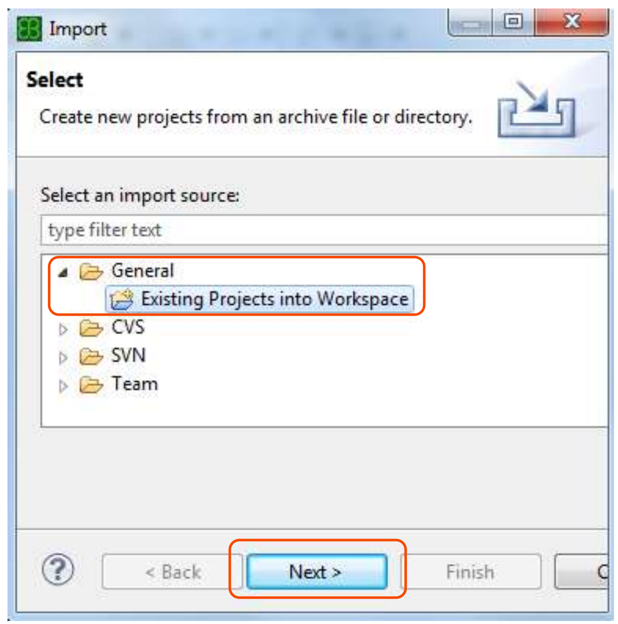
LAB1: Blinking LED

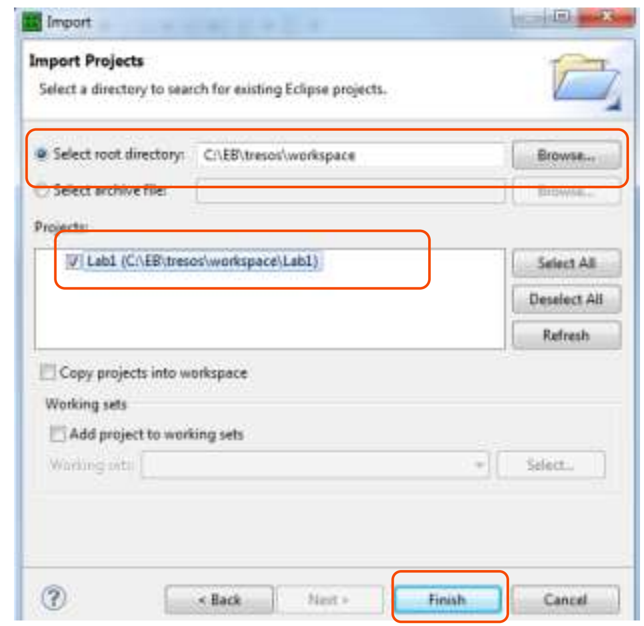
1. Opening a Tresos Project
2. Adding an AUTOSAR Module to the Project
3. Parameters Configuration for DIO and PORT
4. Code Generation
5. GreenHills Integration
6. Compilation and Debugging
7. AUTOSAR Runtime Application Flow

Opening a Tresos Project

1. ***File*** -> ***Import*** -> ***General*** -> ***Existing Projects into Workspace*** -> ***Select root Directory*** -> ***Browse*** *to c:\eb\tresos\workspace* ***-> Select Lab1 -> Finish***

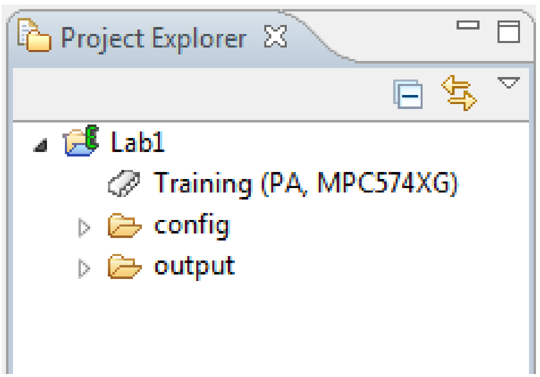


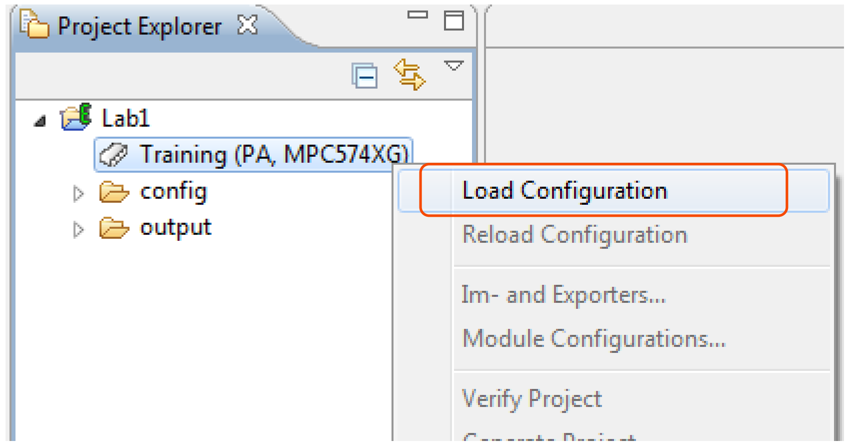


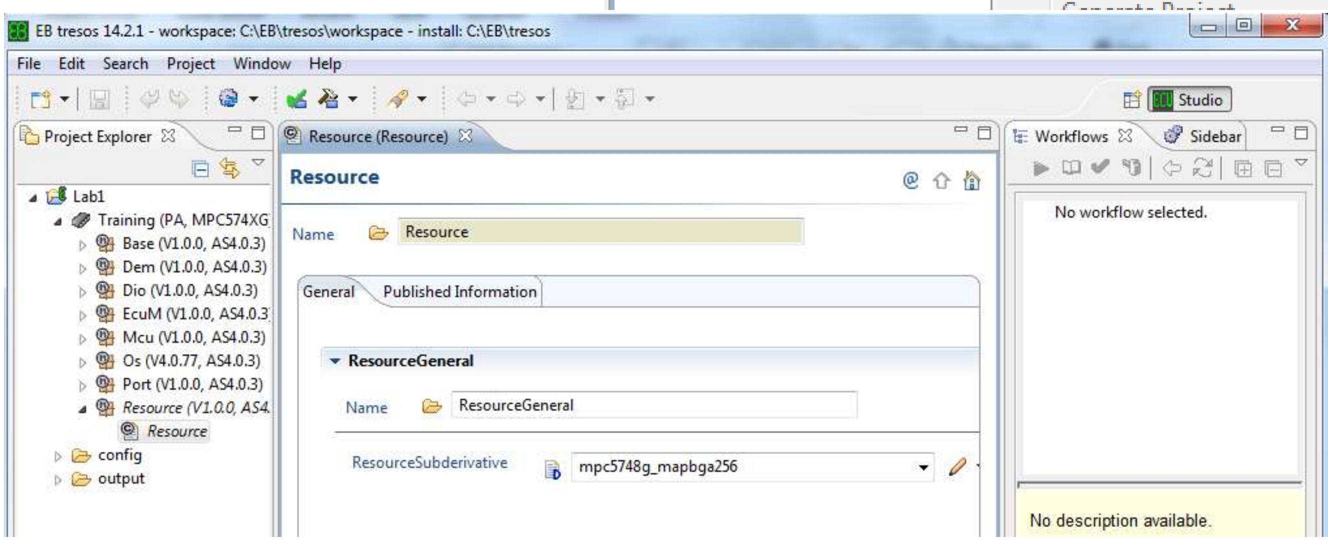


Opening a Tresos Project

2. *Right click on **Training** -> select **Load configuration***



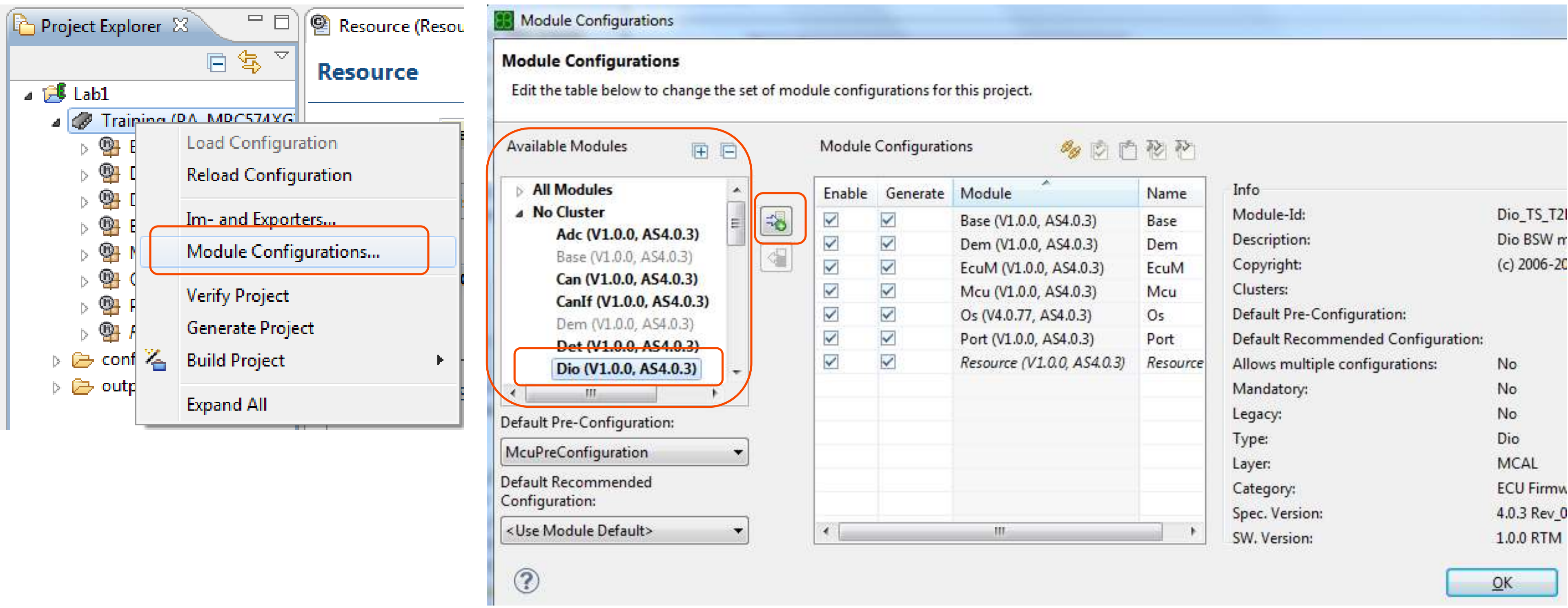




Adding an AutoSAR Module to Project

1. Right Click on ***Training*** and select ***Module Configurations…***

2. From List of ***Available Modules*** select **Dio** and **import** it into ***Module Configurations List*** -> Press ***Ok***



Parameters Configuration

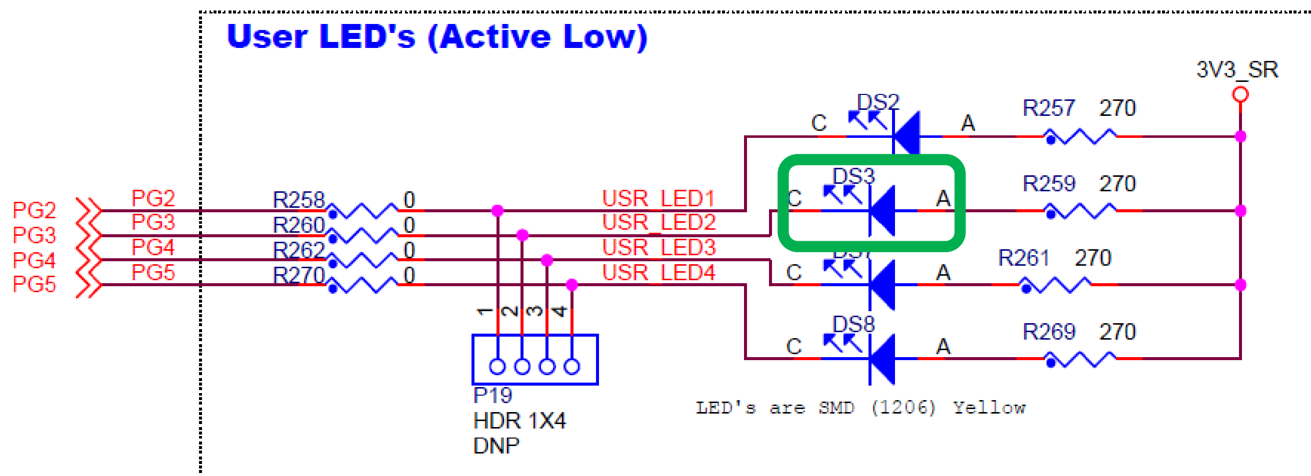
- **Objective**
  - You start with an empty/initial ECU-configuration. This step describes how to complete this configuration for your first project. Therefore, parameters will be modified and containers will be added

- **Procedure**
  - The next slides will show which Containers/Parameters to add/change
  - To open a module configuration, double click the module in the Project Explorer window
  - To navigate within a previously opened module configuration, use the Outline window on the bottom left side
  - To change parameter, click on that parameter in Outline window
  - To add a container, click on the collection item of this container type (e.g. DioPort). You see a listview in the main window which lets you add new entries by clicking the + button
  - To edit a previously added container in the main window, click on it in the Outline window

Parameters Configuration

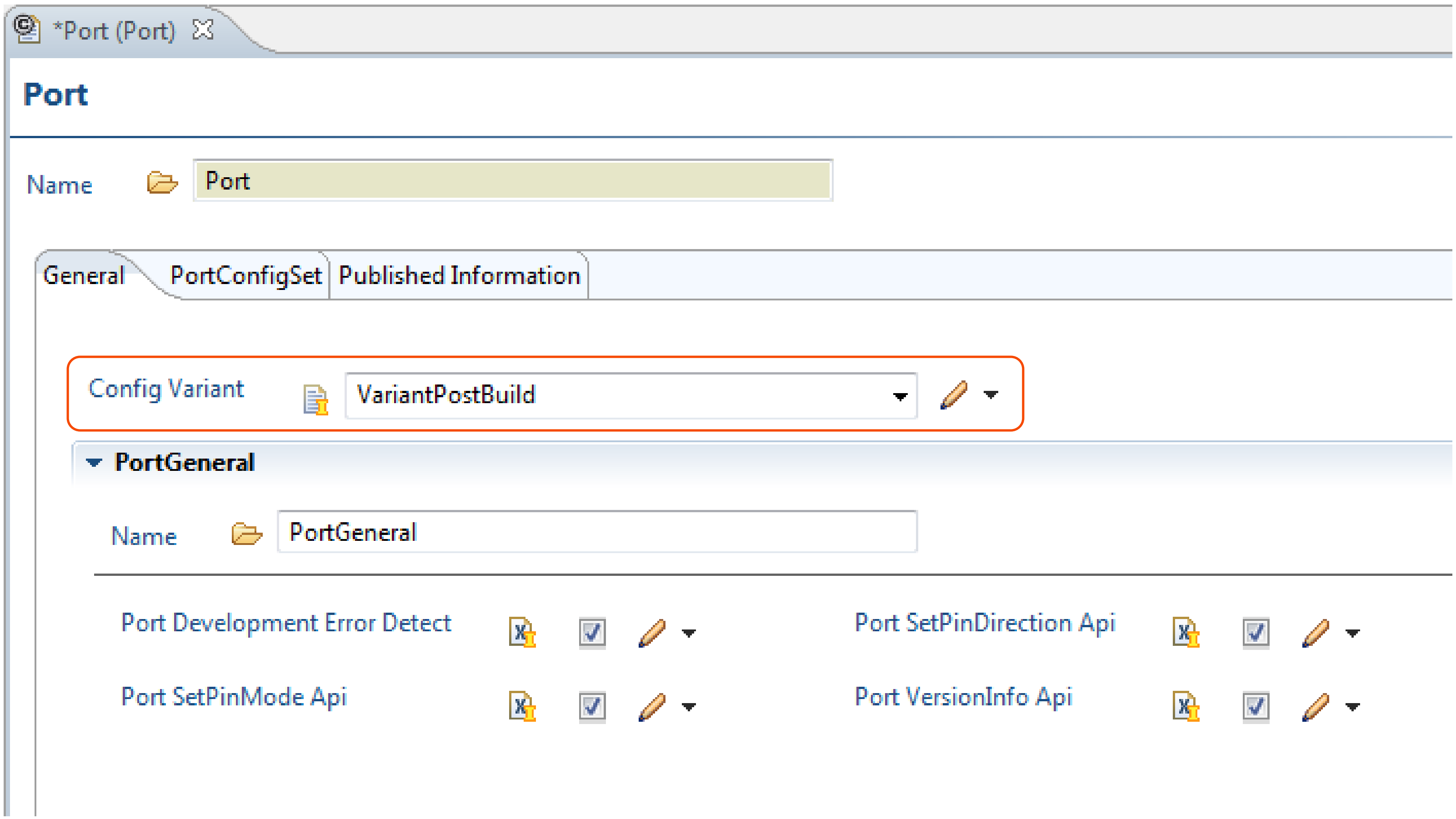
- **Port**
  - Open and Explorer the container “Port”
  - Open PortConfigSet_0 container
  - Add a PortPin to the container PortConfigSet_0
    - Name: Led2
    - PortPinPcr = 99
    - PortPinDirection = PortPinDirectionOut

- **Dio**
  - Open and Explorer the container “Dio”
  - Go to the container “Dio_Port_0” and add
  - a port with the following proprieties:
    - Name: Dio_PG
    - DioPortId: 6
  - Add a DioChannel to the Container “Dio_PG”
    - Name: Dio_Led2
    - DioChannelId: 6



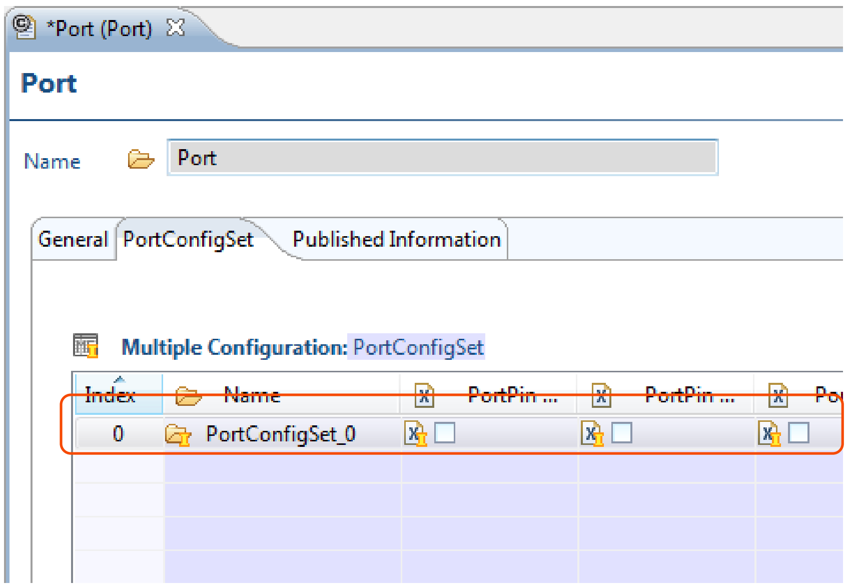
PORT Module Configuration

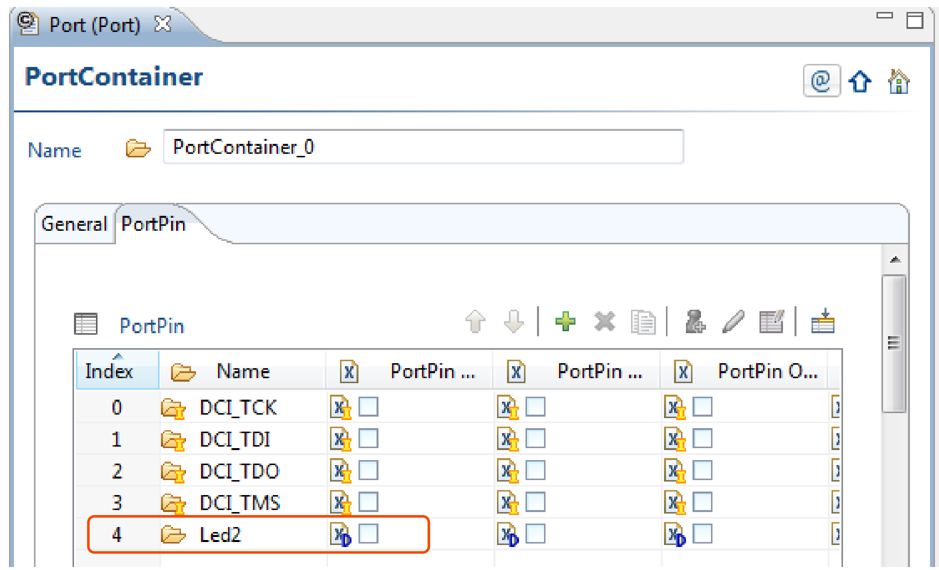
- Config Variant



PORT Module Configuration

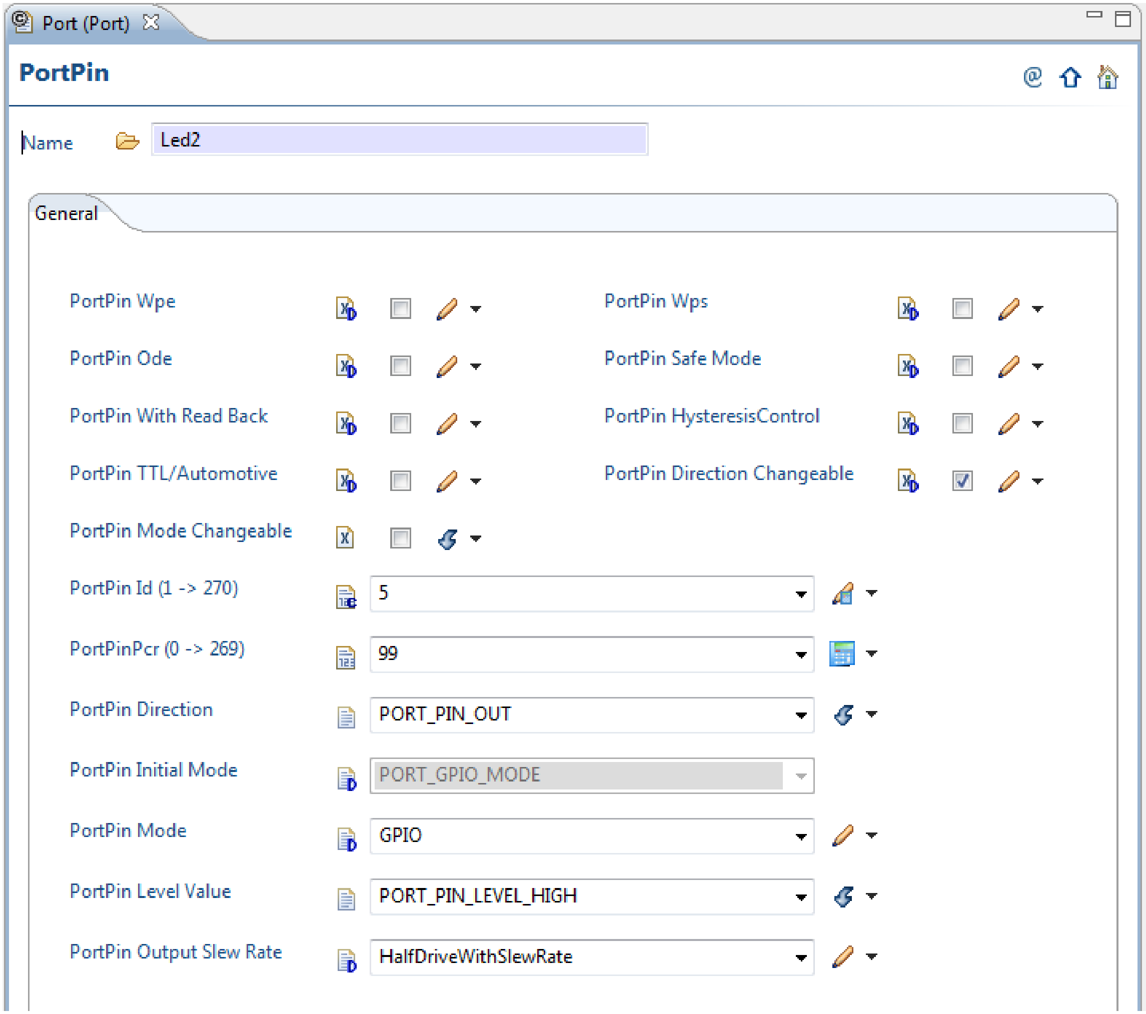
- PortConfigSet and PortPin





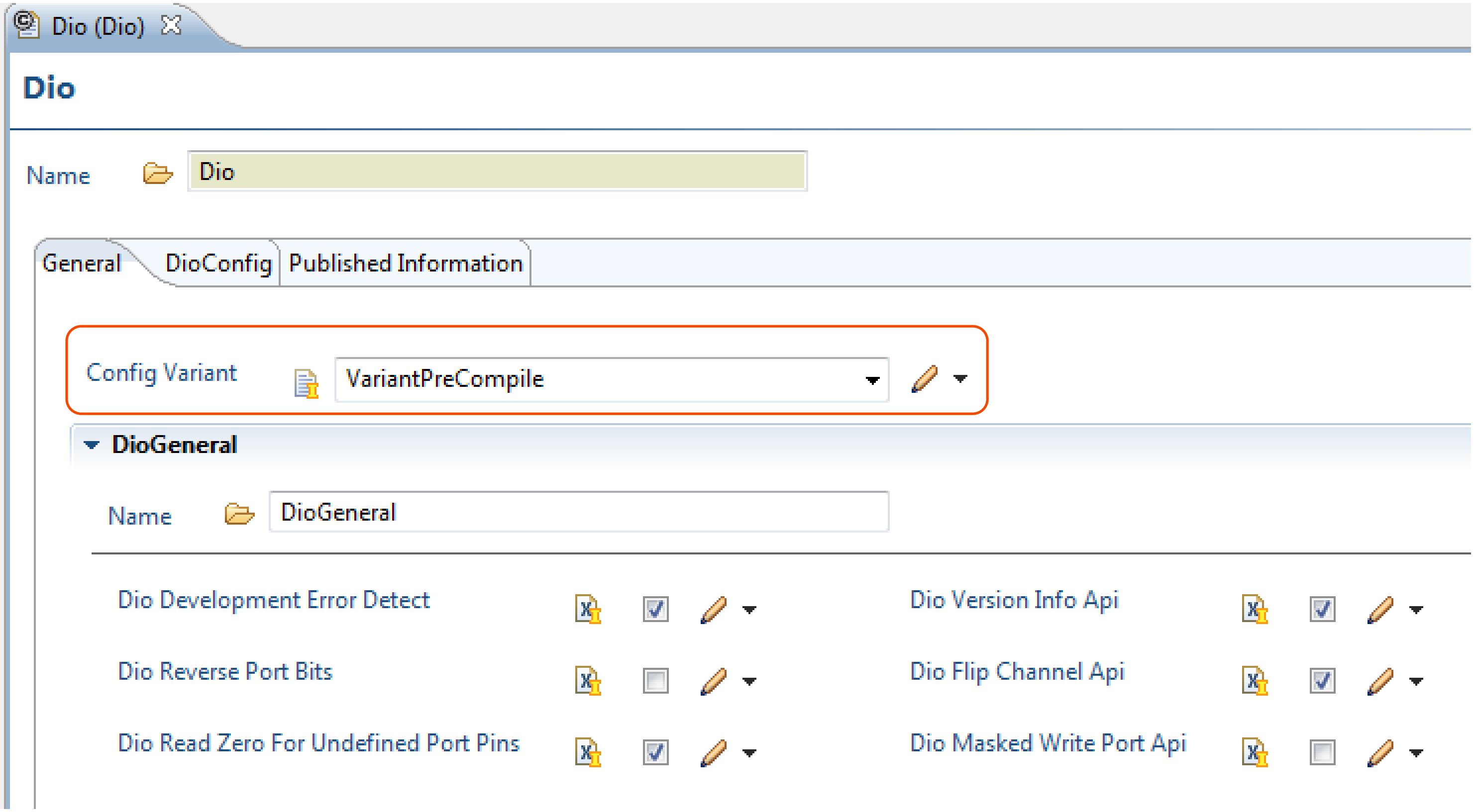
PORT Module Configuration

- PortPin configuration



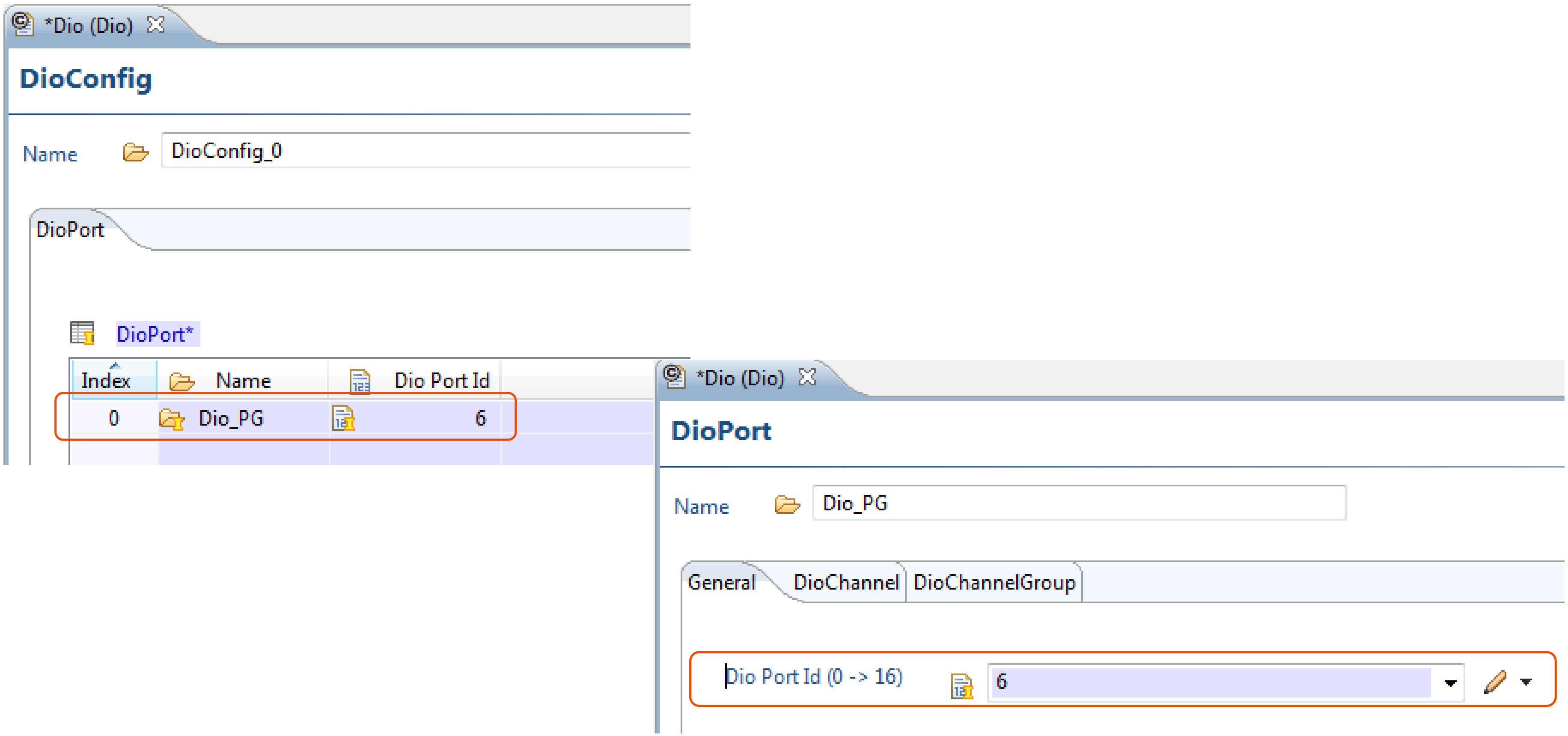
DIO Module Configuration

- ***Config Variant***



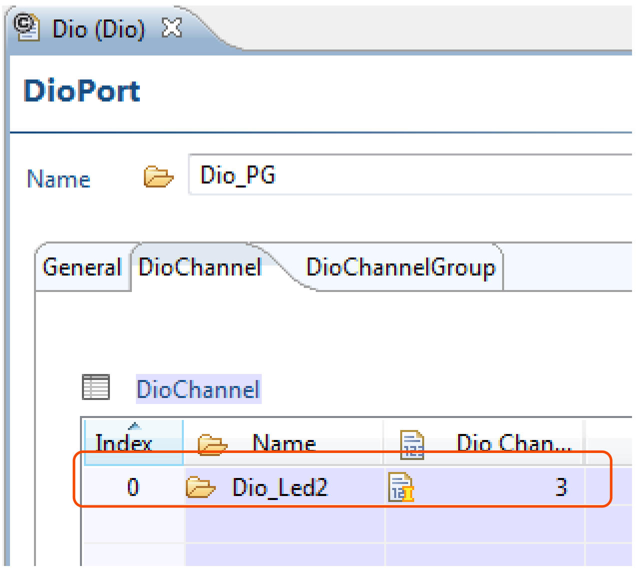
DIO Module Configuration

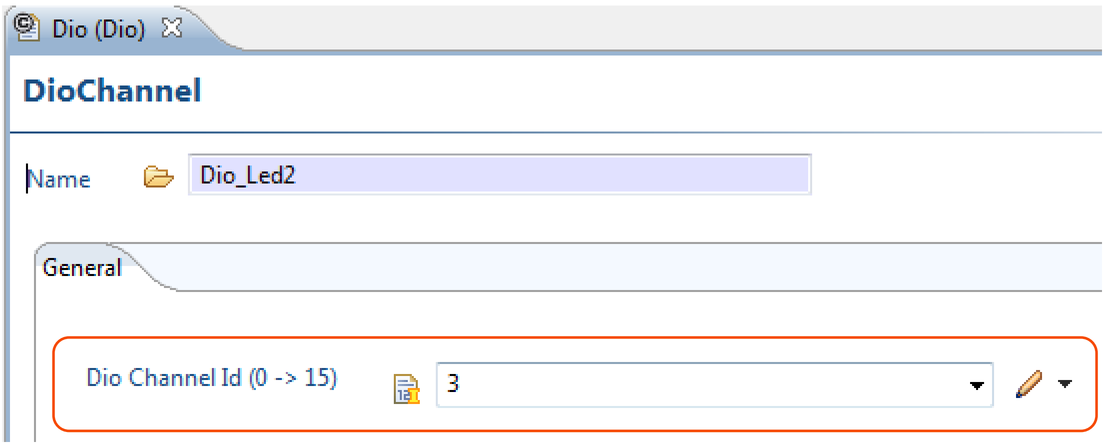
- ***DioPort*** and ***DioPortId***



DIO Module Configuration

- ***DioChannel*** and ***DioChannel*** configuration



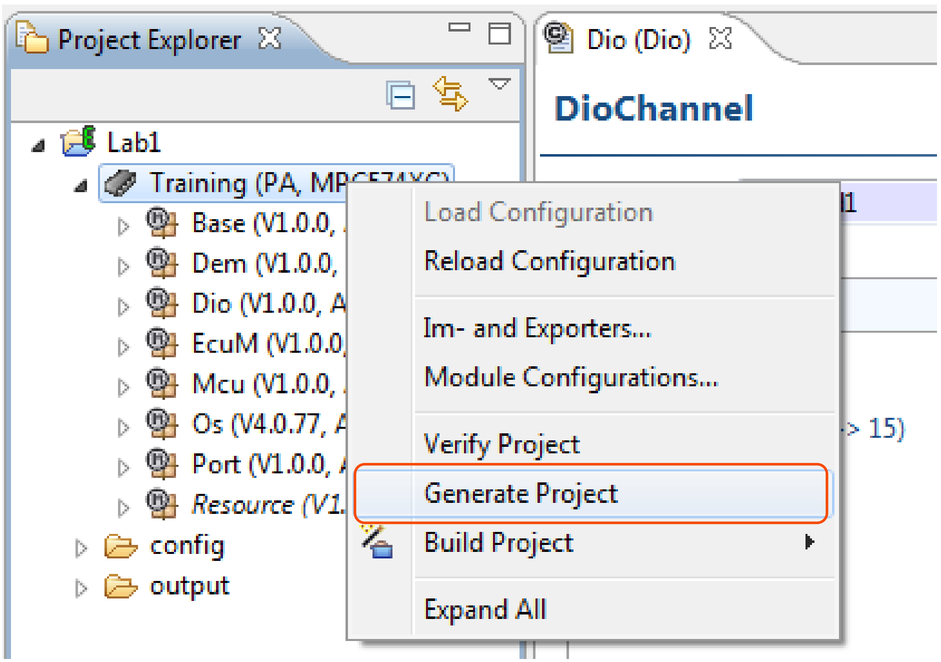


Code generation

- ***Objective***: Generate configuration data

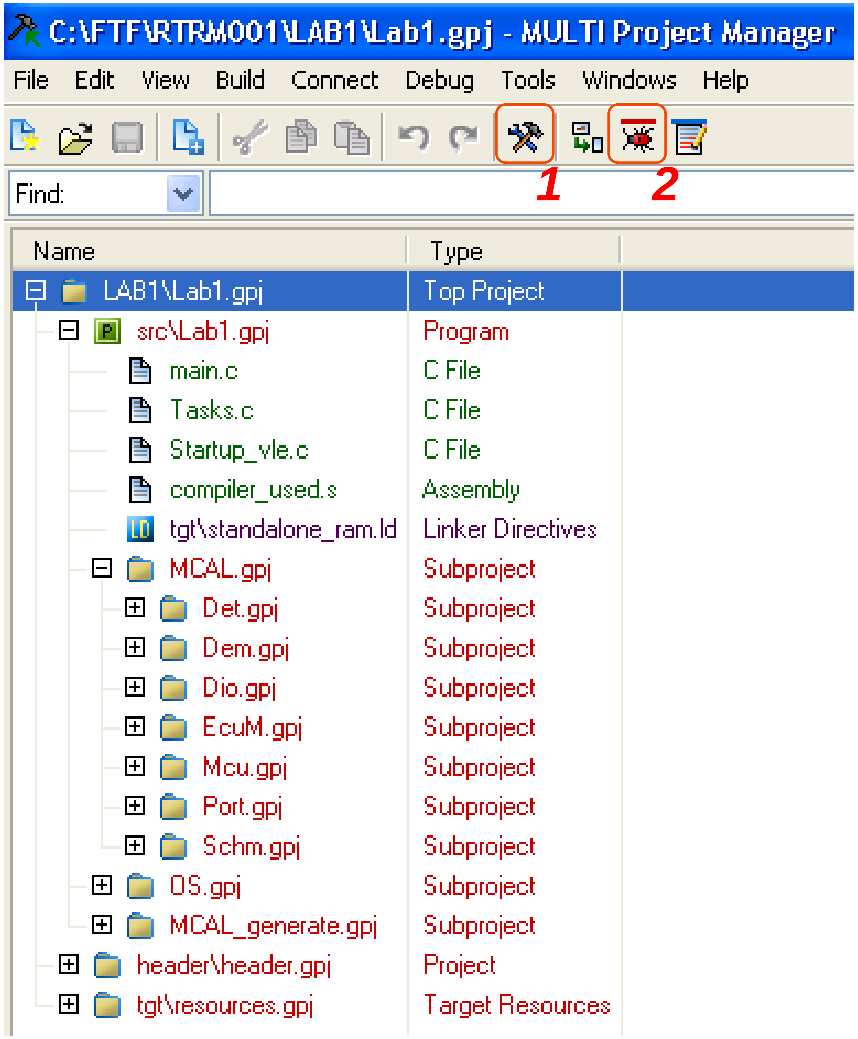
    *Right click on **Training** -> select **Generate Project***

Note: make sure that NO ERROR is reported ***to Error Log*** Window



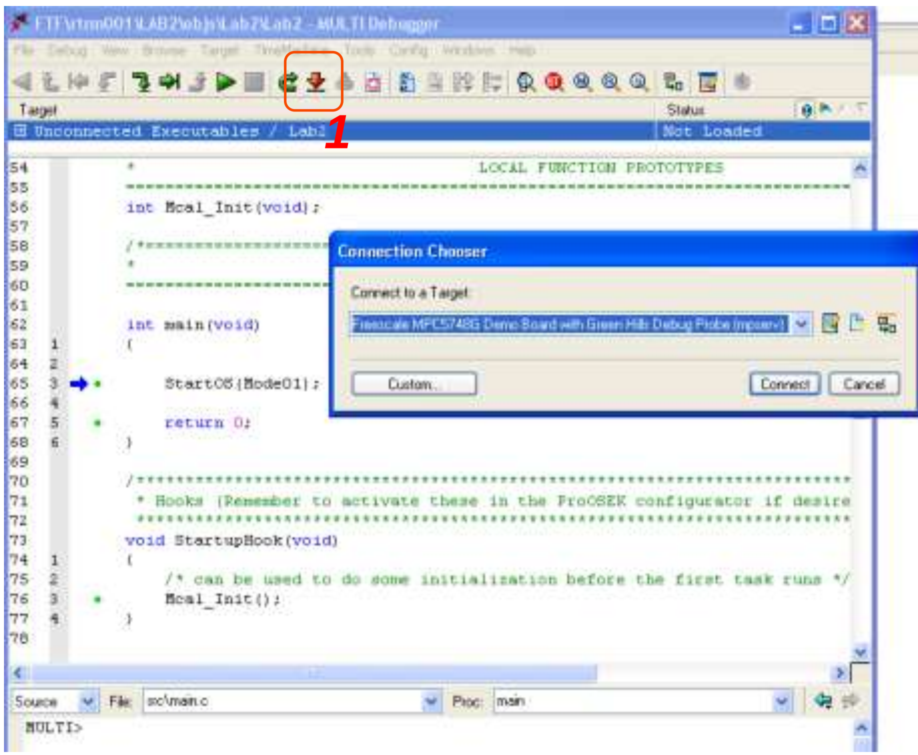
Code Compilation

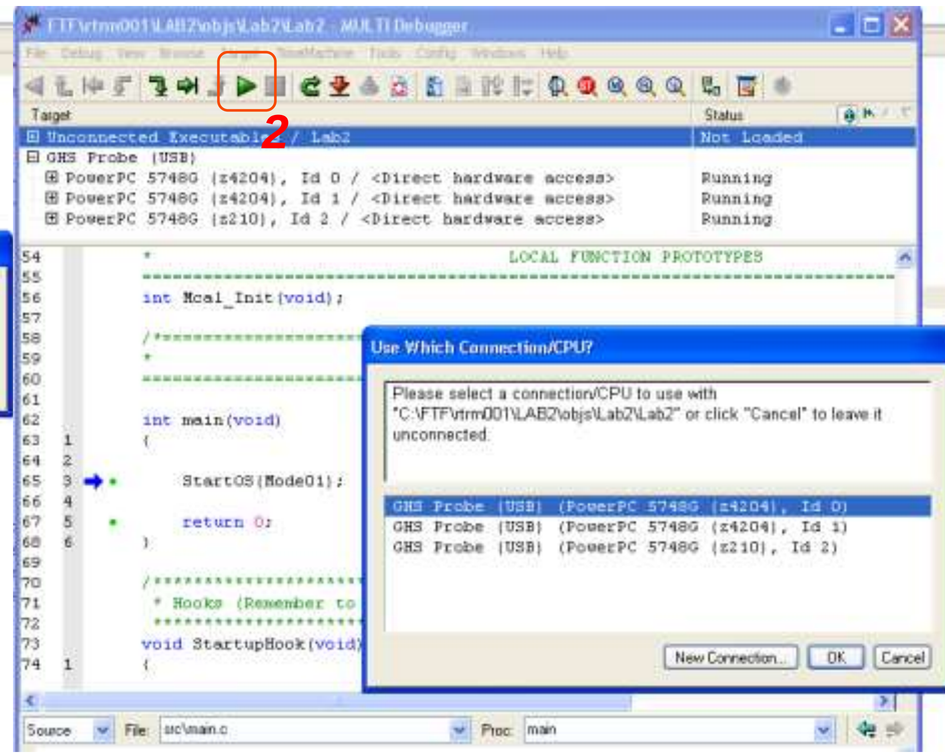
1. Open GreenHills Project from Desktop/GHS_Projects/Lab1.gpj
2. Build the project by clicking on ***1***
3. Lauch the debugger application by clicking on ***2***



Debug and Run the Code

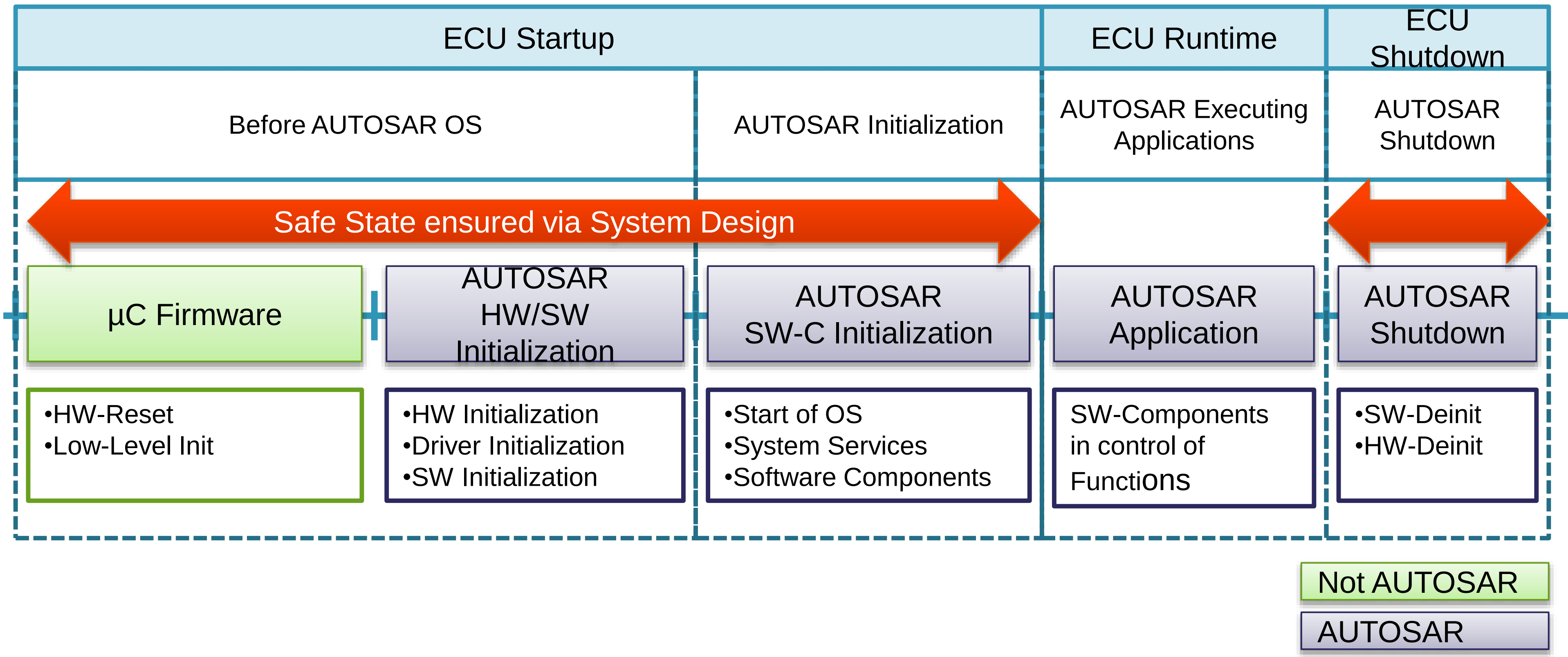
- Download the code by clicking on ***1*** and then **Connect** to the target
- **Select** GHS Probe (USB) (PowerPC 5748G (z4204), Id 0), then press **Ok**
- Run the code by clicking on ***2***





- **Result:** LED2 start blinking with a 1 sec period

AUTOSAR RunTime Application Flow



Lab2 Dimming LED

- **Objective**
  - Implementing **ADC** reads and **PWM** changes with AUTOSAR MCAL in context of AUTOSAR OS
  - Get familiar with AutoSAR OS

- **Environment**
  - AutoSAR MCAL and AutoSAR OS v4.0
  - Tool: Elektrobit tresos Studio 2014.2.1
  - Compiler: GreenHills for PPC
  - Debugger: GreenHills Probes
  - Hardware: MPC5748G Evaluation Board

- **Functional description**
  - The AutoSAR BSW modules Mcu,Dio, Port, Adc, Pwm Os, EcuM, RTE are applied to build an application which **toggles one LED** every second and **dimms another LED**

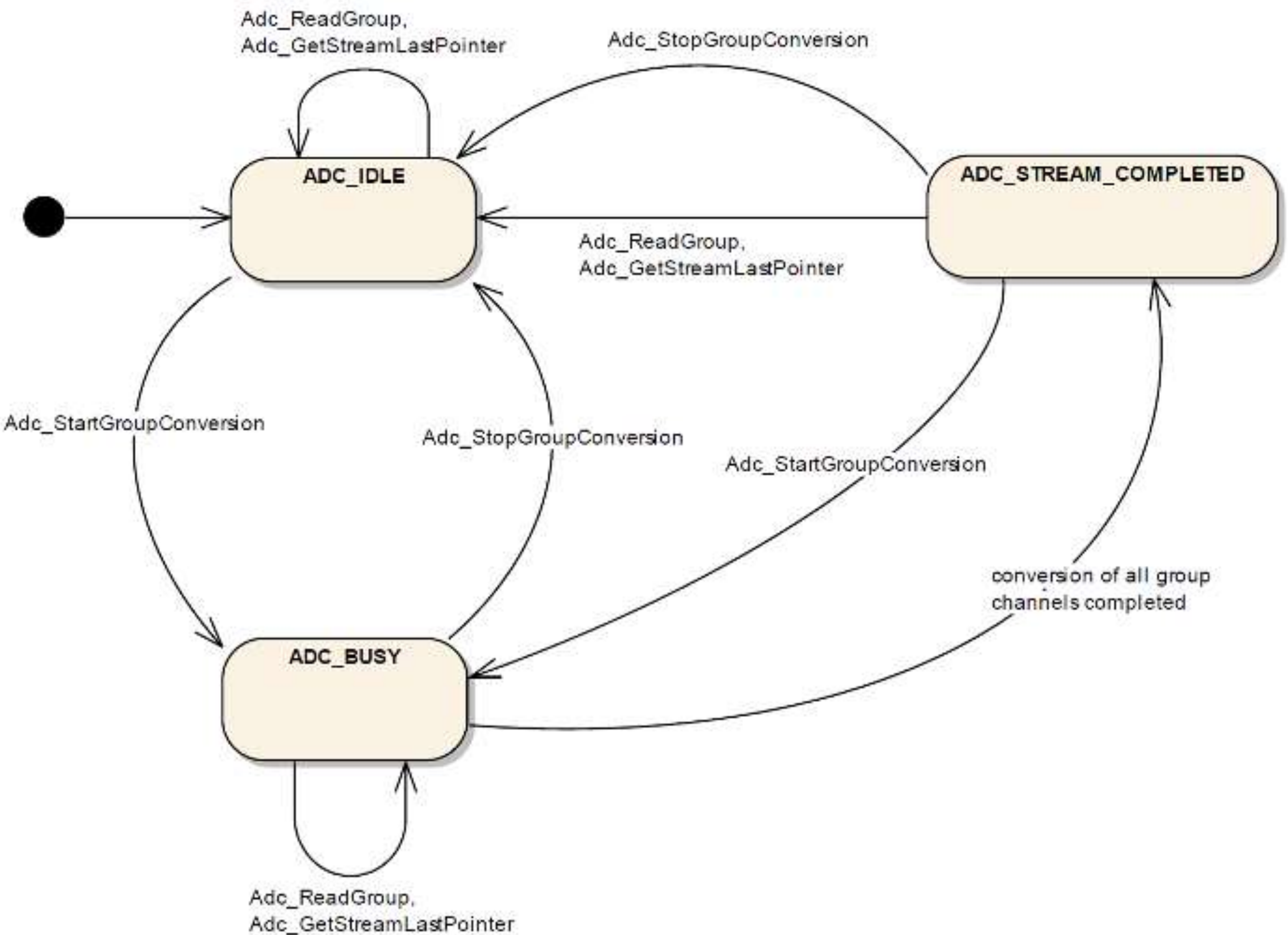
ADC Driver: Functional Overview

- **Adc Channel** represents a ADC entity bound to one port pin
  - NO own RAM buffer

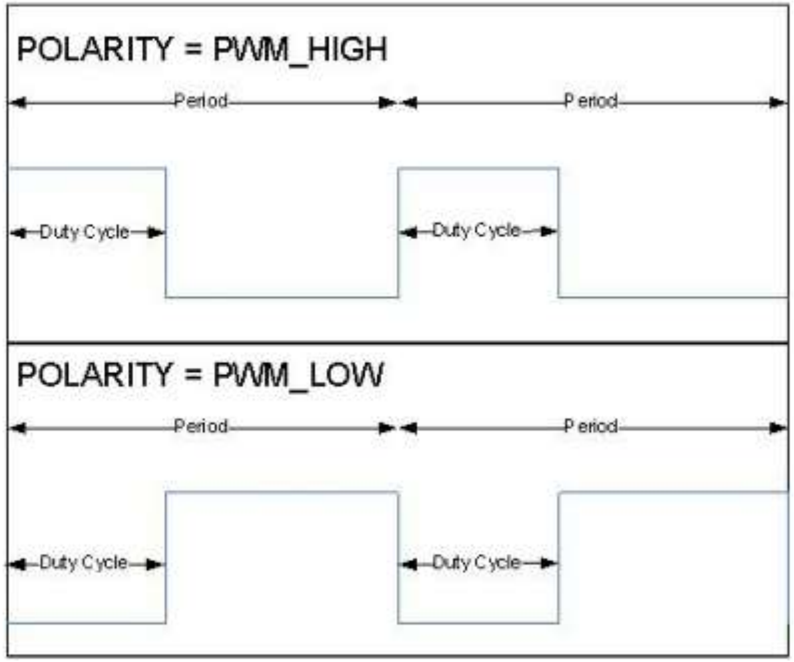
- **Adc Channel Group**
  - A group of Adc Channels linked to the same hardware unit
  - Only groups can be triggered for conversion
  - Adc driver module internally implements a state machine for each group

- **Conversion Modes**
  - **One Shot**: the conversion of an ADC channel group is performed once after a trigger (software or hardware) and the result is written to the assigned buffer
  - **Continous**: the conversions is repeated for each ADC channel in an ADC channel group

ADC Driver — Channel Group State Machine
One Shot / Software Trigger / Single Access



PWM Driver: Functional Overview



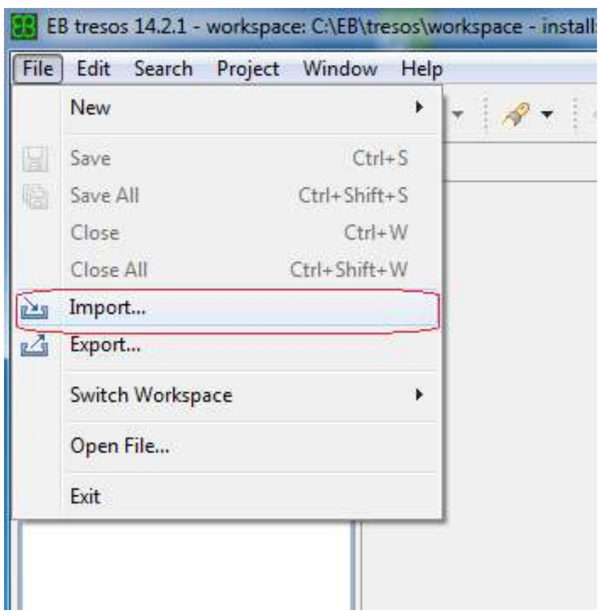
- Each PWM channel corresponds to a hardware PWM on the device
- Polarity
  - A parameter `PwmPolarity` specifies the pin output level for each channel for dutycycle and off-dutycycle.
- PWM duty cycle scaling
  - resolution: 16bit
  - range: 0x0000 (0%) to 0x8000 (100%)
- PWM Time Unit
  - Timing is adressed by Mcu. Pwm expects all time values expressed in ticks.
- Type of PWM channel is implementation specific (e.g. center align, left align, ...)

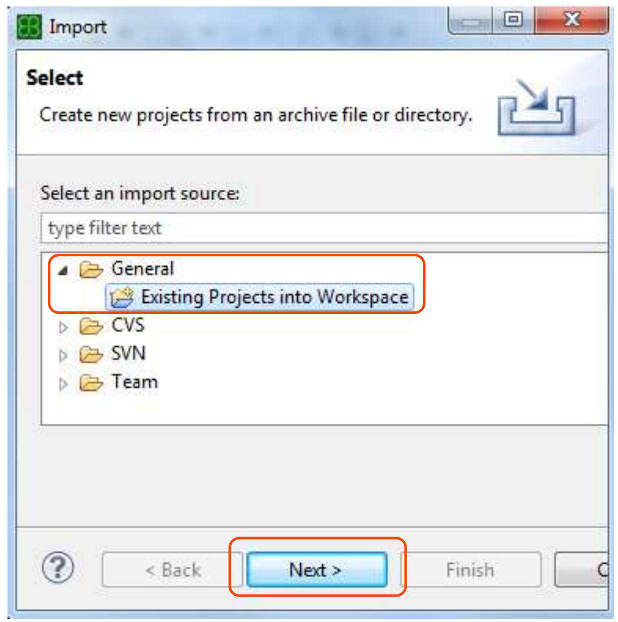
LAB2: Dimming LED

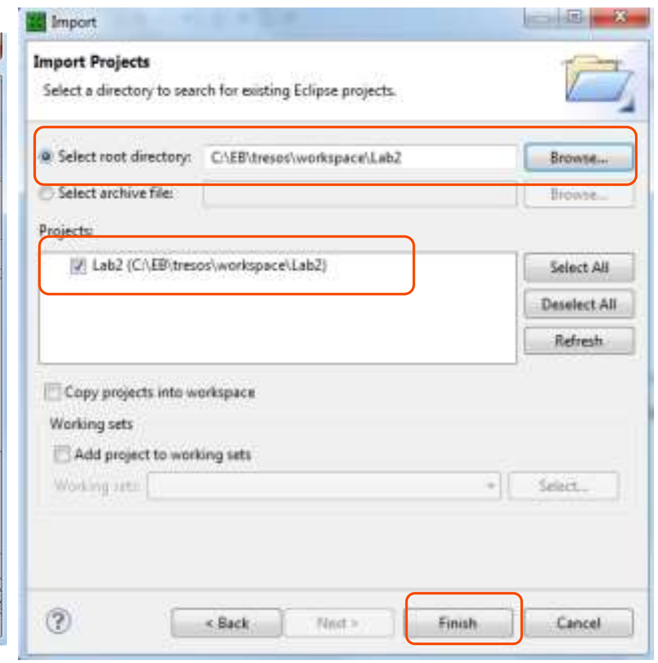
1. Opening a Tresos Project
2. Exprore PWM and ADC parameters
3. Create a new OS TASK for LED Dimming
4. Code Generation
5. GreenHills Integration
6. Compilation and Debugging

Opening a Tresos Project

1. ***File*** *->* ***Import*** *->* ***General*** *->* ***Existing Projects into Workspace*** *->* ***Select root Directory*** *->* ***Browse*** *to c:\eb\tresos\workspace* ***-> Select Lab2 -> Finish***

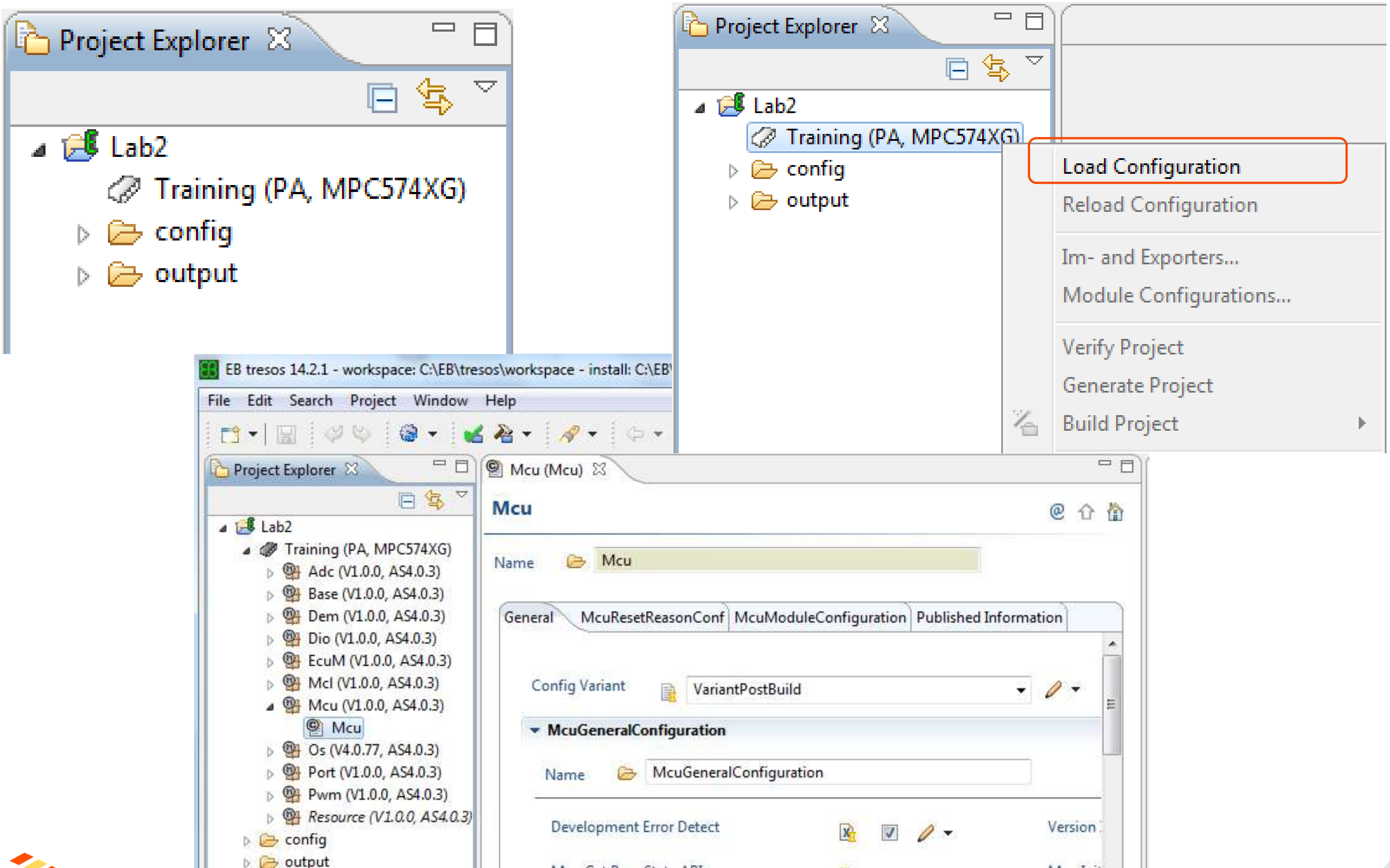






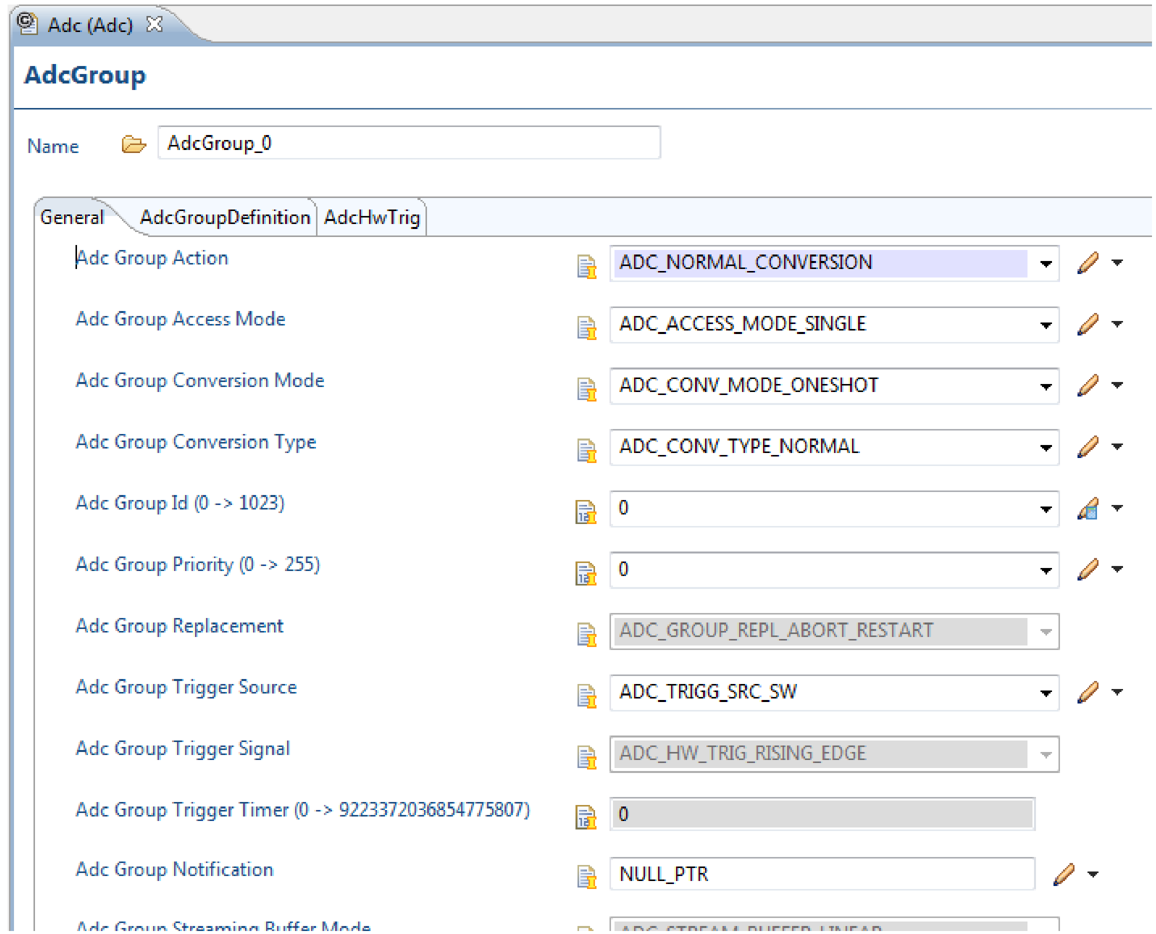
Opening a Tresos Project

2. *Right click on* ***Training*** *-> select* ***Load configuration***



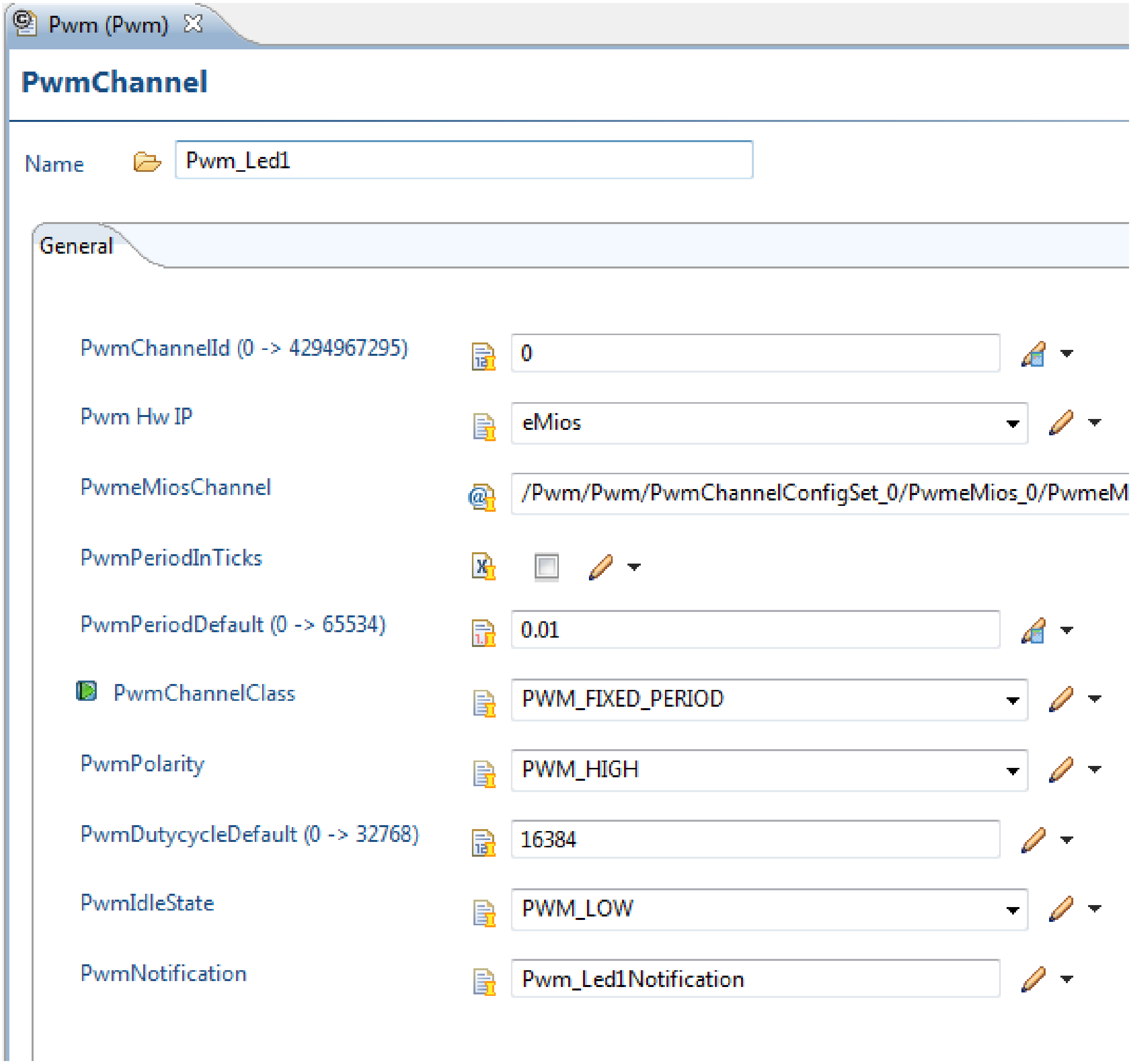
ADC Driver: Configuration Parameters Exploration

- **Adc Group**
  - **Adc Group Actions**: NORMAL CONV.
  - **Adc Conversion Mode**: ONESHOT
  - **Adc Conversion Type**: NORMAL
  - **Adc Trigger Source**: SW



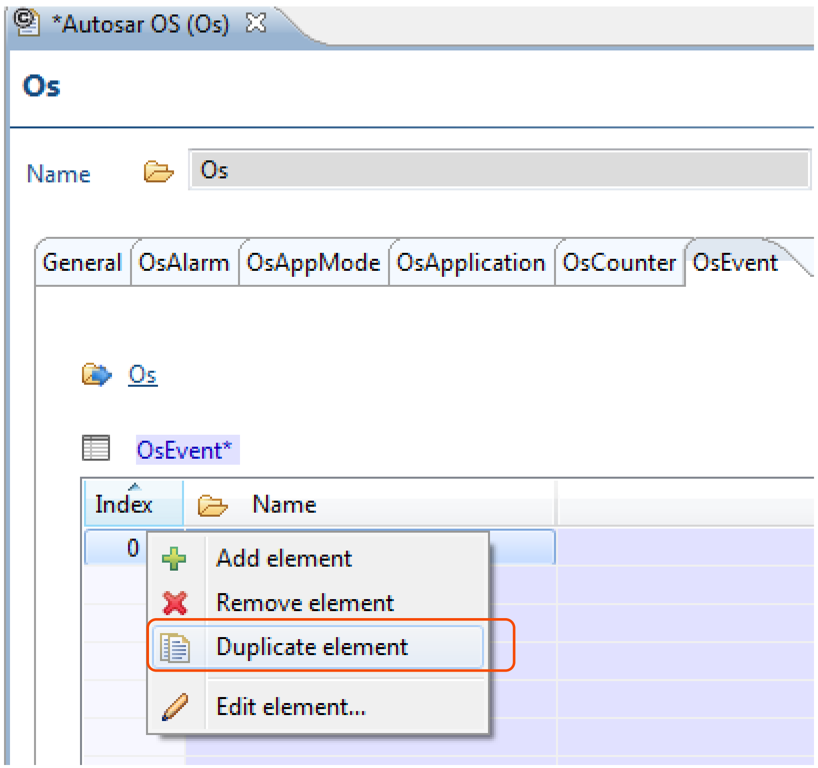
PWM Driver: Configuration Parameters Exploration

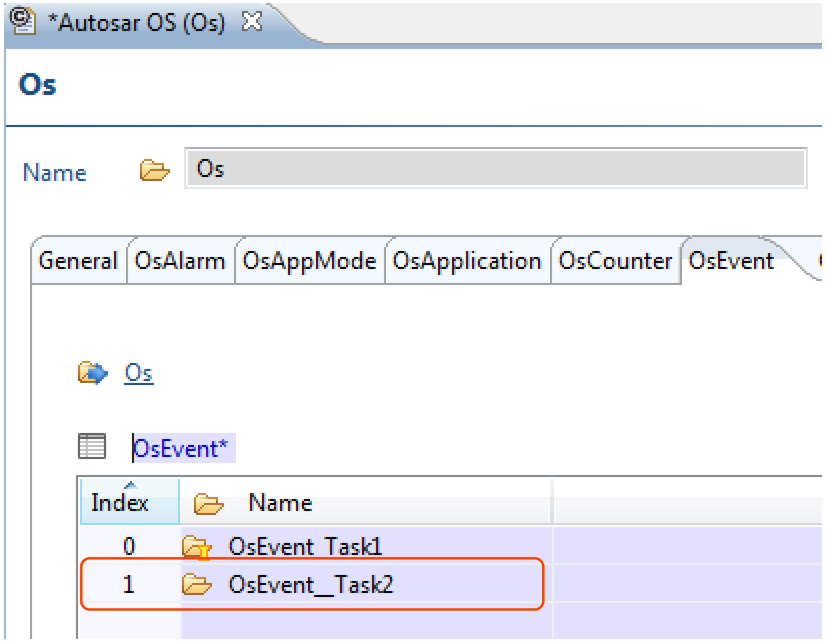
- **Pwm**
  - **Pwm Channel**: Pwm_Led1
  - **Pwm HW IP**: eMIOS
  - **Pwm Channel Class**: FIXED_PERIOD
  - **Pwm Default Period**: 0.01 ticks
  - **Pwm Default DutyCycle**: 50%



OS Config: Create a New OS Event for LED Dimming

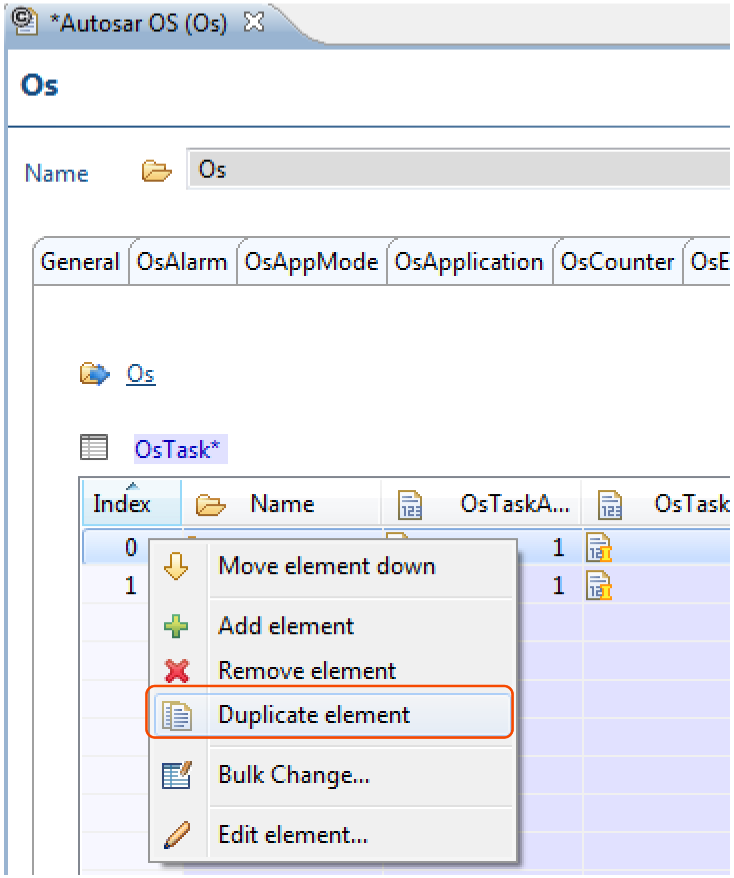
1. Go to the **OsEvent** Tab -> Right **Click on OsEvent_Task1** and select ***Duplicate Element***
2. Rename the new event to ***OsEvent_Task2***

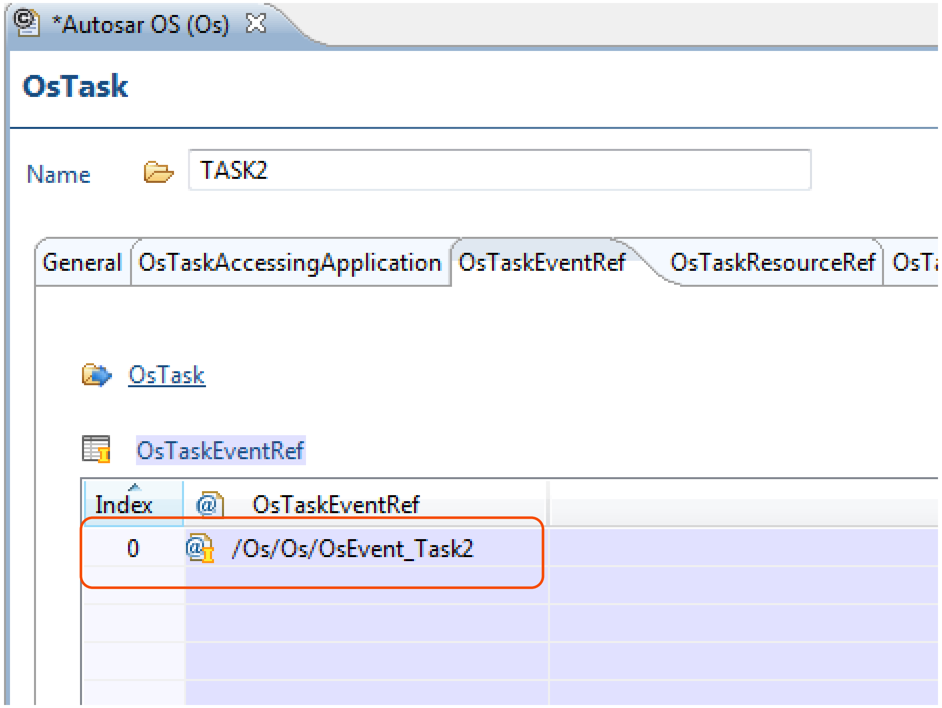




OS Config: Create a New Task for LED Dimming

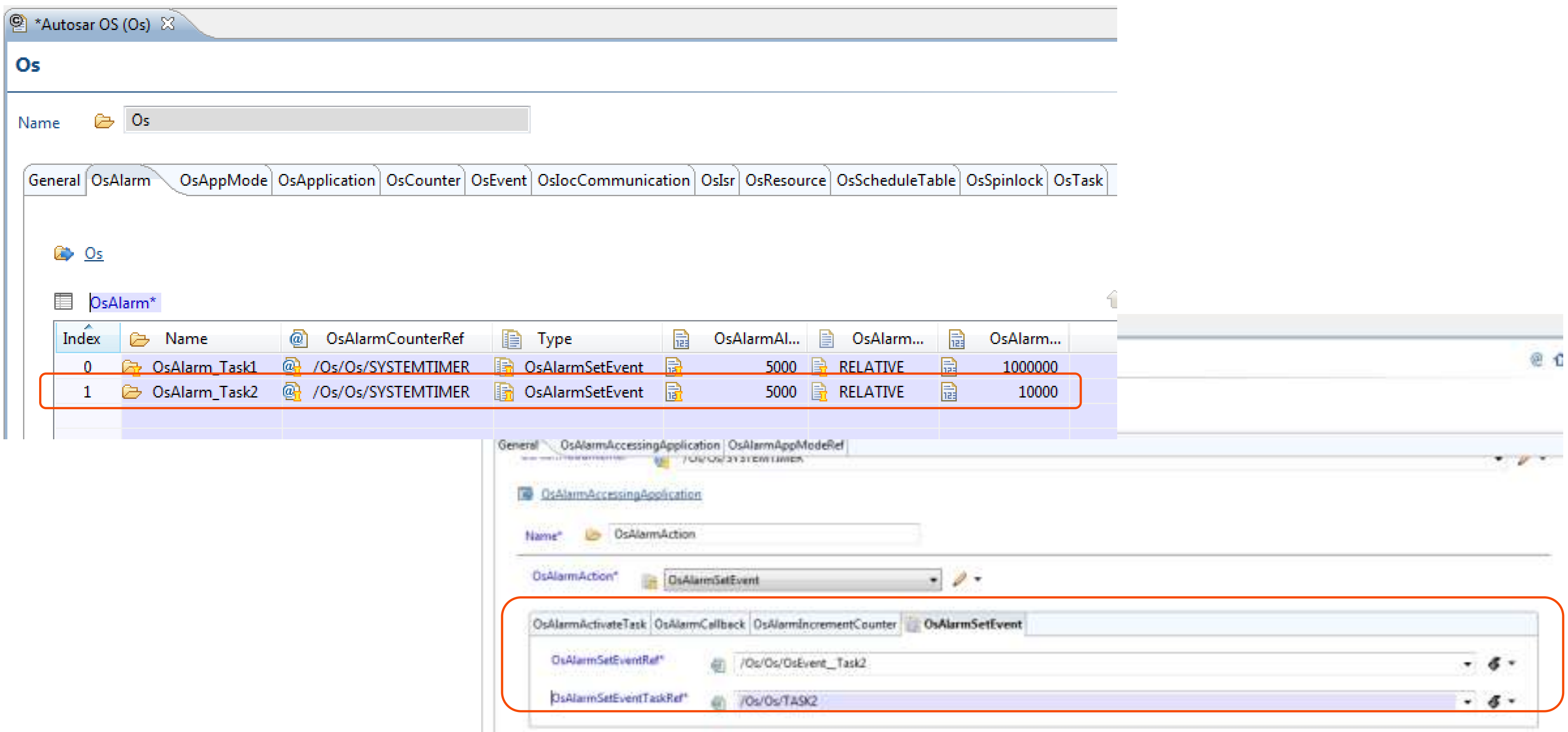
1. Go to the **OsTask** Tab -> Right **Click on TASK1** and select ***Duplicate Element***
2. Rename the new task to **TASK2** and from ***OsTaskEvent*** select ***OsEvent_Task2***





OS Config: Create a New OS Alarm for LED Dimming

1. Go to the **OsAlarm** Tab -> Right **Click on OsAlarm_Task1** and select ***Duplicate Element***
2. Rename the new event to **OsAlarm_Task2** and set the params as below

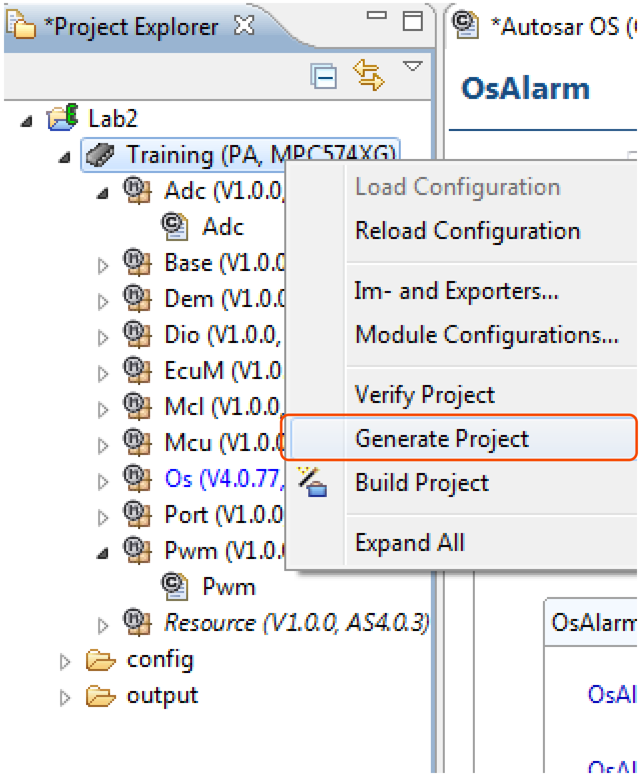


Code Generation

- ***Objective***: Generate configuration data

  *Right click on **Training** -> select **Generate Project***

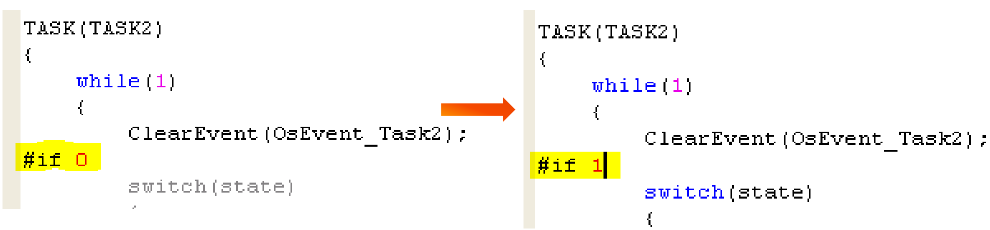
Note: make sure that NO ERROR is reported ***to Error Log*** Window



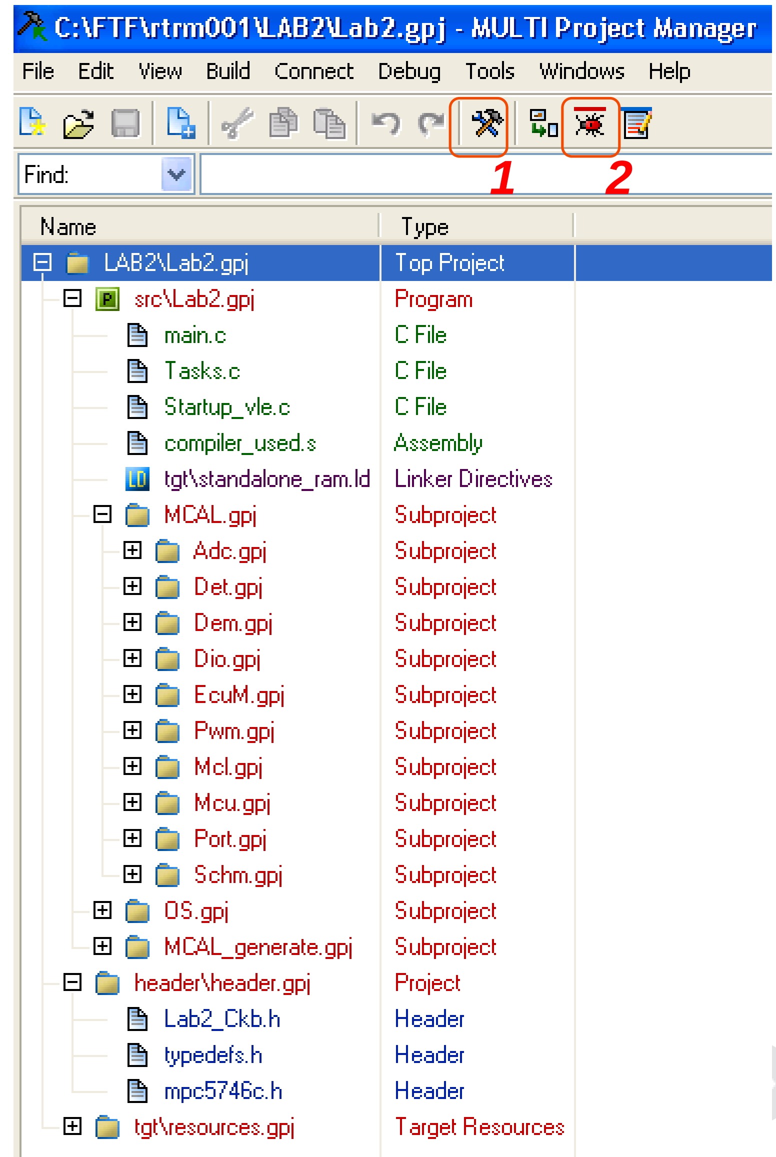
Code Compilation

1. Open GreenHills Project from Desktop/GHS_Projects/Lab2.gpj
2. Go to ***Task.c*** and ***uncomment***

TASK2 body, then save the changes

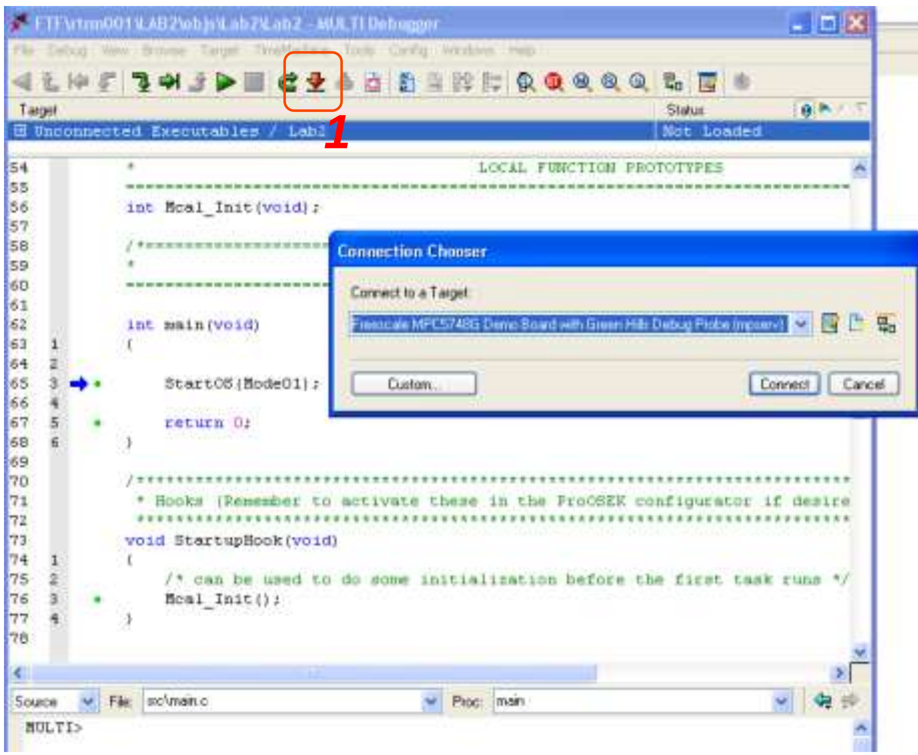


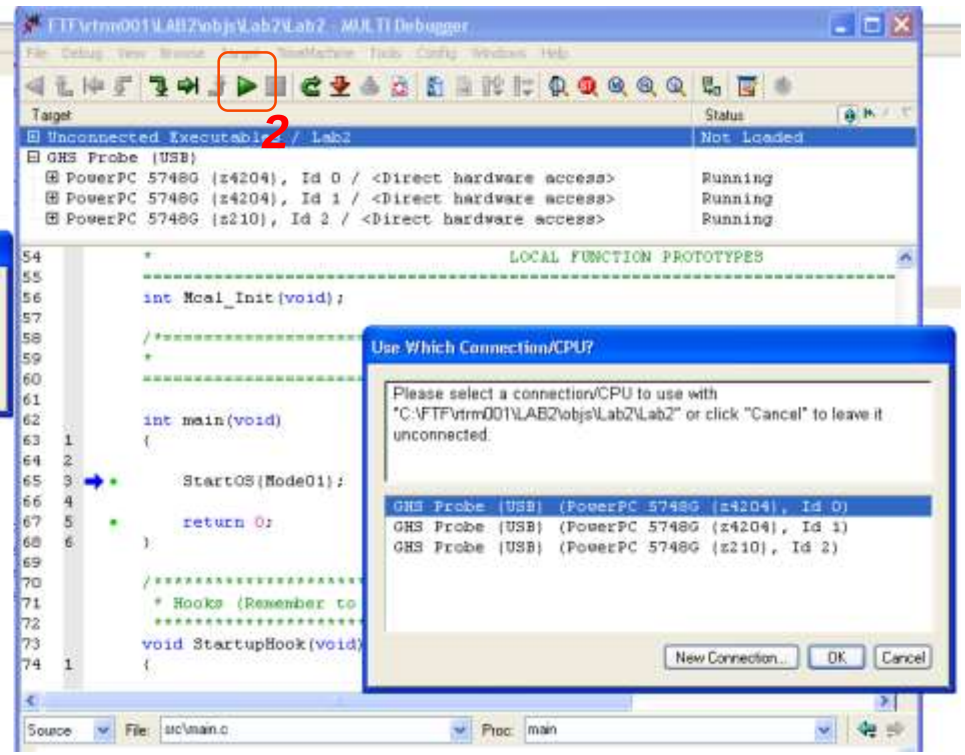
3. Build the project by clicking on ***1***
4. Debug your project clicking on ***2***



Debug and Run the Code

- Download the code by clicking on ***1*** and then **Connect** to the target
- **Select** GHS Probe (USB) (PowerPC 5748G (z4204), Id 0), then press **Ok**
- Run the code by clicking on ***2***





- Turn the potentiometer and see the LED1 dimming www.Freescale.com

© 2015 Freescale Semiconductor, Inc. | ***External Use***