AUTOSAR MCAL for i.MX 8

Salam Zeidan

Sr. Project Manager

MICR Advanced Technologies

October 2018 | AMF-AUT-T2976

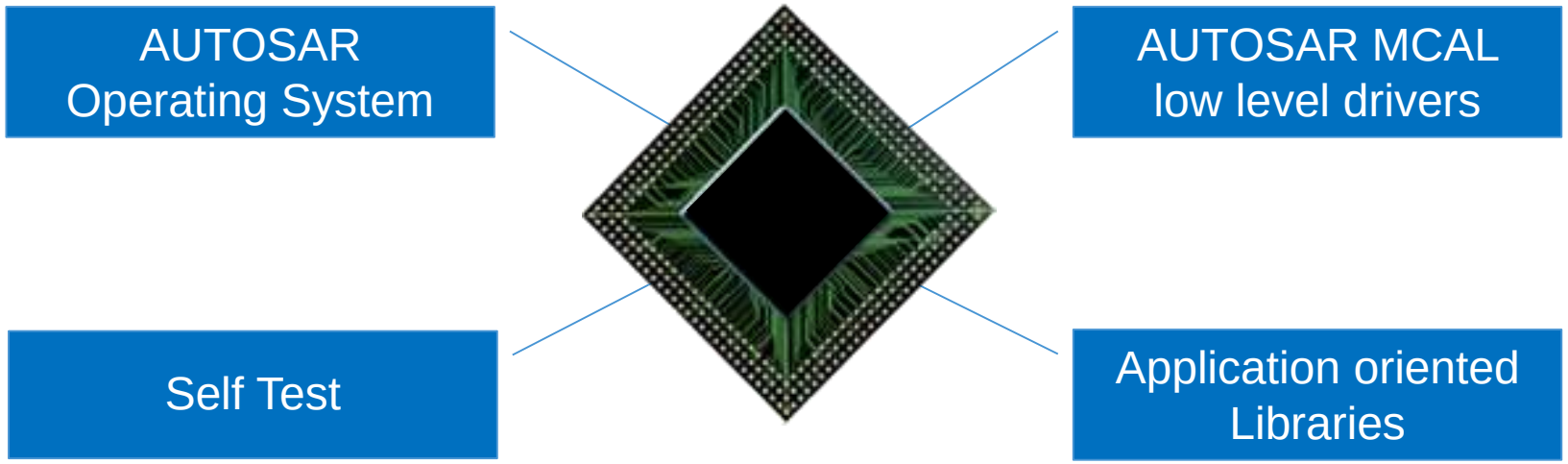
NXP | CONNECTS

Agenda

- General Information
- i.MX AUTOSAR MCALs:
  - Implementation For All Derivatives In i.MX Families
  - Example Startup
  - Configuration Tool View
  - Microcontroller Drivers
  - I/O Drivers
  - Communications Drivers
  - Memory Drivers
  - Complex Drivers

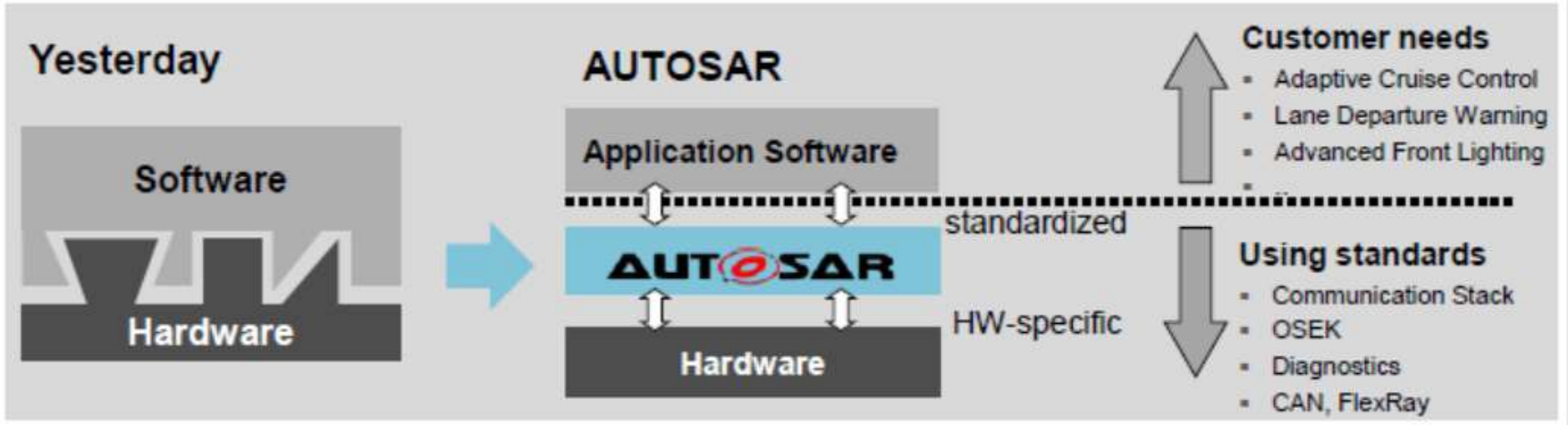
General Information

i.MX AUTOSAR Solution



AUTOSAR – Global Automotive SW Standard

AUTOSAR (AUTomotive Open System ARchitecture) Org. aims to improve complexity management of integrated E/E architectures through increased reuse and exchangeability of SW modules between OEMs and suppliers. The essential means is the standardization of the software architecture of ECUs.



- Hardware and software is widely independent of each other.
- Development can be de-coupled by horizontal layers. Reduces development time and costs.
- Reuse of software enabled at OEM and at suppliers. Enhances quality and efficiency.

AUTOSAR – NXP AUTOMOTIVE SW

PREMIUM PARTNER

Design and use the AUTOSAR standards

As a premium member of the AUTOSAR partnership NXP continues to develop its AUTOSAR SW offering across its product lines for use in automotive applications

AUTOSAR Documents

- Released AUTOSAR documents can be found at www.autosar.org

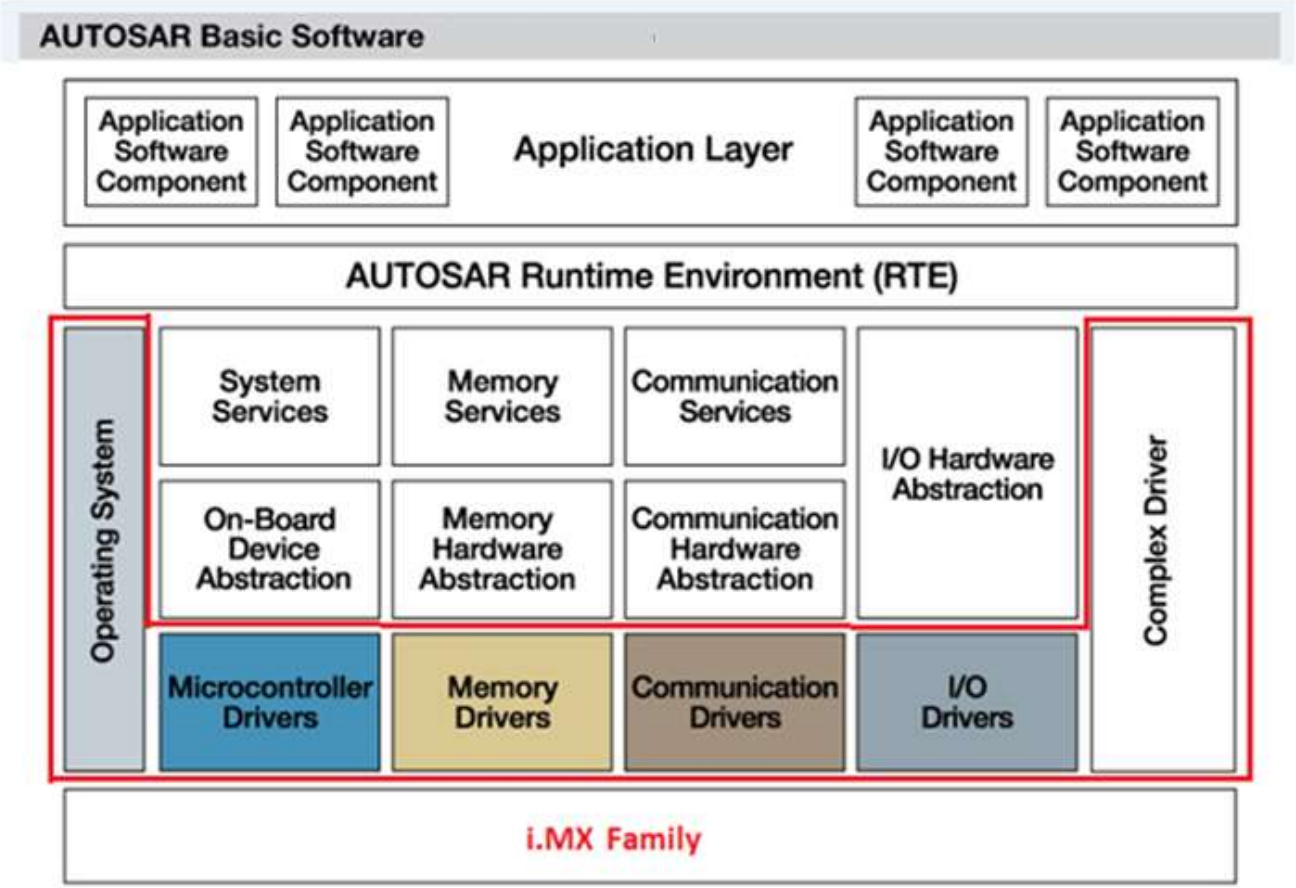
- Two documents exist for each BSW module:
  - SRS: Software requirement specification
  - SWS: Software Specification

- The **SRS** describes requirements, that must be fulfilled by a Basic
  - Software Module (BSW).

- The **SWS**(Software Specification) contains the most detailed information for each Basic Software Module

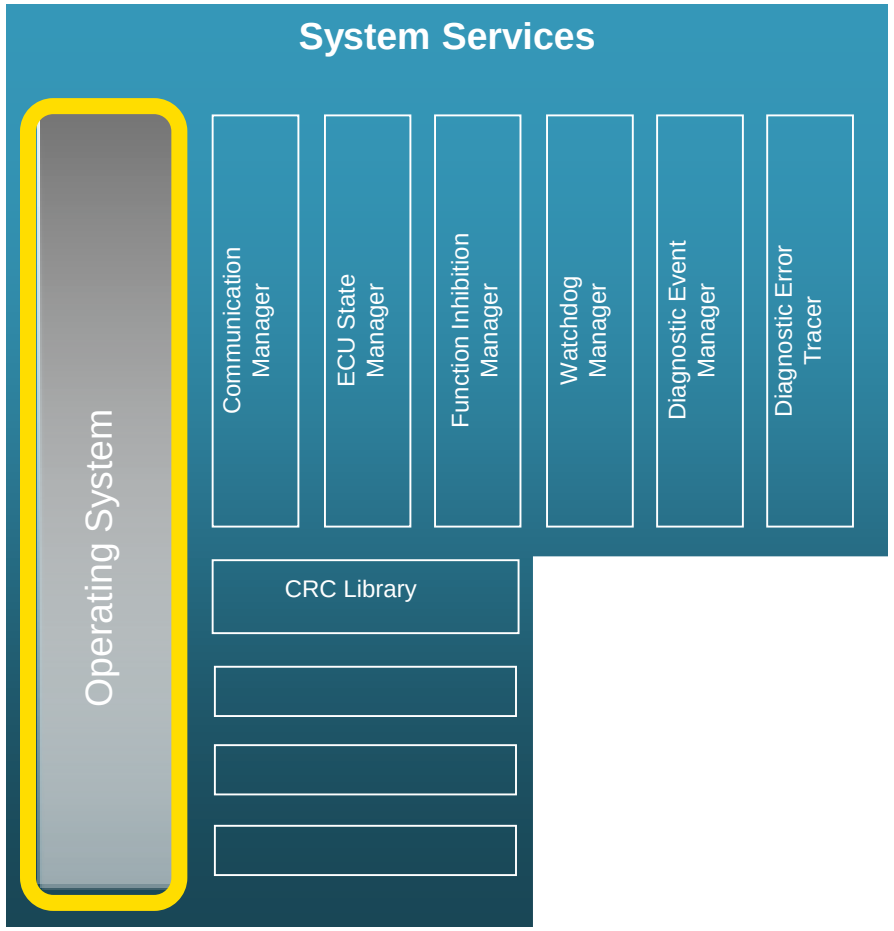
i.MX AUTOSAR Product

i.MX AUTOSAR OS, MCALs, and Complex Drivers



i.MX AUTOSAR OS

- High-quality production-intent software
- SPICE compliant development process
- Configurable with EB tresos® Studio, or other AUTOSAR compliant configuration tools
- Compliant to versions 4.x of the AUTOSAR standard
- Supports industry standard compilers
- Offered as Engineering Services per customer Request
- Available in Scalability Classes 1



AUTOSAR Software Release Framework Overview

- BETA Release (Select Customers)
  - Includes all MCAL Drivers (Feature Complete) fully verified and documented
  - Includes Integration Testing
  - Complete Quality Package

- Release to Market Candidate – RTMC (All Customers)
  - Beta criteria +
  - 100% Decision Coverage for one configuration
  - No open S1 and S2 defects

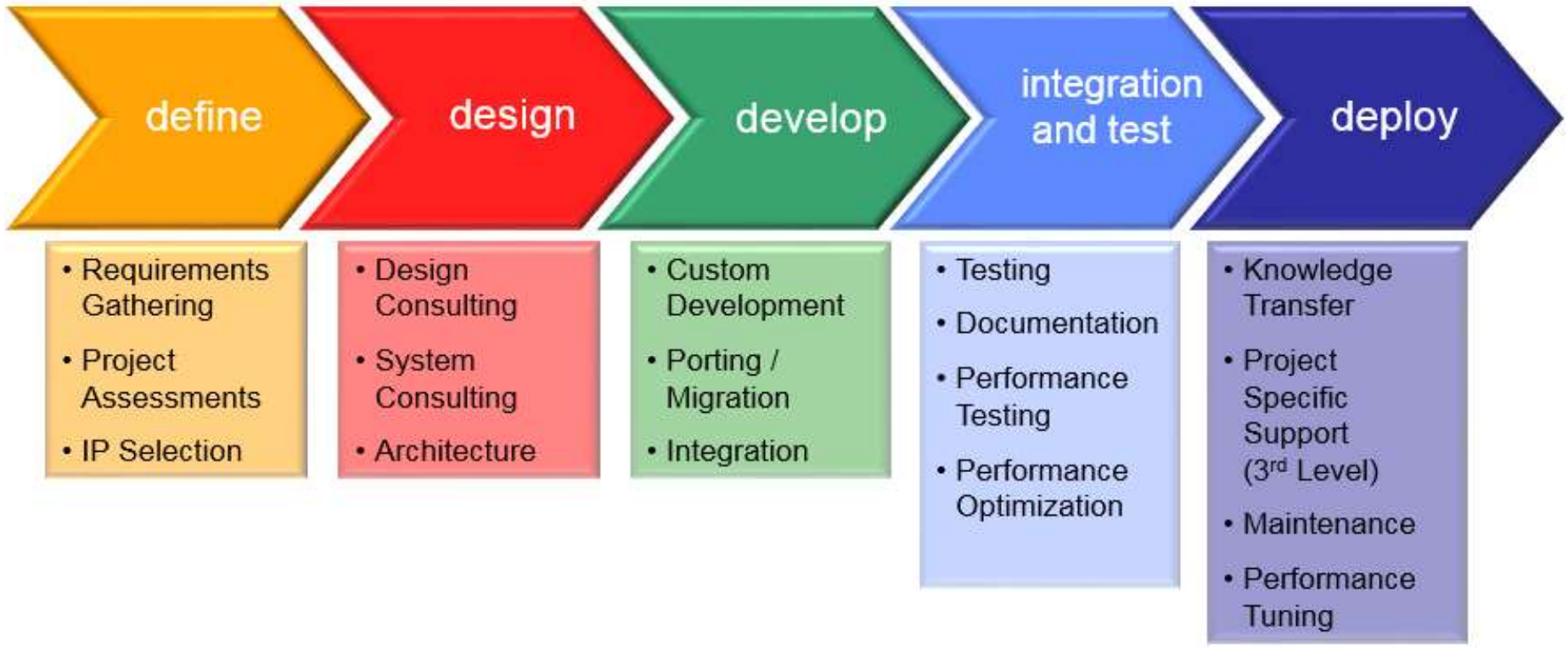
Each ASR version and i.MX derivative will have an individual release

AUTOSAR Software Release Framework Overview

- Customer Compiler Tests
  - Current MCALs tested using GHS ARM Compiler
  - Specific cutsomer compiler version and settings will be provide through enineering services as a Frozen Branch Release (FBR)

- Customer Specific HW Module MCALs Tests
  - Specific for the customer i.MX based HW module

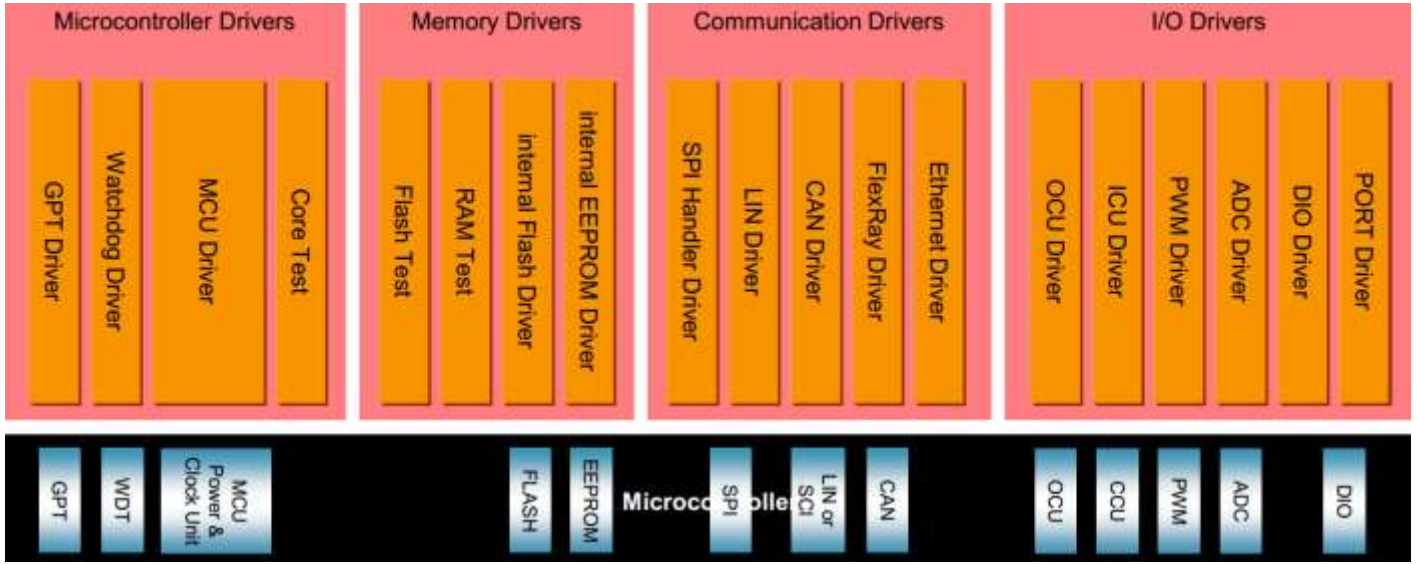
Software Life Cycle Methodologies & Quality: Engineering Discipline



AUTOSAR MCALs for i.MX Family

i.MX AUTOSAR MCALs Product

- Autosar 4.x MCAL: Tested running from RAM using NXP MEK/EVB
- FlexRay, and WDG-External can be provided as a additional Complex Drivers Services
- Core Test, Flash Test, or RAM Tests are not provided
- All components configurable in any AUTOSAR-compliant configuration tool
- Configuration Tool EB tresos StudioTM and plugins are part of the product
- MCALs also been tested using Vector DaVinci Configuration Tool



AUTOSAR Software i.MX6 RTM Available Releases

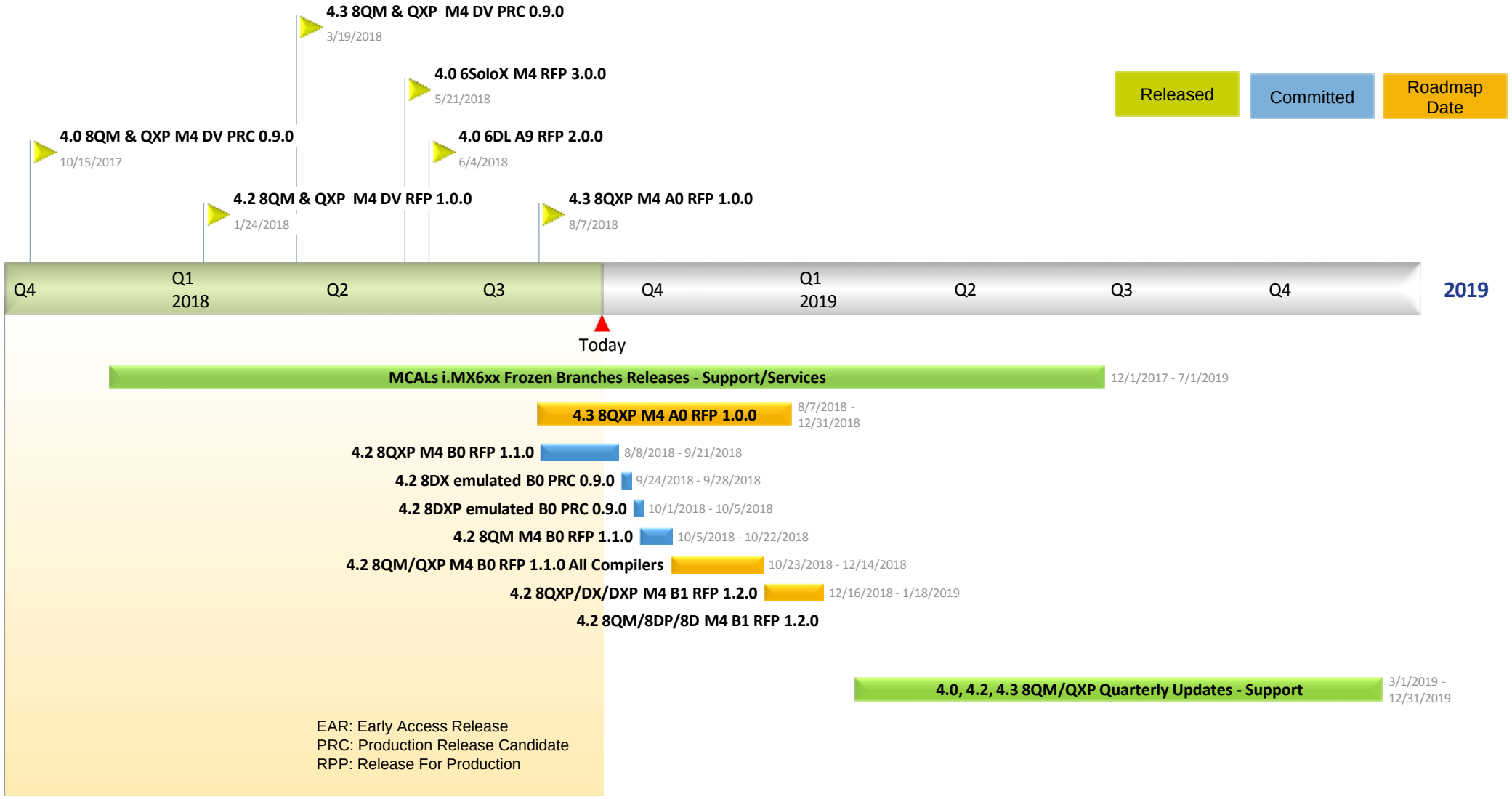
MCAL - i.MX 6Family							
Processor		i.MX6Quad	i.MX6Dual	i.MX 6SoloX		i.MX 6DualLite	i.MX 6Solo
Core		A9_Core0	A9_Core0	M4	A9	A9_Core0	A9
MCALs							
Autosar Revisions	3.x						
	4.0.3						
	4.1.3						
	4.2.1						
Complex Drivers							
Compilers	Diab						
	GHS						
	ARM DS5						
	Others						
Functional Saftey ISO26262	ASIL A						
	ASIL B			Some			

Legend	
	Available
	Available - Within 8weeks from Receiving a PO. Per Silicon Availability
	Not Available-Will be handled through Engineering Services Specific Project Quote

AUTOSAR Software i.MX8 RTM Planned Releases

MCAL - i.MX 8Family								
Sub Family		I.MX8			i.MX8 X			
Part/Derivative		i.MX8QM	i.MX8QP	i.MX8DM	i.MX8QXP	i.MX8DXP	i.MX8DX	
Core		M4	M4	M4	M4	M4	M4	
I/O Modules	Port	Yes	Yes	Yes	Yes	Yes	Yes	
	Dio	Yes	Yes	Yes	Yes	Yes	Yes	
	Icu	Yes	Yes	Yes	Yes	Yes	Yes	
	Pwm	Yes	Yes	Yes	Yes	Yes	Yes	
	Adc	Yes	Yes	Yes	Yes	Yes	Yes	
Comm Modules	Spi	Yes	Yes	Yes	Yes	Yes	Yes	
	Can	Yes	Yes	Yes	Yes	Yes	Yes	
	Lin	Yes	Yes	Yes	Yes	Yes	Yes	
	Eth	Yes	Yes	Yes	Yes	Yes	Yes	
Mcu Modules	Mcu	Yes	Yes	Yes	Yes	Yes	Yes	
	Gpt	Yes	Yes	Yes	Yes	Yes	Yes	
	Fls FlexSPI	Yes	Yes	Yes	Yes	Yes	Yes	
	Wdg Int.	Yes	Yes	Yes	Yes	Yes	Yes	
Complex Drivers	I2C	Yes	Yes	Yes	Yes	Yes	Yes	
Compilers	GHS	Yes	Yes	Yes	Yes	Yes	Yes	
	GCC	Yes	Yes	Yes	Yes	Yes	Yes	
	DS5	Yes	Yes	Yes	Yes	Yes	Yes	
	Diab	Yes	Yes	Yes	Yes	Yes	Yes	
Autosar Revisions	4	Yes	Yes	Yes	Yes	Yes	Yes	
	4.2	Yes	Yes	Yes	Yes	Yes	Yes	
	4.3	Yes	Yes	Yes	Yes	Yes	Yes	
Safety	ASIL B	Yes	Yes	Yes	Yes	Yes	Yes	

i.MX8 AUTOSAR MCAL Roadmap



Licensing Options

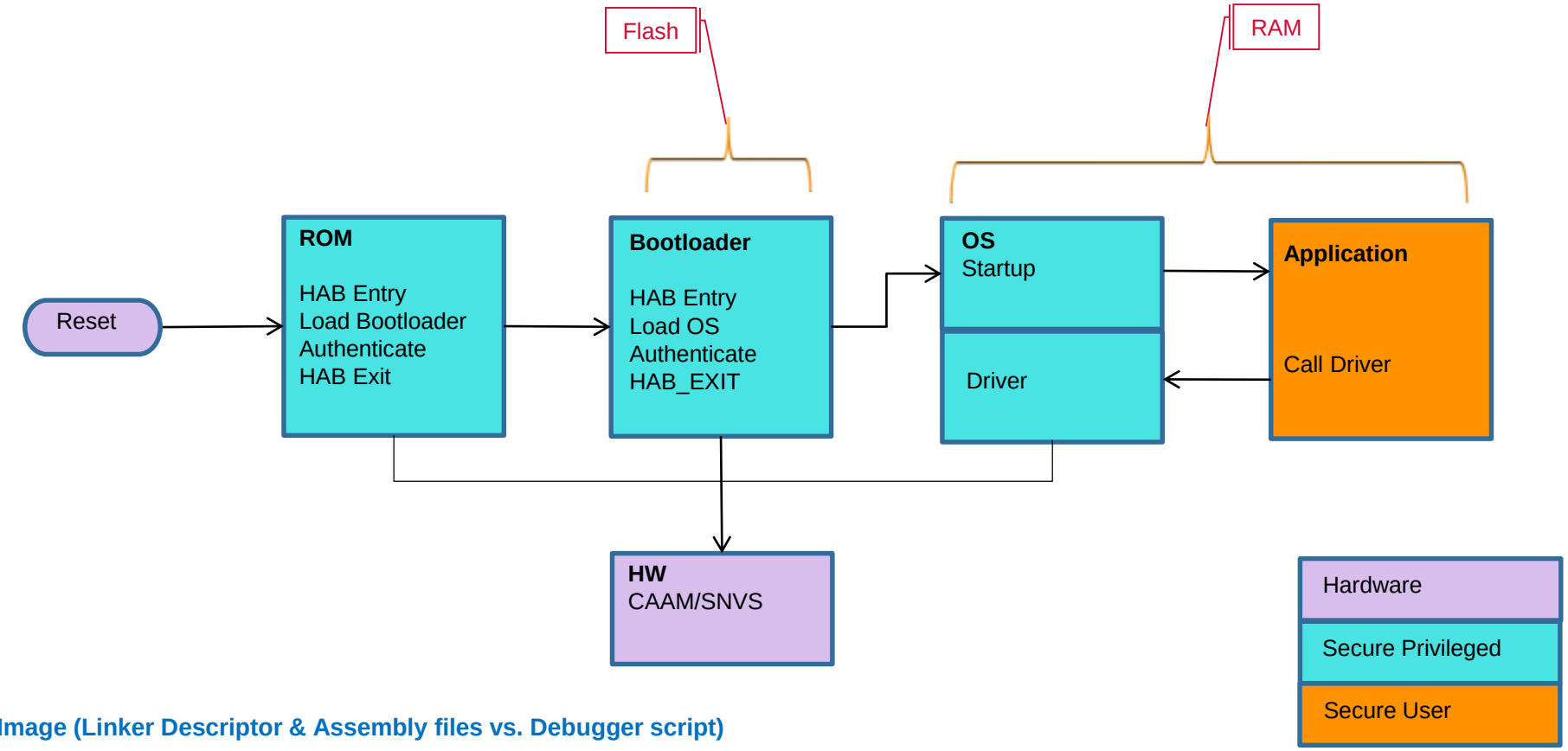
License	Description	Support	Upgrade options
Project License	One NXP Target Product/Core (ex. i.MX6SoloX M4) tested on NXP EVB HW running in RAM only for one Customer Target Project. Specific MCAL drivers List.	1 year support included, 20% of the license price year 2 and beyond.	To any higher priced production license. Additional SoC at a discount. -Additional Compiler test -Additional MCAL drivers, complex drivers, or testing on customer HW ECU will be handled as Engineering Services.
Product Line License	One NXP Target Product only for one Customer Product Line (Clusters.) Specific MCAL drivers List.	1 year support included, 20% of the license price year 2 and beyond.	To any higher priced production license. Additional SoC at a discount.
Family Multi-Project License	One NXP Target Product Family, (ex. i.MX6 Family,), only for Customer Target Project or Customer Product Line, no restrictions. Specific MCAL drivers List.	1 year support included, 20% of the license price year 2 and beyond.	Additional SoC at a discount.

AUTOSAR Software i.MX Family Product Selection Matrix Worksheet

MCAL - i.MX			Selection
Family		iMX 6/ iMX 8	
Processor/Derivative Part		Q/D/SoloX/DL/S/SL/UL/QM/QXP/DXP…	
Core		A72/A53/A9/M4	
Basic Modules	I/O Modules	Port	Yes
		Dio	Yes
		Icu	Yes
		Pwm	Yes
		Adc	Yes
	Comm Modules	Spi	Yes
		Can	Yes
		Eth	Yes
	Mcu Modules	Mcu	Yes
		Gpt	Yes
		Fls-Fee	Yes
		Int. Wdg	Yes
Complex Drivers	Per Customer Specific Peripheral		
Compilers	ARM		
	Diab		
	GCC		
	GHS		
Autosar Revisions	4.0.3		
	4.1.3		
	4.2.1		
	4.3		
Functional Saftey ISO 26262	ASIL A		
	ASIL B		

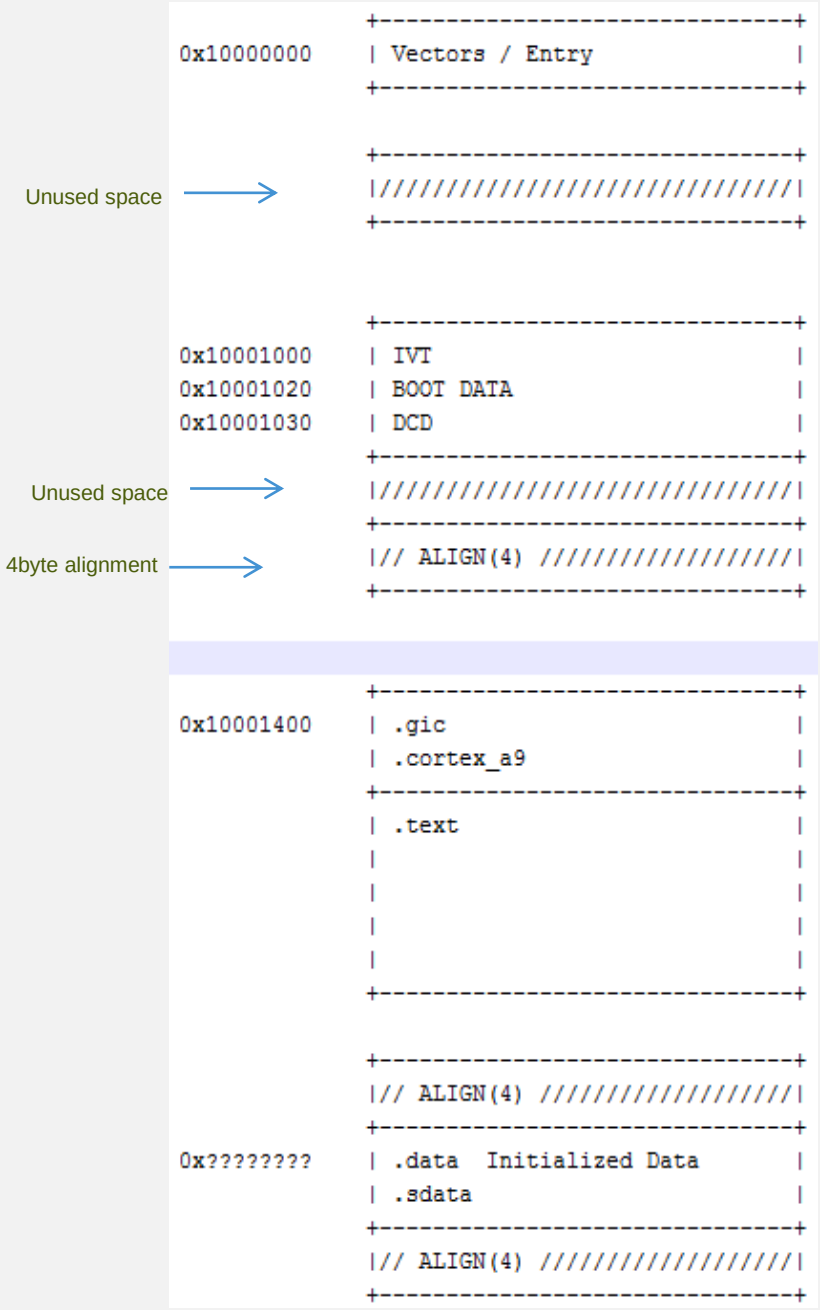
i.MX MCAL Startup Example

i.MX Typical Boot Flow



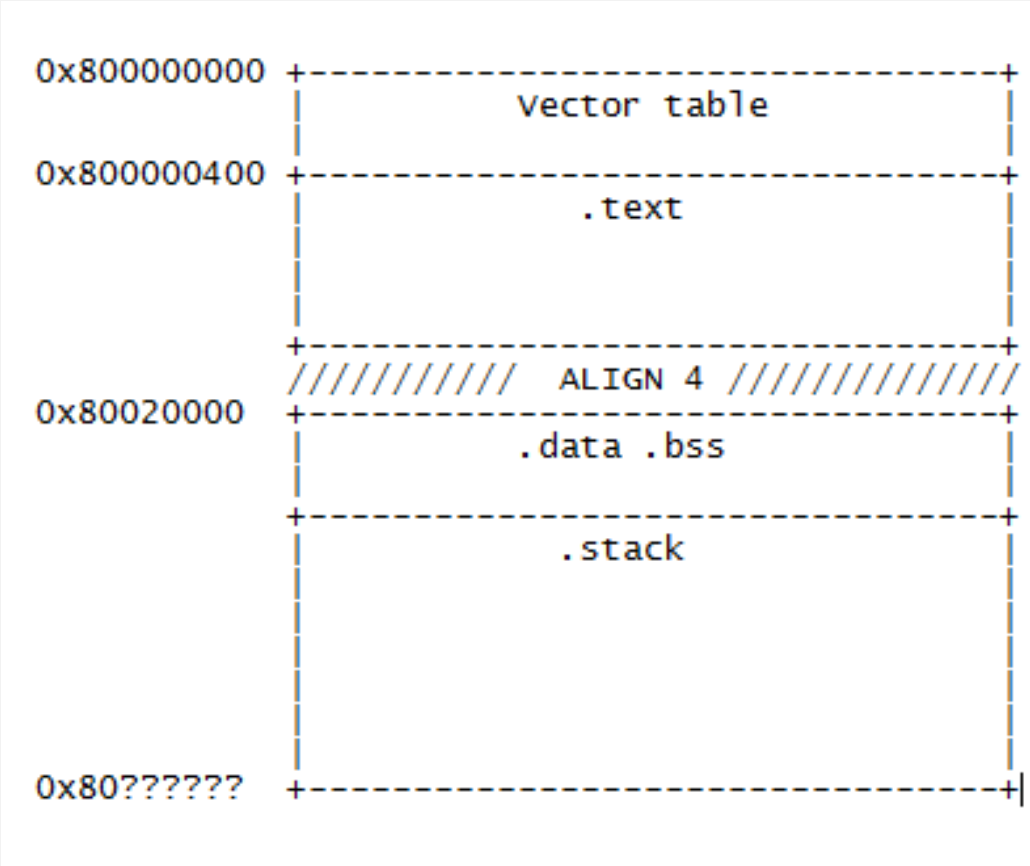
User Program Image (Linker Descriptor & Assembly files vs. Debugger script)

- Image vector table—A list of pointers located at a fixed address that the ROM examines to determine where other components of the program image are located
- Boot data—A table indicating the program image location, program image size in bytes, and the plugin flag
- Device configuration data—IC configuration data
- User code and data



Linker Descriptor for IMX6

- This Linker descriptor describes the alignments made in the memory layout
- The table and below description are for cortex A9 of IMX6
  - Entry begin execution at a 0x10000000
  - Exception Vector handlers are defined here whose definition are found in vectors.s
  - Startup.s is called to initialize stack, neon co-processor, .bss, configure on-chip peripherals to default values via DCD .
  - Enable interrupts via gic.s
  - CPU initialization via cortexA9.s
  - Application code sits in *.text section. Data sits in *.data section followed by stack and heap sections



Linker Descriptor for IMX8

- This Linker descriptor describes the alignments made in the memory layout
- The table and below description are for cortex M4 of IMX8
  - Entry begin execution at 0x80000000 (RAM region)
  - Exception Vector handlers are defined here whose definition are found in exceptions.c
  - startup.s is called to initialize stack, .bss segment
  - Application code sits in *.text section. Data sits in *.data section followed by .stack and heap sections

Load Sample Application Image to IMX8QXP Board

- There are several ways:
  - Using debugger (TRACE 32 Lauterbach)
  - Boot from SD Card
  - Boot from NOR Flash

Using Debugger (TRACE32 Lauterbach)

- In order to load image (e.g. MCAL application image) to hardware via Lauterbach, the following steps are needed:

  - The application binary is needed and loaded to hardware via iMX8xx.cmm script file. The PC register must point to the application

  - Open Trace32 Application “t32marm.exe” => File/Run script and select .cmm file to run or File/Edit script and select .cmm file to debug script

Boot from SD-Card

- Input: application image, scfw image, ahab-container.img
- Tool: imx-mkimage tool

Step 1 - Create boot image by following below steps

- Copy imx-mkimage tool and input files to Linux environment
- Use the command below to create the final image

/mkimage_imx8 -soc QX -rev B0 -append ./ahab-container.img -c -scfw ./scfw_tcm.bin -m4 ./sample_app_mcal.elf.bin 0 0x80000000 -o flash_app.bin

Step 2 – Load boot image to SD Card by using below command:

sudo dd if=flash_app.bin of=/dev/sdc bs=1k seek=32 conv=fsync

Step 3 – Insert SD Card, change the BOOT MODE to SD1 and reset the board

Boot from NOR Flash (1)

Input: application image, scfw image, ahab-container.img, QSPI configuration

Tool: UUU (Universal Update Utility)

Universal Updated Utility (UUU) tool is developed by NXP in order to load the boot image to NOR flash. User can download the tool at the link below (version 1.1.41)

https://github.com/NXPmicro/mfgtools/releases\

Step 1: Create boot image by using the below commands in Linux environment:

./mkimage_imx8 -soc QX -rev B0 -dev flexspi -append ahab-container.img -c –scfw scfw_tcm_mek.bin -m4 sample_app_mcal.elf.bin 0 0x80000000 –out flash_sa_qspi.bin

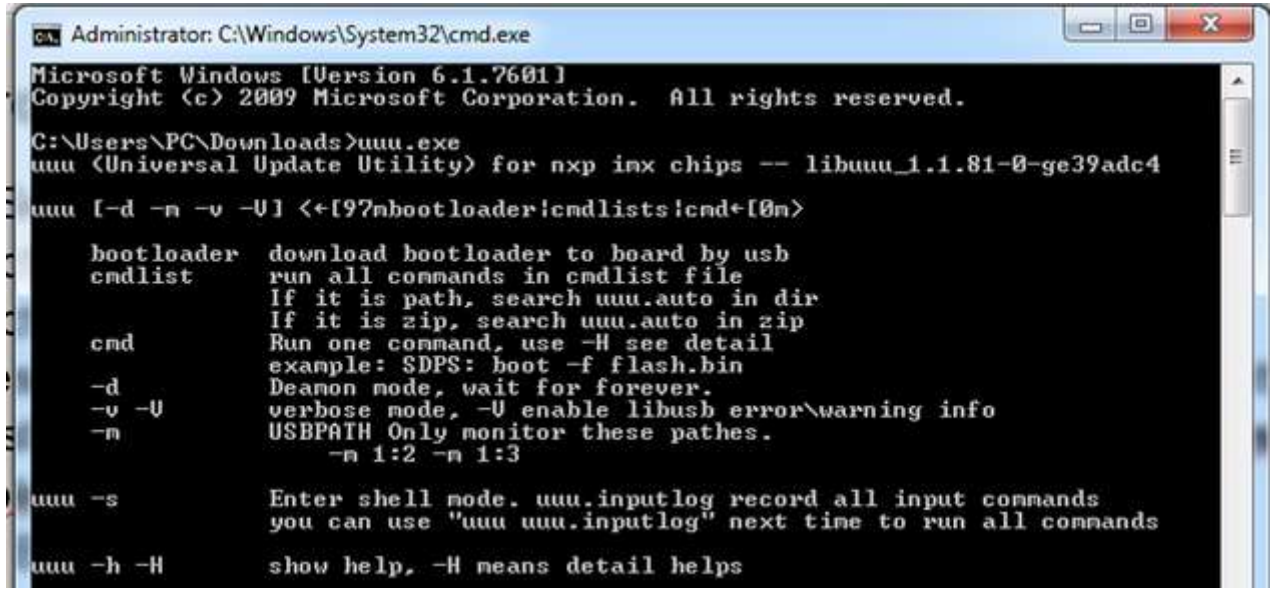
dd if=**qspi-bin_b0** of=./flash.bin bs=1k seek=1

dd if=./flash_sa_qspi.bin of=./flash.bin oflag=append bs=1k seek=4

Boot from NOR Flash (2)

Step 2: Setup hardware and run tool to load image to NOR flash

- Connect serial port (J11- MEK board) to PC
- Connect USB TYPE C (J10 – MEK board) to PC.
- Hold reset button on MEK board then in the PC DOS Command Line run the following command (ensure administrator permission to download flash.bin to QSPI NOR): **uuu -b qspi flash.bin**
- Switch BOOT MODE in MEK board to QSPI, then reset the board

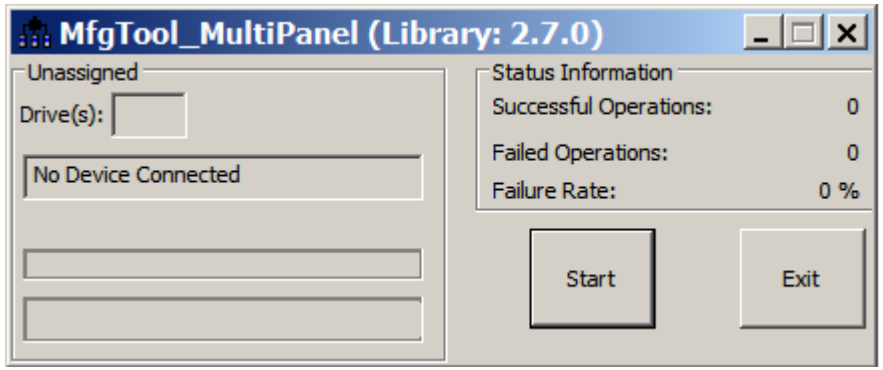


Manufacture Tool (MFG tool)

- MFG tool is developed to support customer in order to load the image to hardware. Steps below is to load the image to hardware
  - Create application image from application binary file, dependent input (e.g. scfw binary of iMX8)
  - Update mfgtools\Profiles\Linux\OS Firmware\ucl2.xml script by modifying name of application binary
  - Run script to start MFG tool.

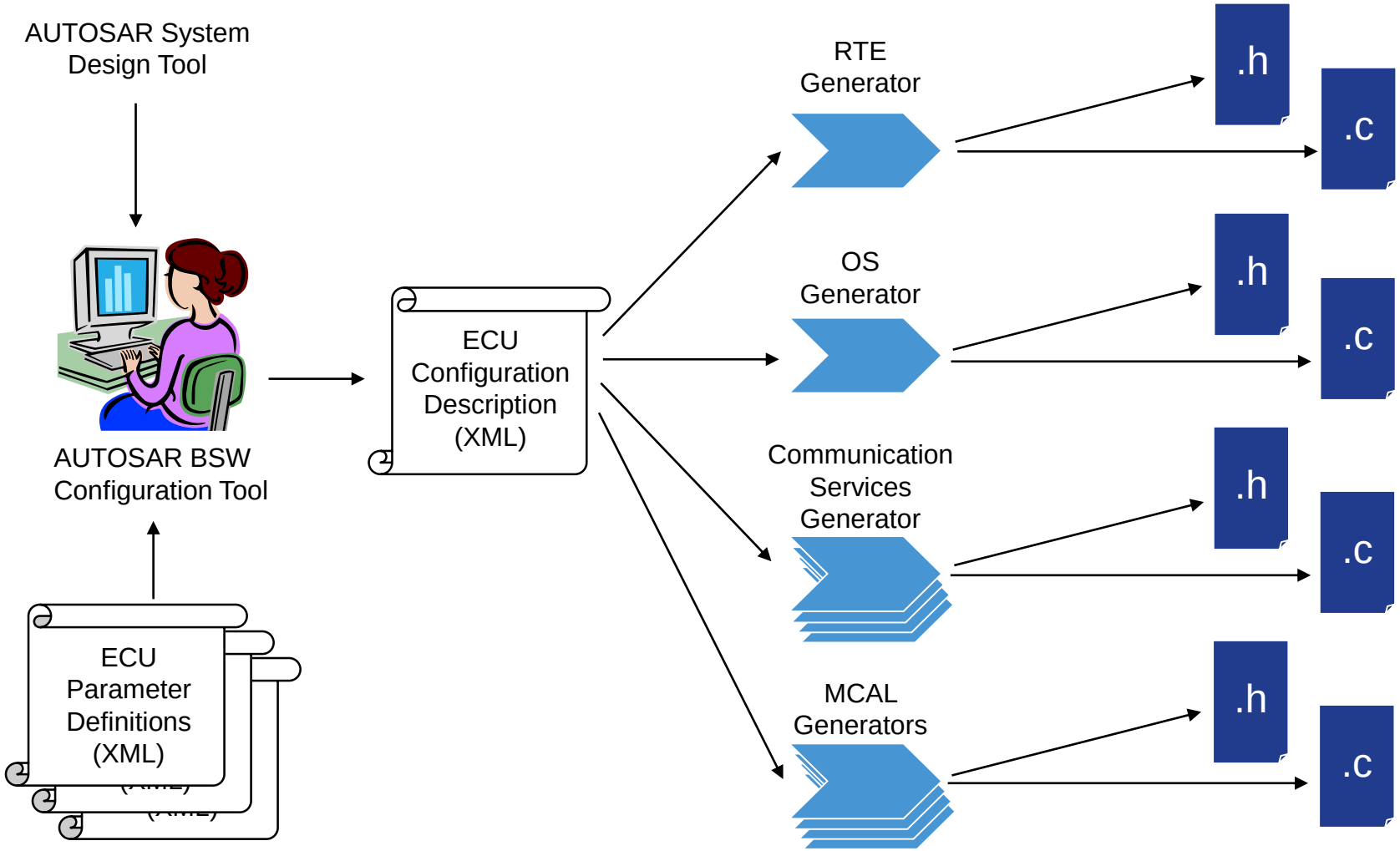
  For example:

  mfgtool2-yocto-mx8-arm2-lpddr4-sdcard-emmc1.vbs for eMMC memory mfgtool2-yocto-mx8-arm2-lpddr4-sdcard-sd2.vbs for Sdcard memory
  - Click Start button to load the image prepared to the hardware



MCAL BSW Configuration Tool

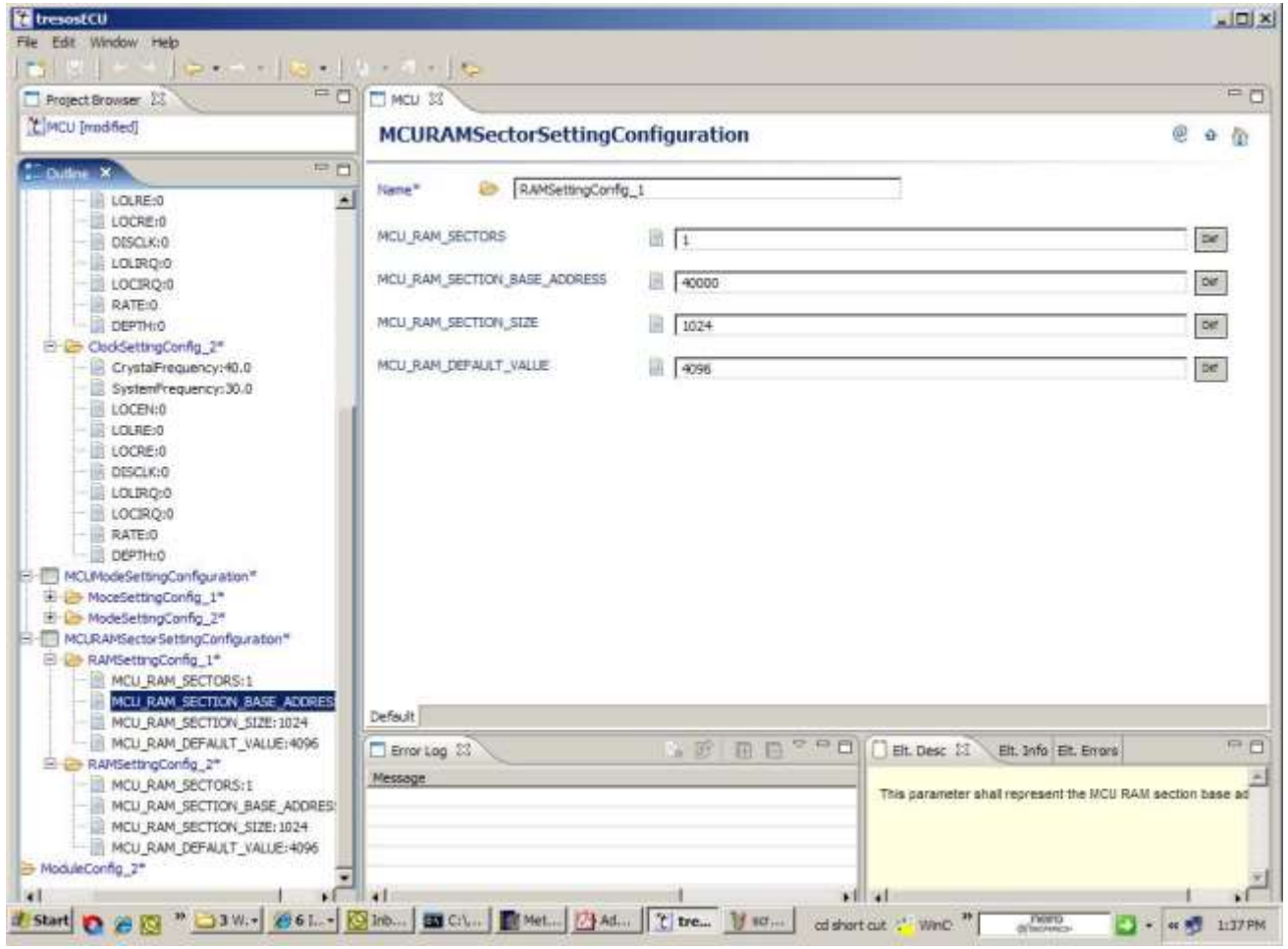
Basic Software Configuration Process



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

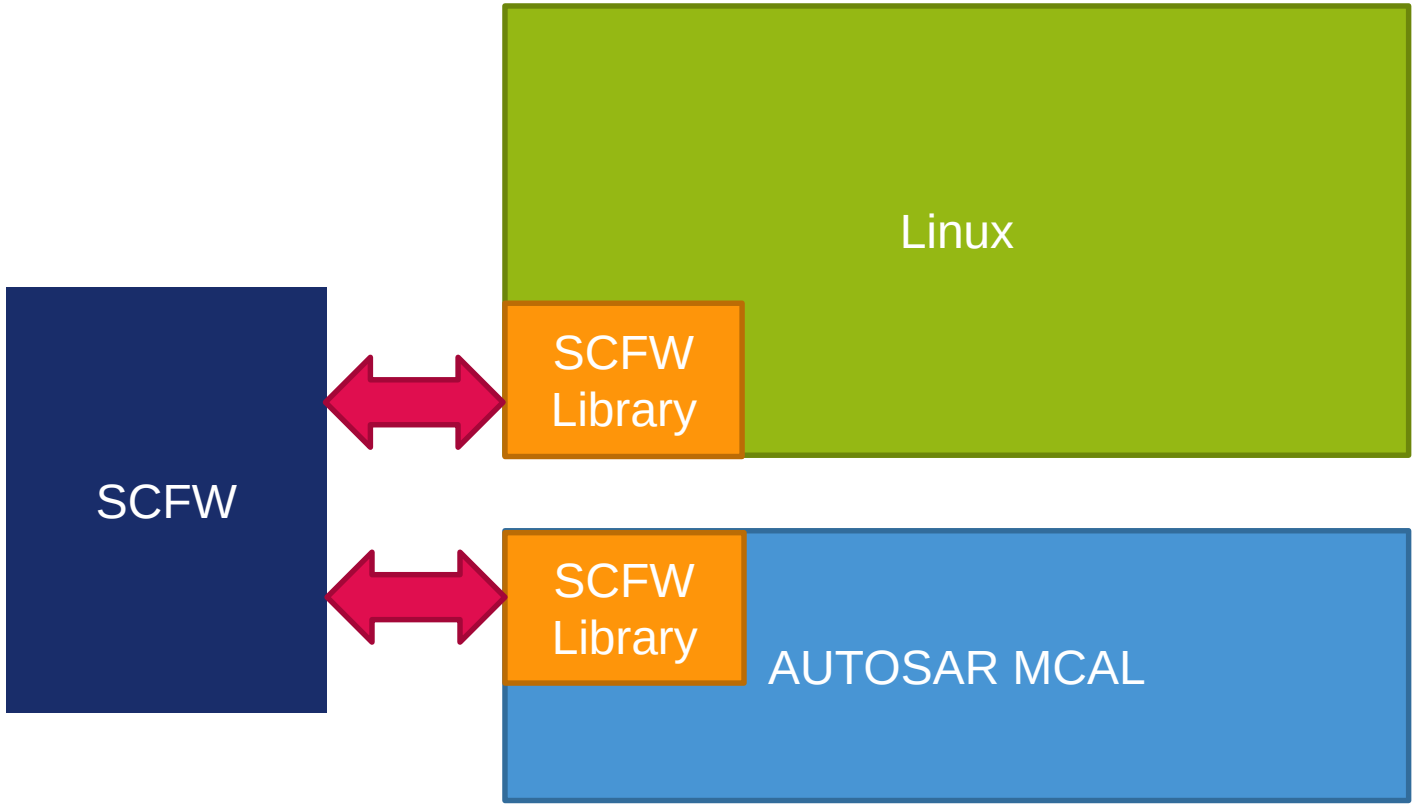


i.MX MCAL Microcontroller Drivers

Background - System Controller Firmware (SCFW)

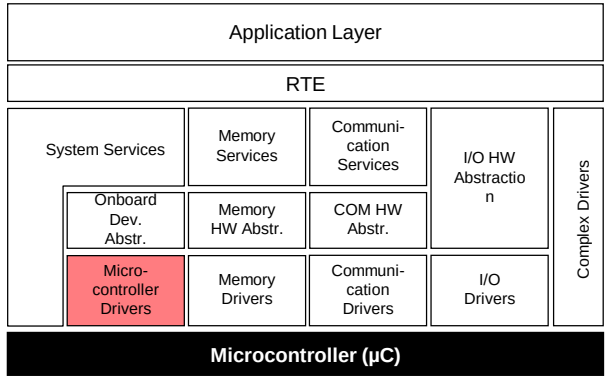
The System Controller Firmware (SCFW) runs on the System Controller Unit and is responsible for managing requests from other cores in the system.

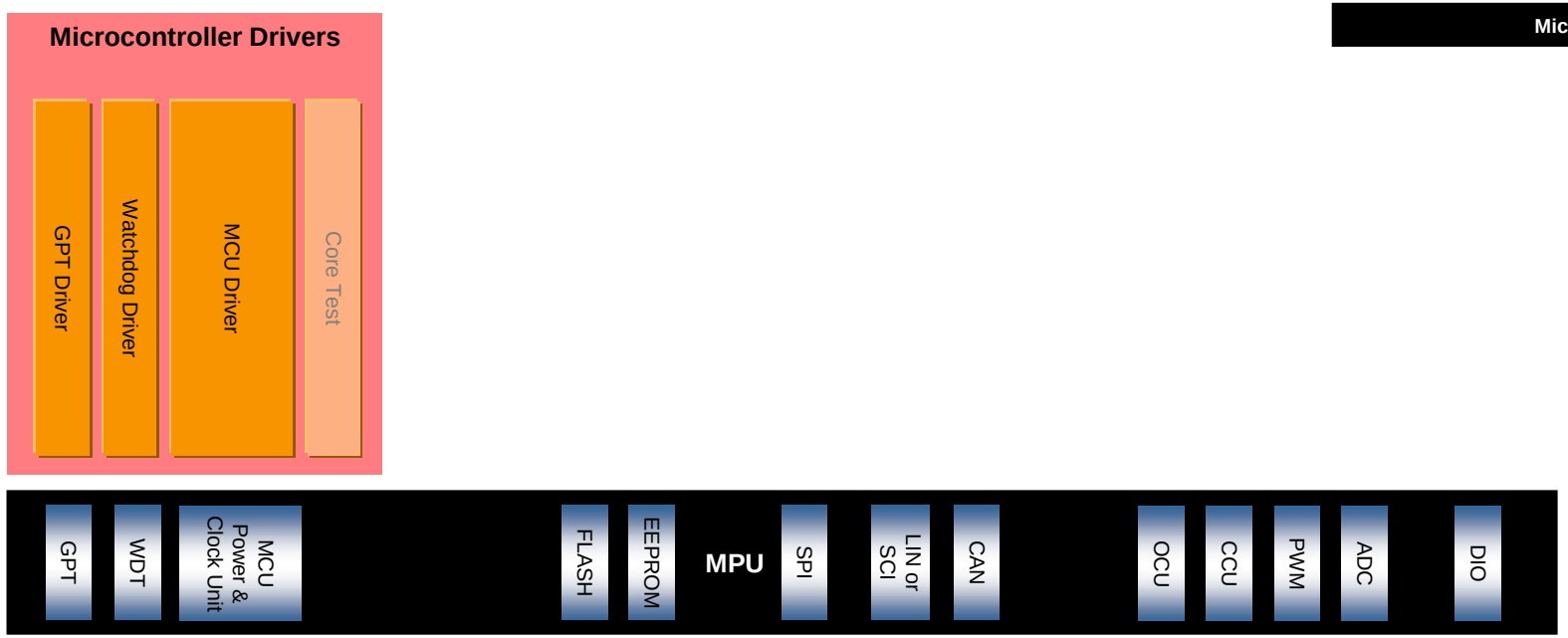
MCAL software components communicate with System Controller Unit (SCU) via an Application Programming Interface (API) library.



i.MX MCALs— Microcontroller Abstaction Layer

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



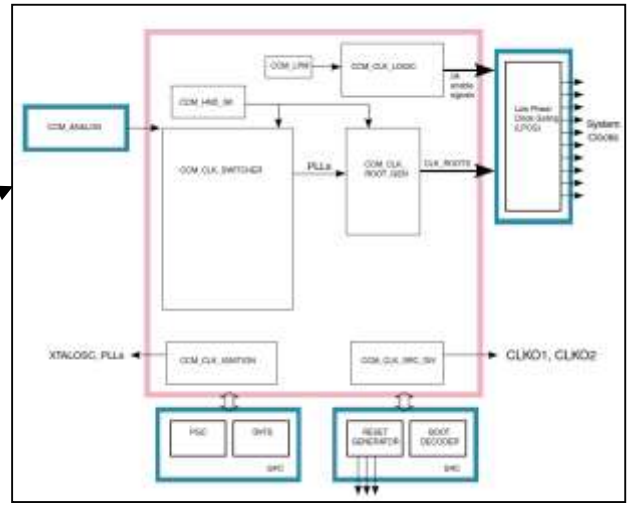


Source: AUTOSAR

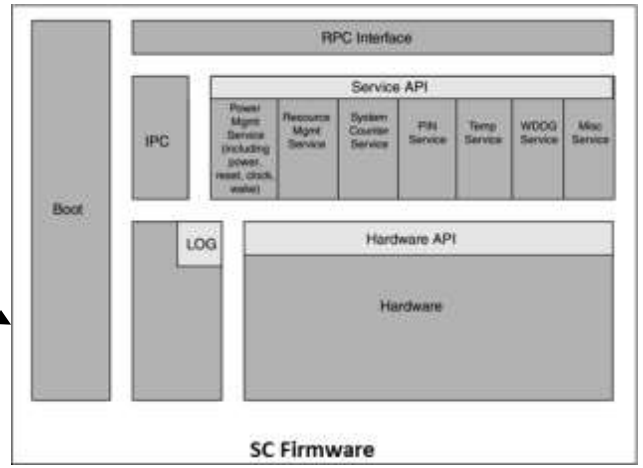
MCU Module – NXP Implementation

MCU implementation:
Mcu_Init (...)
Mcu_InitRamSection (...)
Mcu_InitClock (...)
Mcu_GetResetReason(...)
Mcu_DistributePllClock (...)
Mcu_GetPllStatus (...) (
Mcu_GetResetRawValue (...)
Mcu_PerformReset (...)
Mcu_SetMode (...)
Mcu_GetVersionInfo (...)
Mcu_GetRamState (...)

IMX6



IMX8



MCU Module – NXP Implementation

For IMX6:

MCU module shall initialize and manage its functionalities (clock, mode,…) as shown in the AUTOSAR specification through accessing register directly.

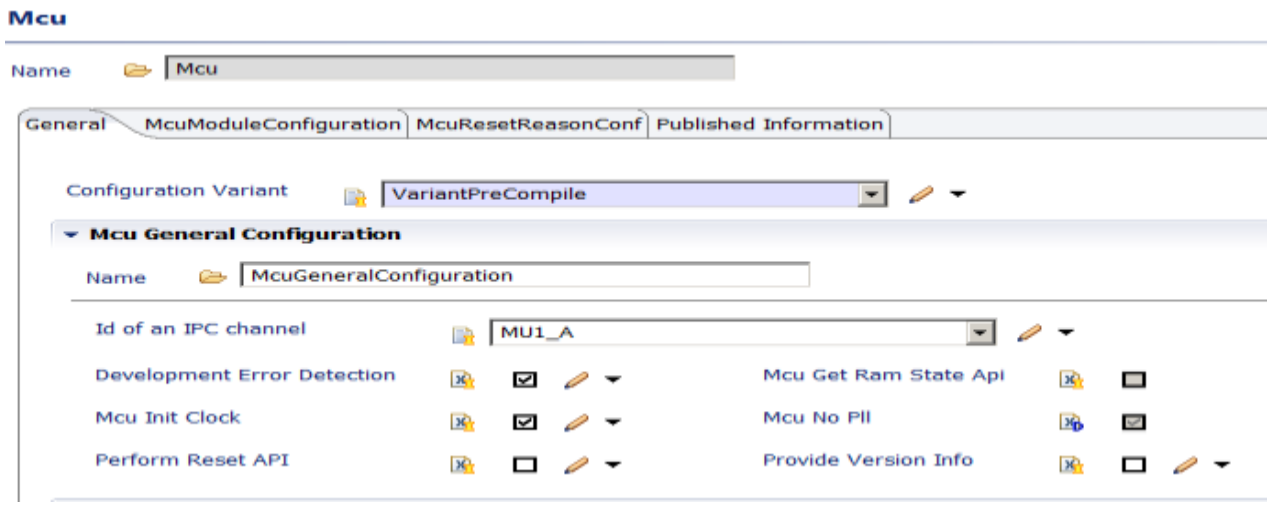
For IMX8:

MCU module shall initialize and manage its functionalities (clock, mode,…) as shown in the AUTOSAR specification through system controller firmware (SCFW) binary that is loaded to SCU

MCU driver for IMX8 does not support APIs “Mcu_DistributePllClock, Mcu_GetResetRawValue, Mcu_GetRamState are not available.”

MCU Parameters Configuration

Enable/disable the API's required based on requirement.



These parameters are almost the same between IMX6 and IMX8 except for "ID of an IPC channel" parameter.

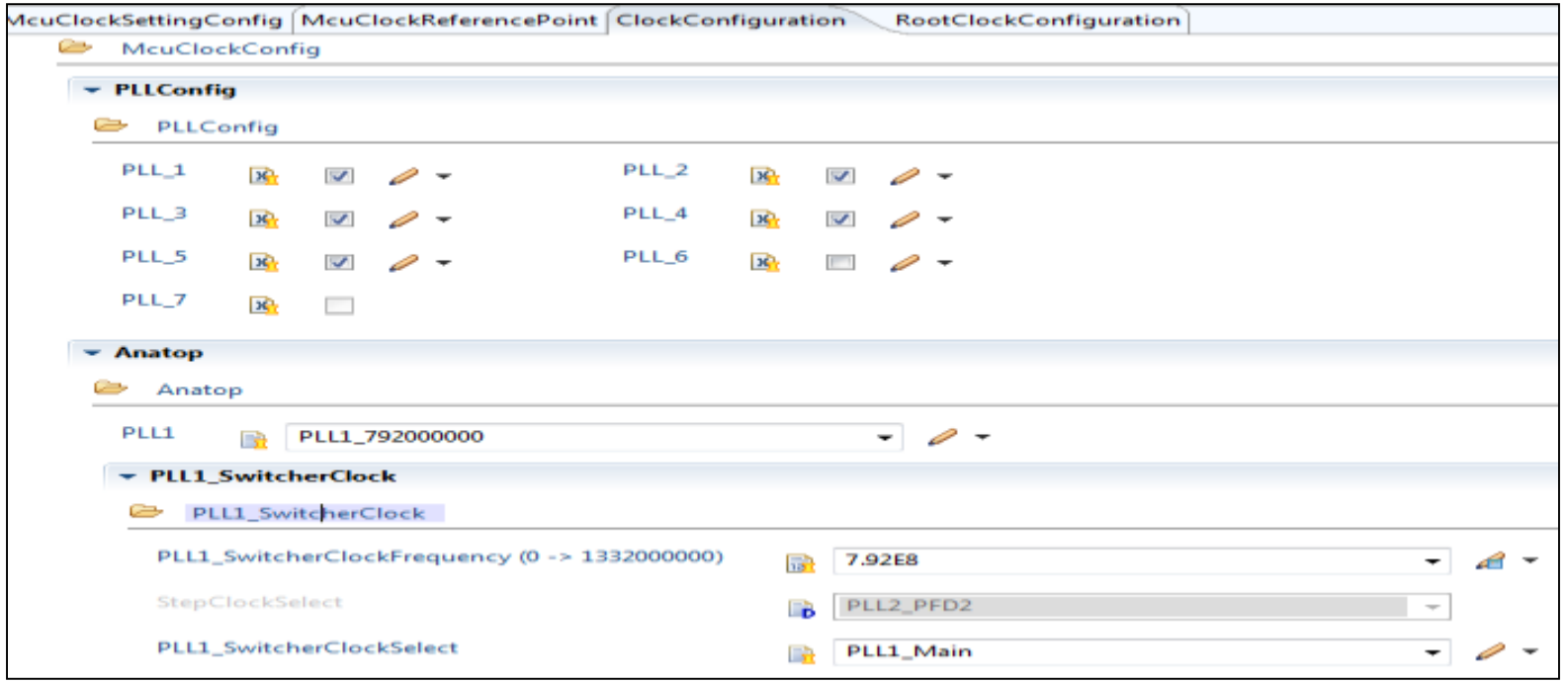
The parameter "ID of an IPC channel" is a specific parameter that is supported by IMX8 only. This parameter is used to select what message unit (MU0_B_BASE, MU1_A_BASE, MU0_A0_BASE, MU0_A1_BASE, MU0_A2_BASE or MU0_A3_BASE) will be used to communicate between MCU driver and SCFW.

MCU Clock Settings Configuration

For IMX6: clock settings for MCU via CCM hardware

Select what PLL/Peripherals, what divide for PLL/Peripherals and what PLL/Peripherals clock value will be configured. Based on these configuration, MCU clocks for PLL and peripherals will be initialized via write the corresponding values to registers via CCM hardware

These parameters are used to initialize registers of peripherals for IMX6 hardware

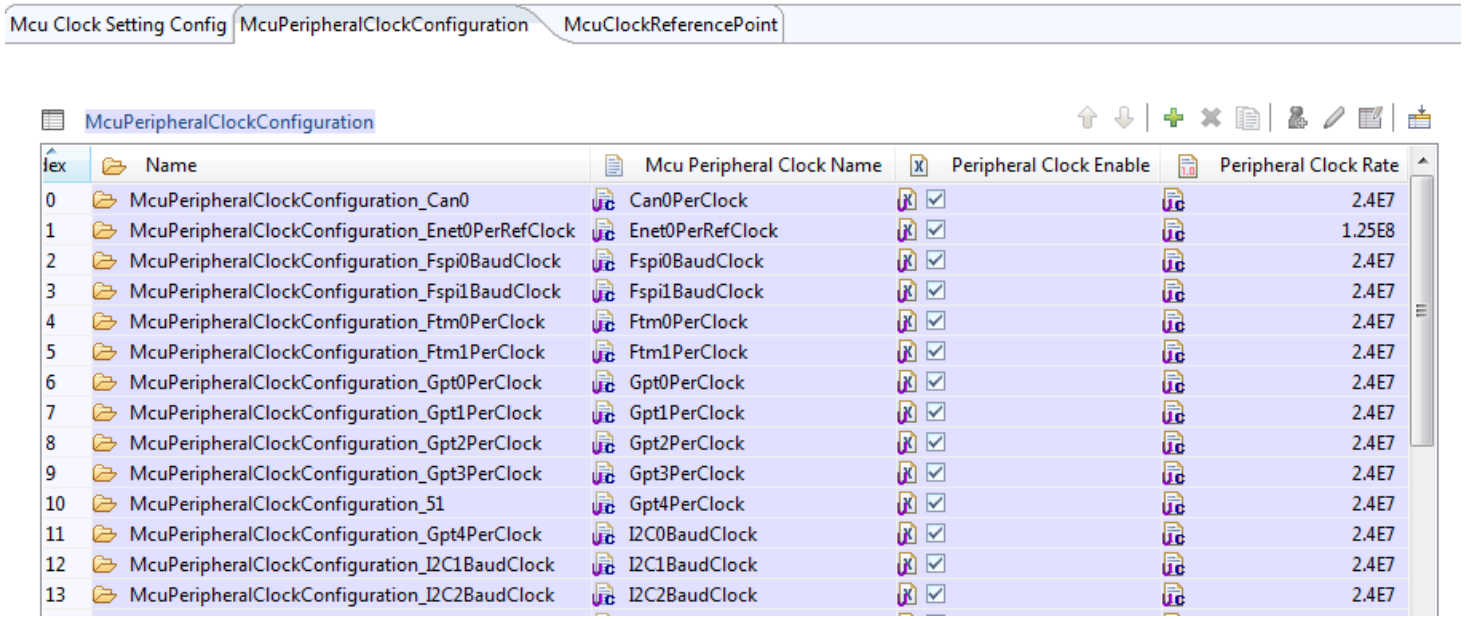


MCU Clock Settings Configuration

For IMX8: clock settings for MCU via SCFW

Users can create a list of peripherals that they want to configure clock value.
peripheral is selected from a list of MCU Peripheral Clock Name.
The list is peripherals that support clock setting is referenced from SCFW firmware User Manual document.

These parameters are used as inputs for APIs of SCFW



lex	Name	Mcu Peripheral Clock Name	Peripheral Clock Enable	Peripheral Clock Rate
0	McuPeripheralClockConfiguration_Can0	Can0PerClock	☑	2.4E7
1	McuPeripheralClockConfiguration_Enet0PerRefClock	Enet0PerRefClock	☑	1.25E8
2	McuPeripheralClockConfiguration_Fspi0BaudClock	Fspi0BaudClock	☑	2.4E7
3	McuPeripheralClockConfiguration_Fspi1BaudClock	Fspi1BaudClock	☑	2.4E7
4	McuPeripheralClockConfiguration_Ftm0PerClock	Ftm0PerClock	☑	2.4E7
5	McuPeripheralClockConfiguration_Ftm1PerClock	Ftm1PerClock	☑	2.4E7
6	McuPeripheralClockConfiguration_Gpt0PerClock	Gpt0PerClock	☑	2.4E7
7	McuPeripheralClockConfiguration_Gpt1PerClock	Gpt1PerClock	☑	2.4E7
8	McuPeripheralClockConfiguration_Gpt2PerClock	Gpt2PerClock	☑	2.4E7
9	McuPeripheralClockConfiguration_Gpt3PerClock	Gpt3PerClock	☑	2.4E7
10	McuPeripheralClockConfiguration_51	Gpt4PerClock	☑	2.4E7
11	McuPeripheralClockConfiguration_Gpt4PerClock	I2C0BaudClock	☑	2.4E7
12	McuPeripheralClockConfiguration_I2C1BaudClock	I2C1BaudClock	☑	2.4E7
13	McuPeripheralClockConfiguration_I2C2BaudClock	I2C2BaudClock	☑	2.4E7

MCU Power Mode Settings Configuration

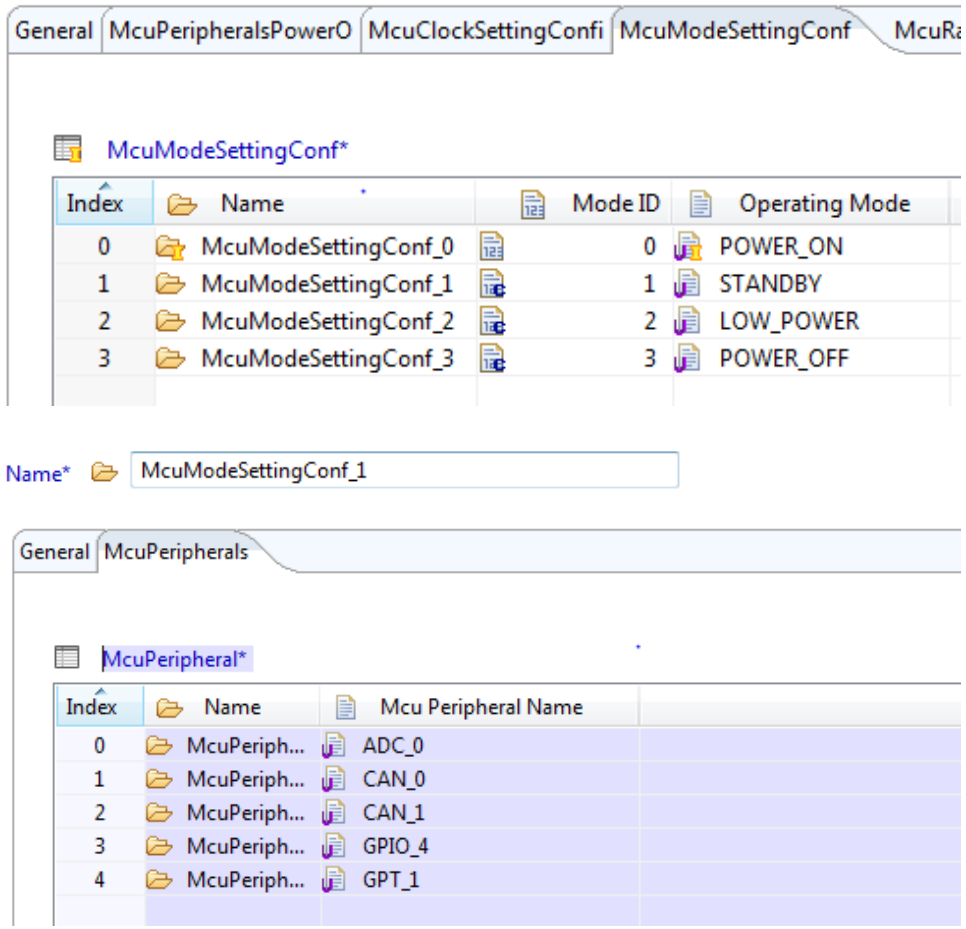
For IMX8: power mode settings for MCU via SCFW

There are four power modes
that user can configure as SCFW supports:

POWER_ON
STANDBY
LOW_POWER
POWER_OFF

Each **McuModeSettingConf** container has a list of peripherals/resources will be changed according to the power mode configured in the container

The list is peripherals that support power mode setting is referenced from SCFW firmware User Manual document.



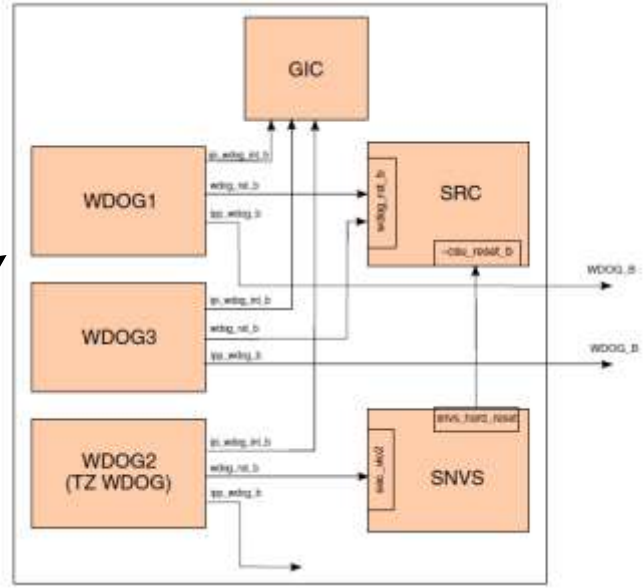
MCU Module Tips and Tricks

Manages frequency scaling procedure for peripheral root clock by programmable divider. The division is done on the fly without loss of clocks.

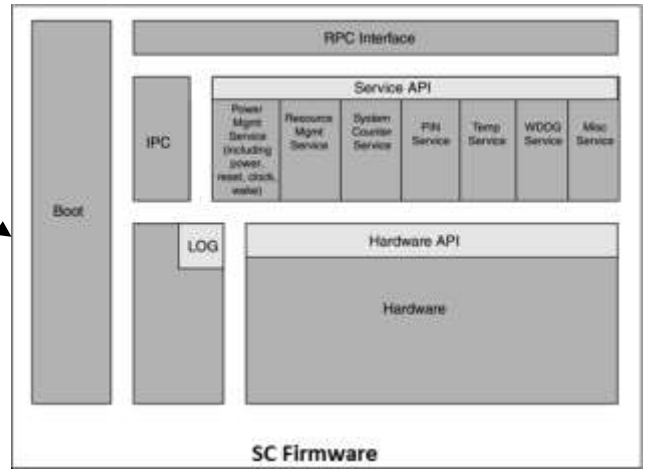
Watchdog Module – NXP Implementation

Watchdog implementation:
Wdg_Init (...)
Wdg_SetMode (...)
Wdg_SetTriggerCondition(...)
Wdg_GetVersionInfo (...)

IMX6



IMX8



Watchdog Module – NXP Implementation

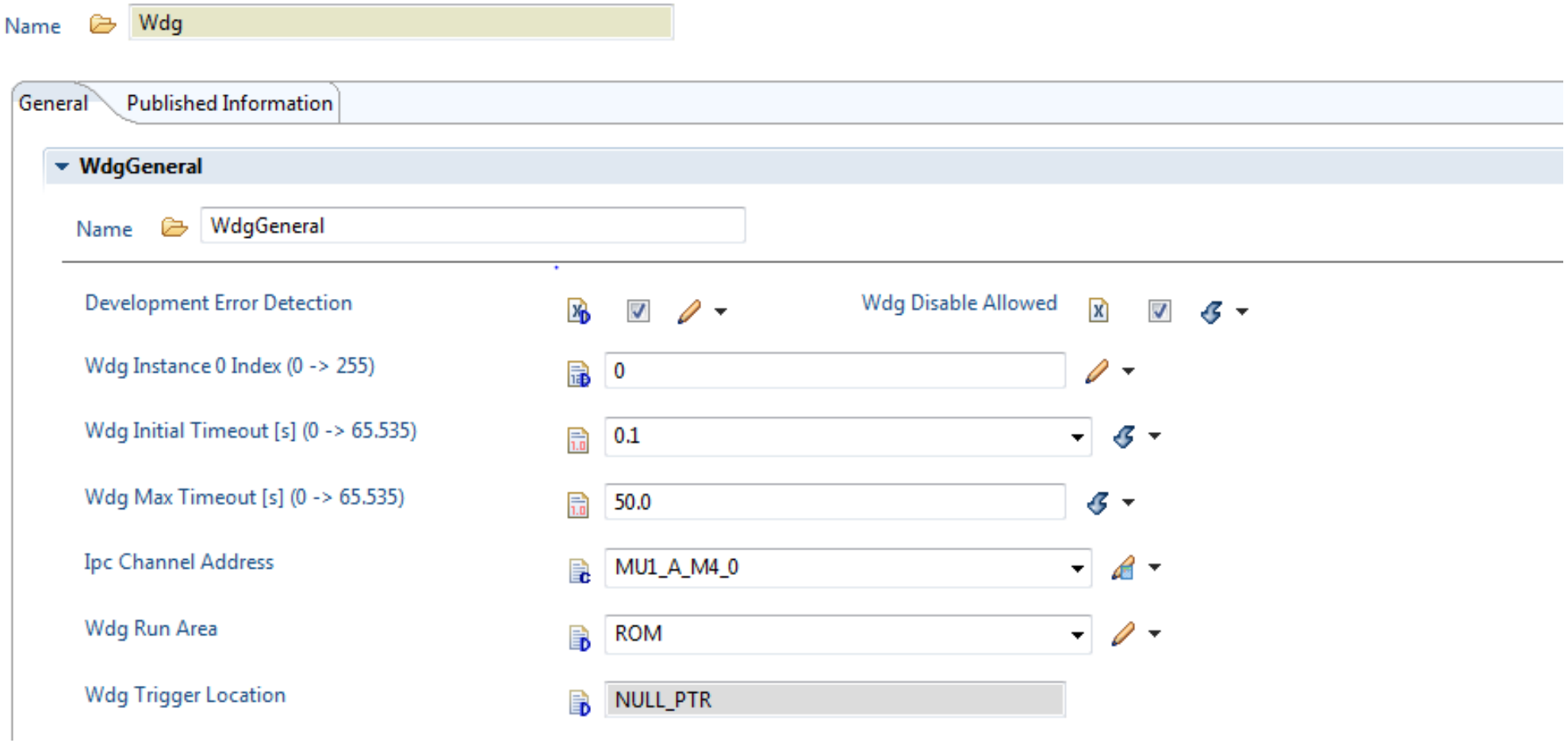
For IMX6:

WDG module shall initialize and manage its functionalities as shown in the AUTOSAR specification through accessing register directly.

For IMX8:

WDG module shall initialize and manage its functionalities as shown in the AUTOSAR specification through system controller firmware (SCFW) binary that is loaded to SCU

WDG Configuration in IMX8



Watchdog Module Tips and Tricks

- For IMX6
  - Watchdog will generate two signals
    - Wdg_B - This signal will power down the chip. “Shutdown”
    - Wdog_RESET_B -This signal is a reset source for the chip. “Restart”

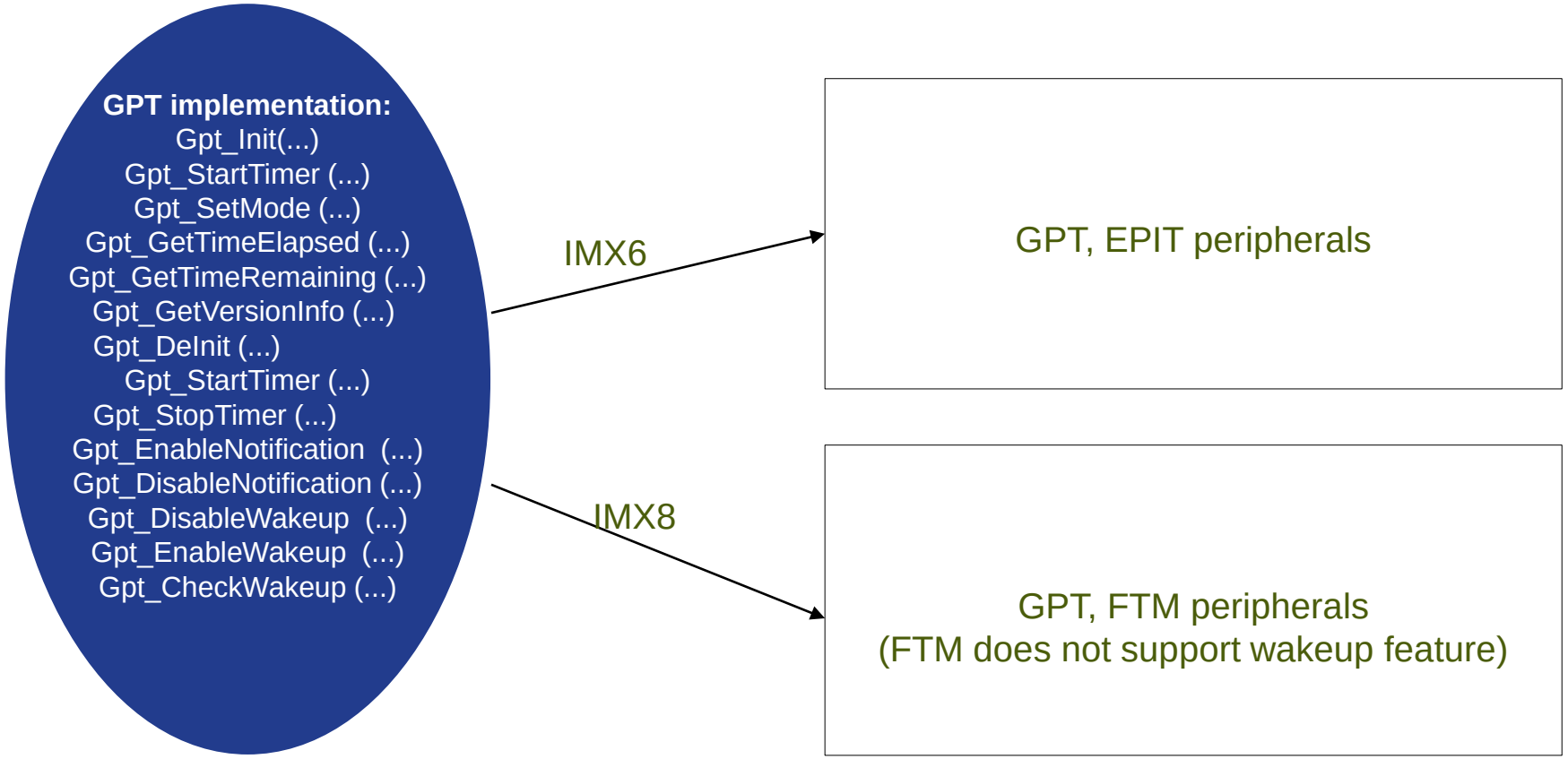
- WDOG1, WDOG2, and WDOG3(Support External Mode) are available on IMX6.

- External Mode is customer HW dependent and hence may require to be handled as complex driver

- For IMX8: WDG is controlled through SCFW

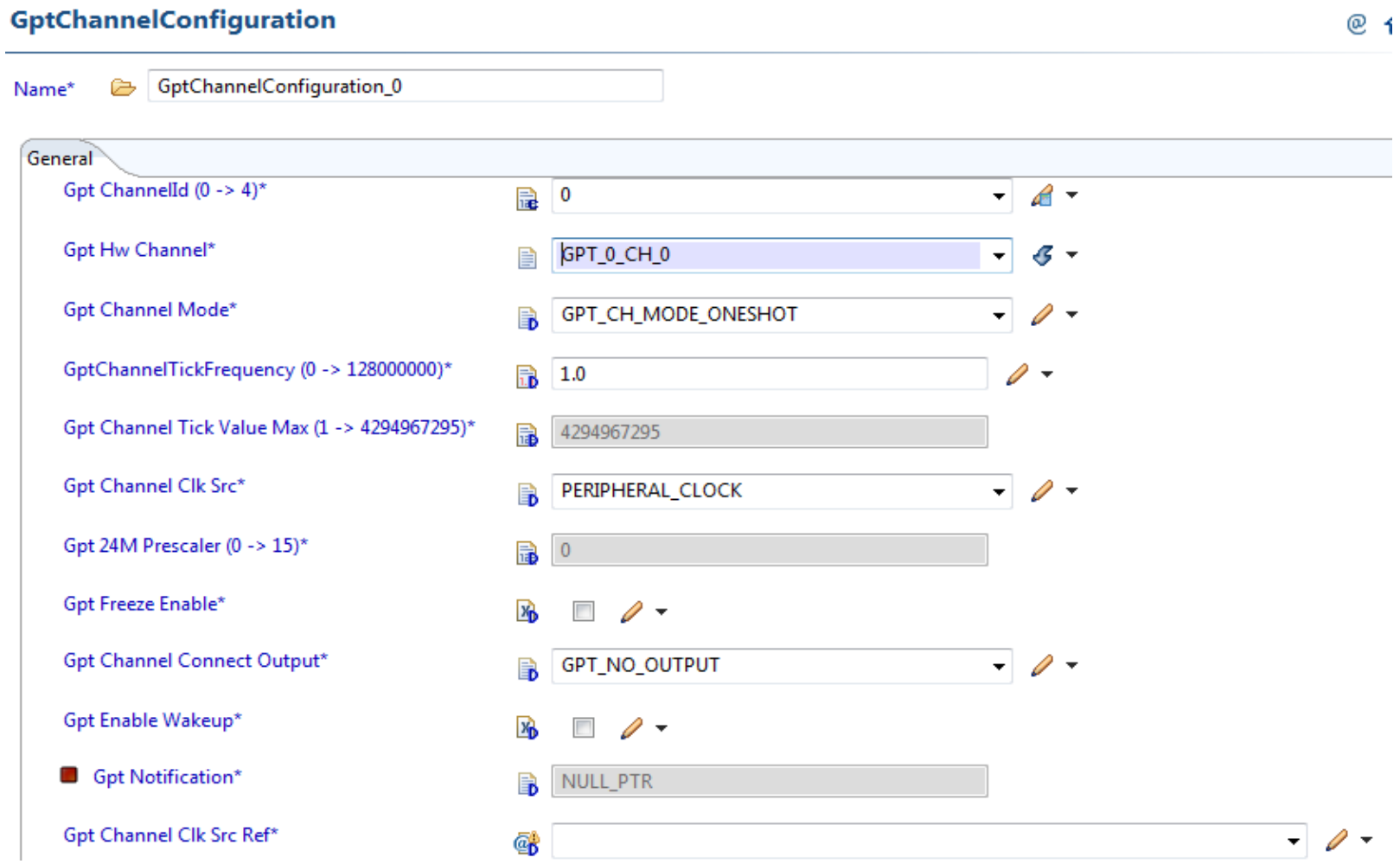
.

GPT Module – NXP Implementation



GPT Channel Configuration

Parameters are the same between IMX6 and IMX8



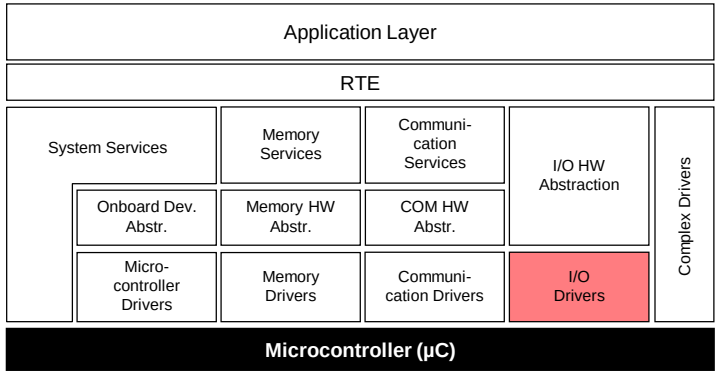
GPT Channel Tips and Tricks

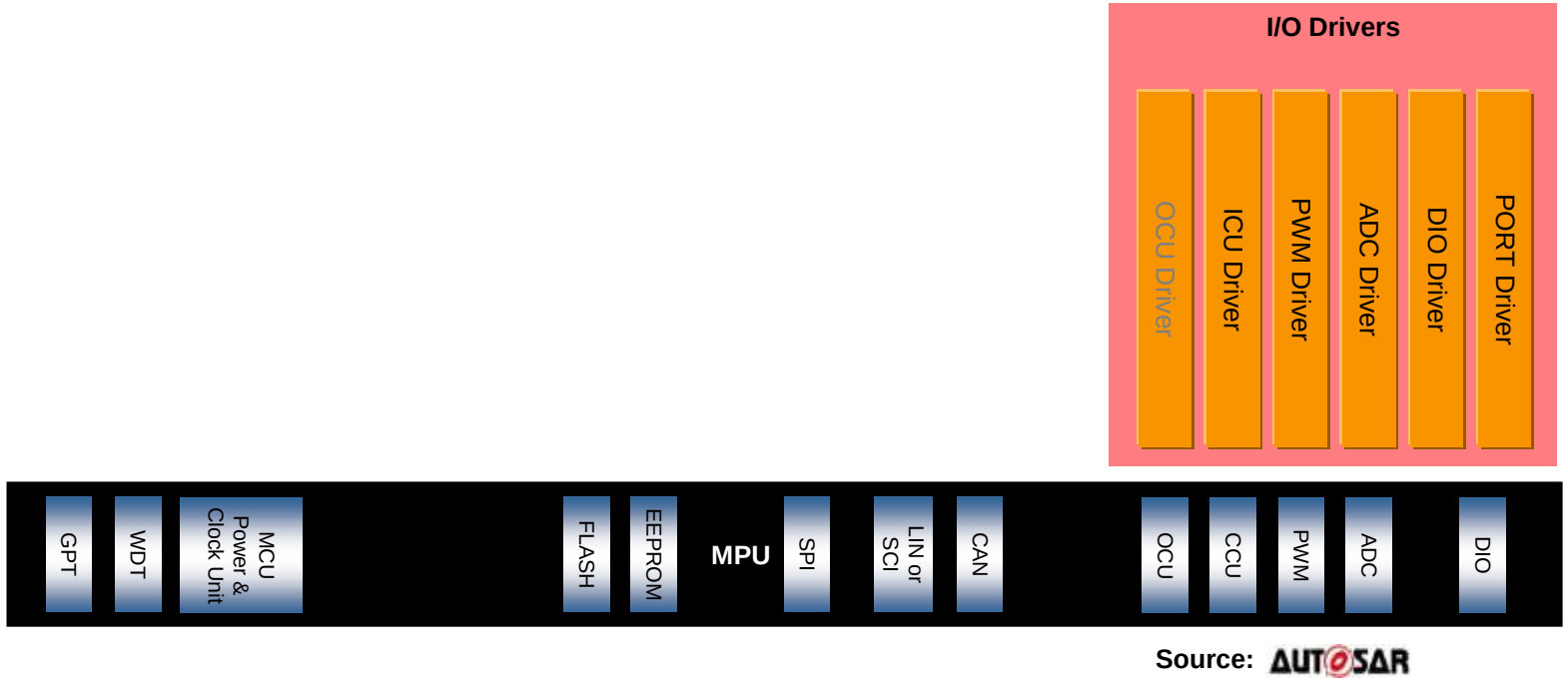
- Hardware requirement only for up counters
- Peripherals (GPT, EPIT, FTM) are used for GPT driver
  - IMX6 has GPT and EPIT peripherals
  - IMX8 has GPT and FTM peripherals

i.MX MCAL I/O Drivers

AUTOSAR – Microcontroller Abstaction Layer

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



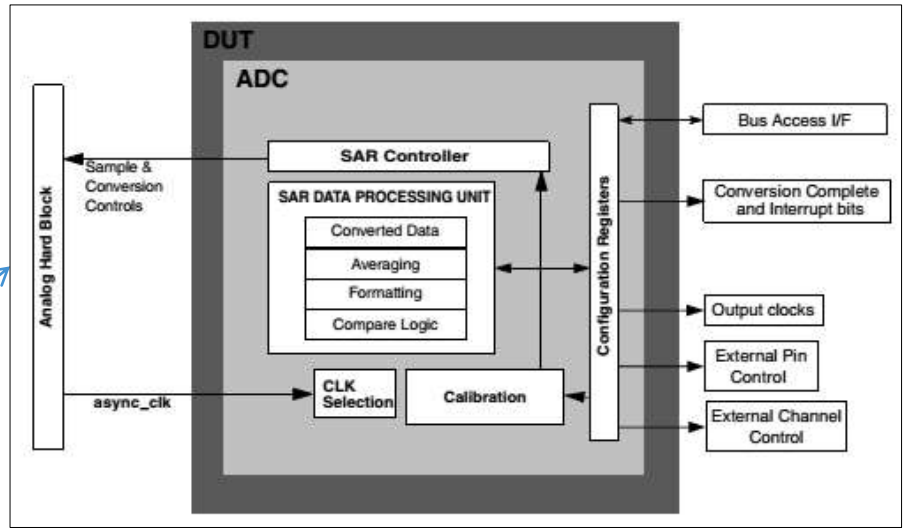


Source: AUTOSAR

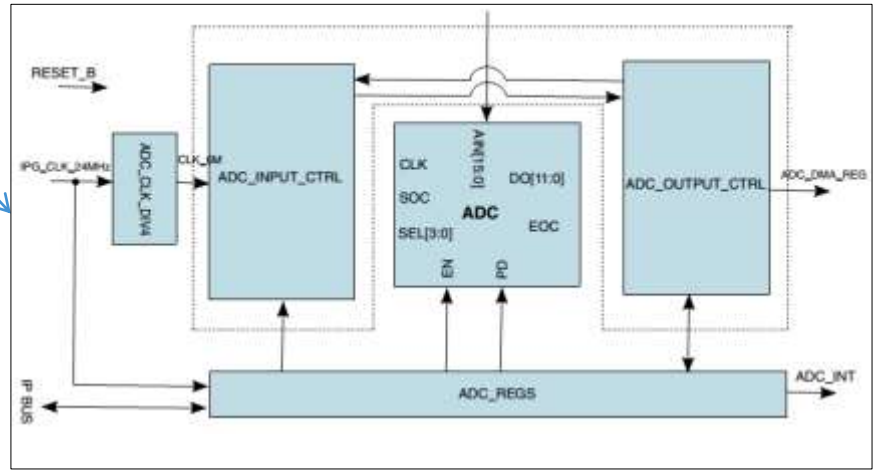
ADC Modules – NXP Implementation

Adc Access:
Adc_Init(...)
Adc_DeInit(...)
Adc_StartGroupConversion(...)
Adc_StopGroupConversion(...)
Adc_ReadGroup (...)
Adc_DisableHardwareTrigger (...)
Adc_EnableHardwareTrigger (...)
Adc_EnableGroupNotification(...)
Adc_DisableGroupNotification(...)
Adc_StopGroupConversion(...)
Adc_GetGroupStatus(...)
Adc_GetStreamLastPointer (...)
Adc_GetStreamLastPointer (...)
Adc_GetVersionInfo(...)
Adc_SetPowerState (...)
Adc_GetCurrentPowerState (...)
Adc_GetTargetPowerState (...)
Adc_PreparePowerState (...)

IMX6soloX



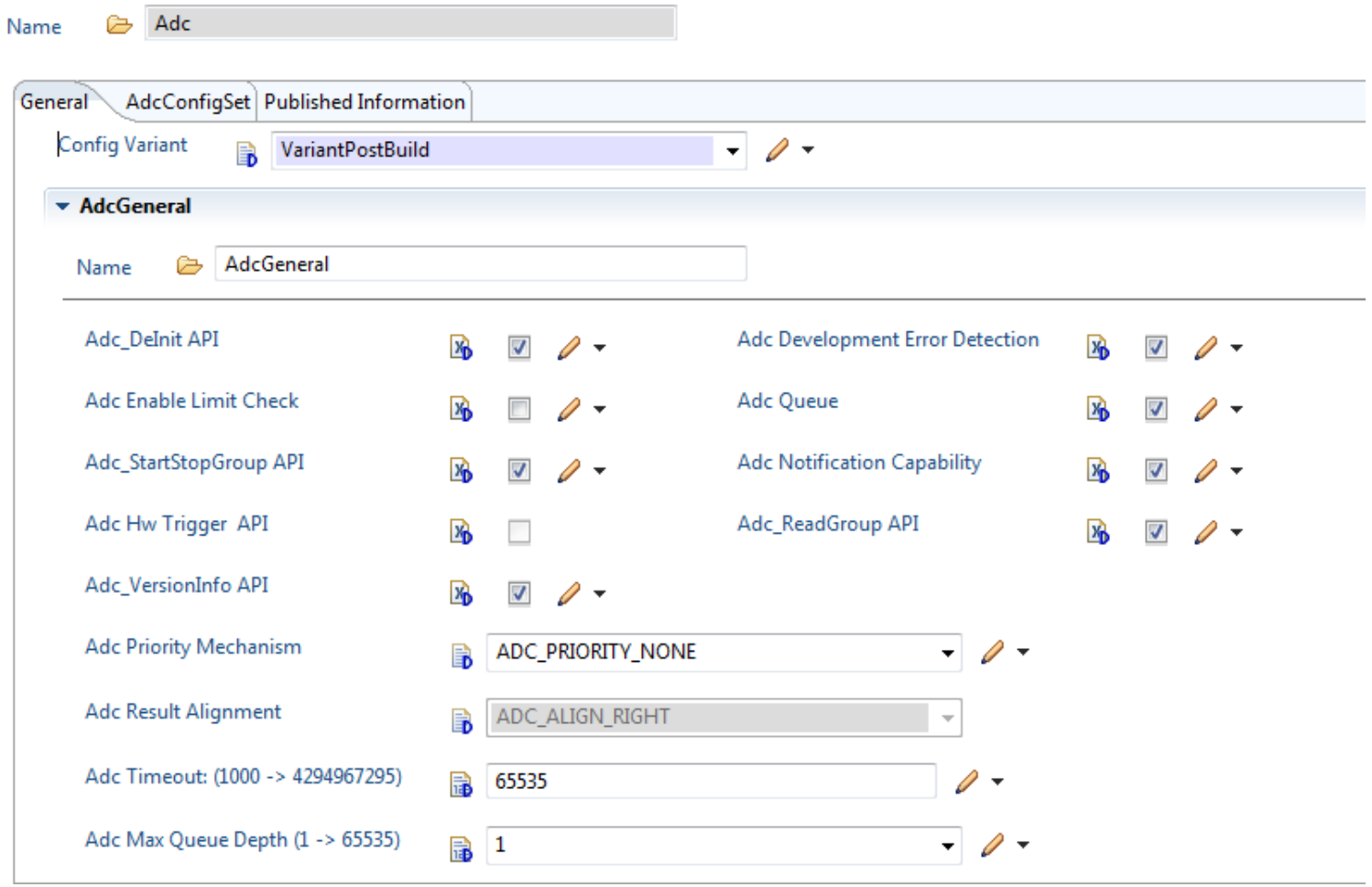
IMX8



APIs in white are for ASR4.0, ASR4.2 and ASR4.3
APIs in yellow are new APIs in ASR4.2 and ASR4.3

ADC Parameters Configuration

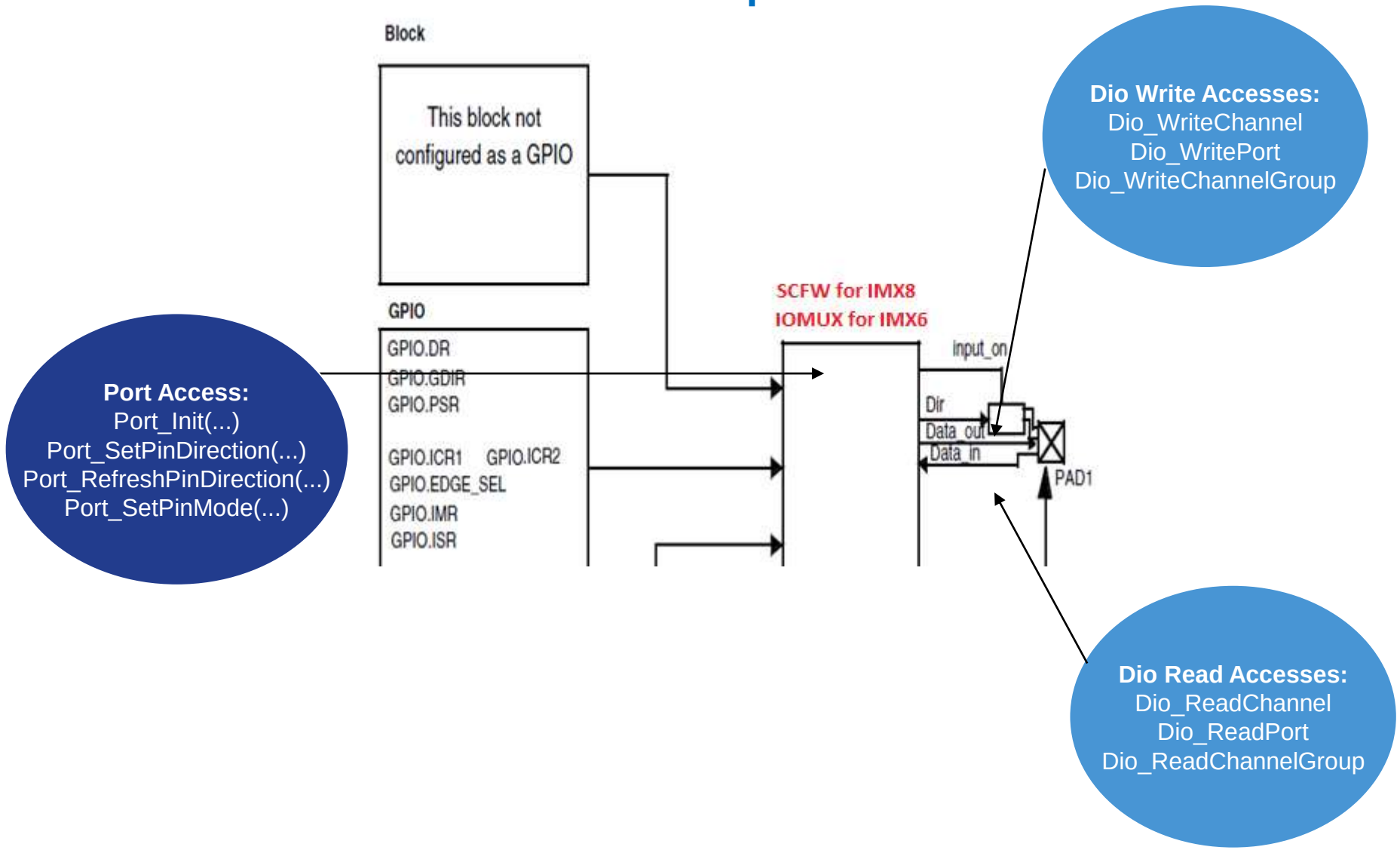
Enable/disable the API's required based on requirement



ADC Module Tips and Tricks

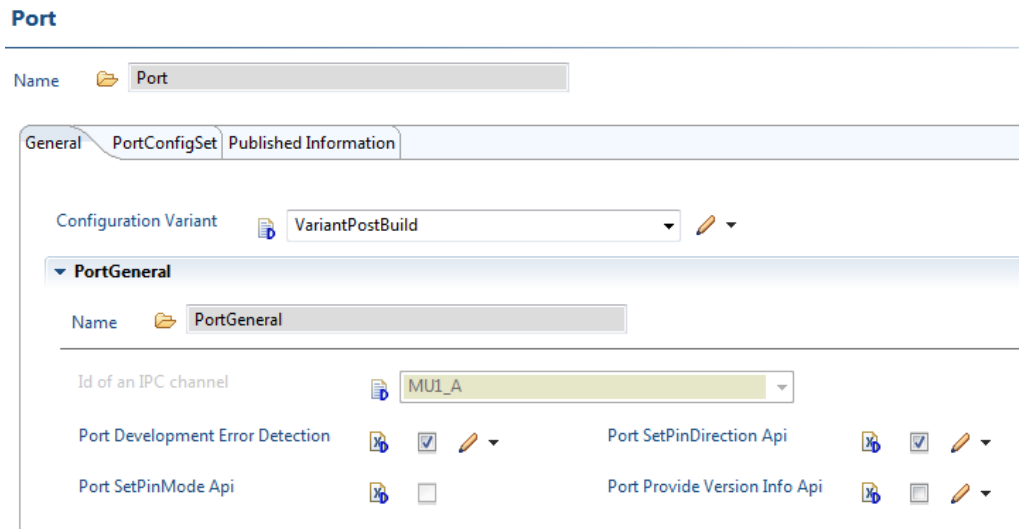
ADC driver for IMX6 and IMX8 does not support APIs relating hardware trigger: Adc_DisableHardwareTrigger and Adc_EnableHardwareTrigger

PORT/DIO Modules – NXP Implementation



PORT Parameters Configuration

Enable/disable the API's required based on requirement.



These parameters are almost the same between IMX6 and IMX8 except for "ID of an IPC channel" parameter.

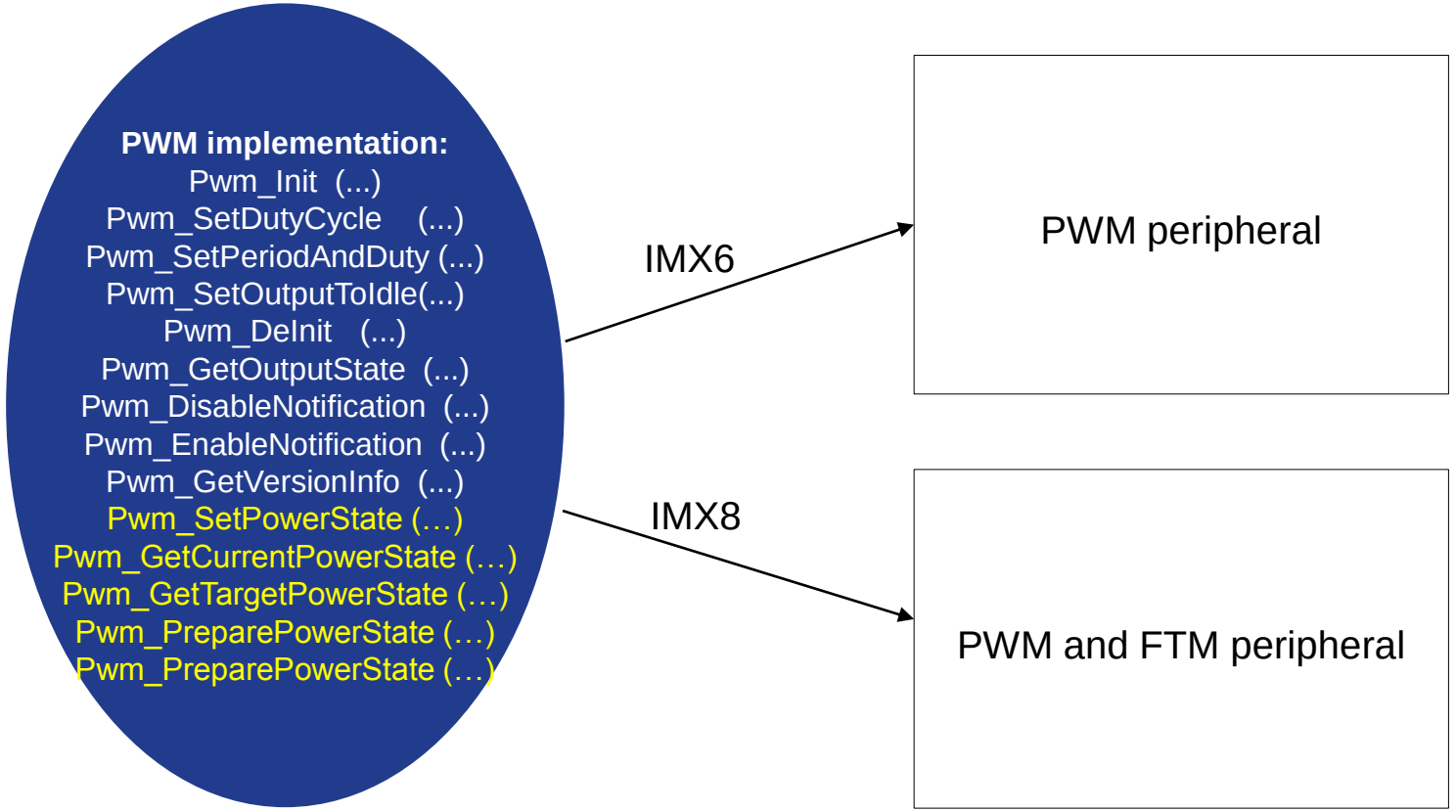
The parameter "ID of an IPC channel" is a specific parameter that is supported by IMX8 only. This parameter is used to select what message unit (MU0_B_BASE, MU1_A_BASE, MU0_A0_BASE, MU0_A1_BASE, MU0_A2_BASE or MU0_A3_BASE) will be used to communicate between MCU driver and SCFW.

This parameter shall be editable if MCU is not configured.

PORT Module Tips and Tricks

- User Application Requirements for PORT
  - User has to configure all available pins. All unused pins are set to its default values
  - Port init function shall be called after a reset, in order to reconfigure the ports and port pins of the MCU. This Port Init function should be called first in during initializations

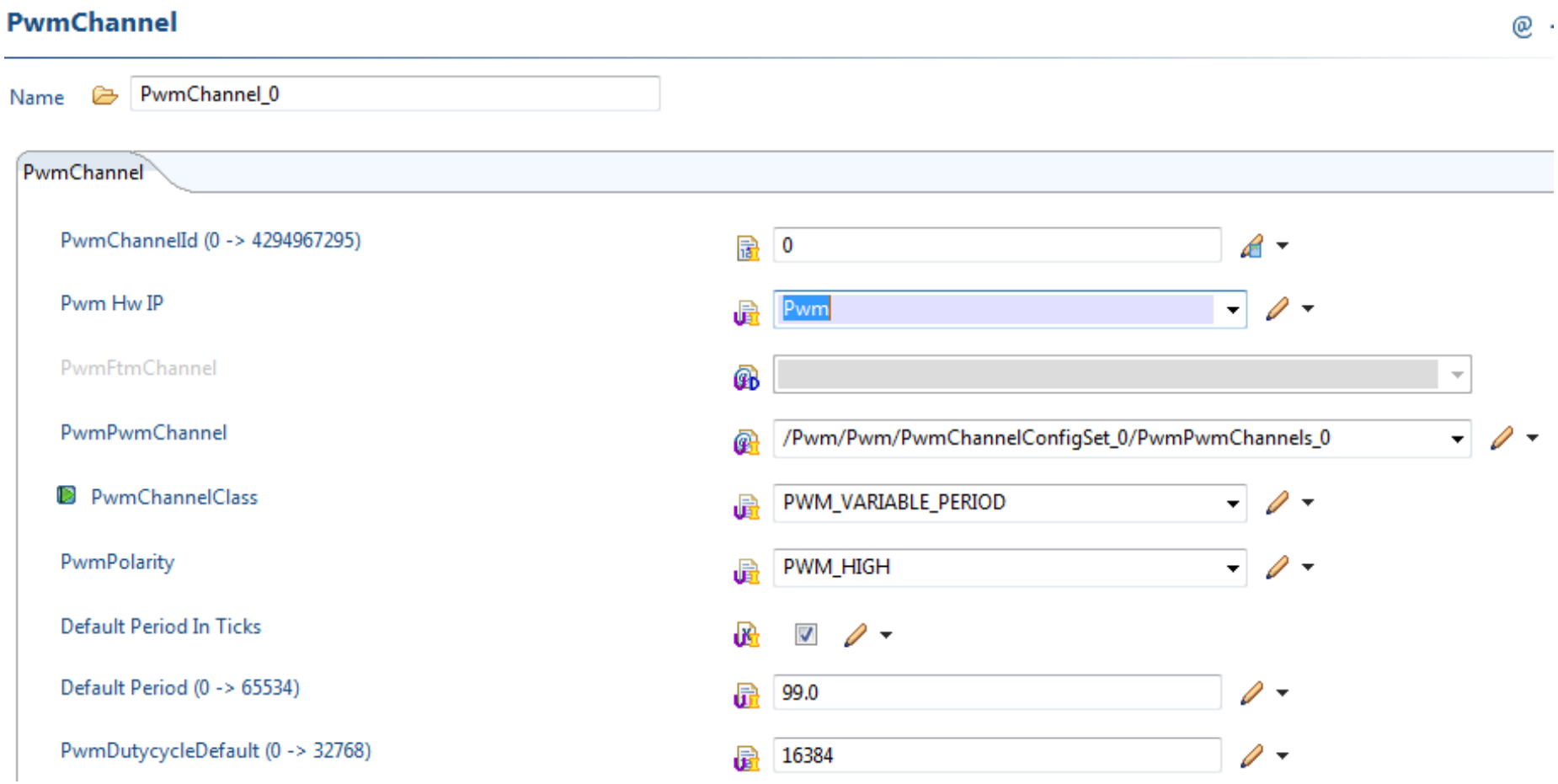
PWM Module – NXP Implementation



APIs in white are for ASR4.0, ASR4.2 and ASR4.3
APIs in yellow are new APIs in ASR4.2 and ASR4.3

PWM Channel Configuration

Parameters are the same between IMX6 and IMX8

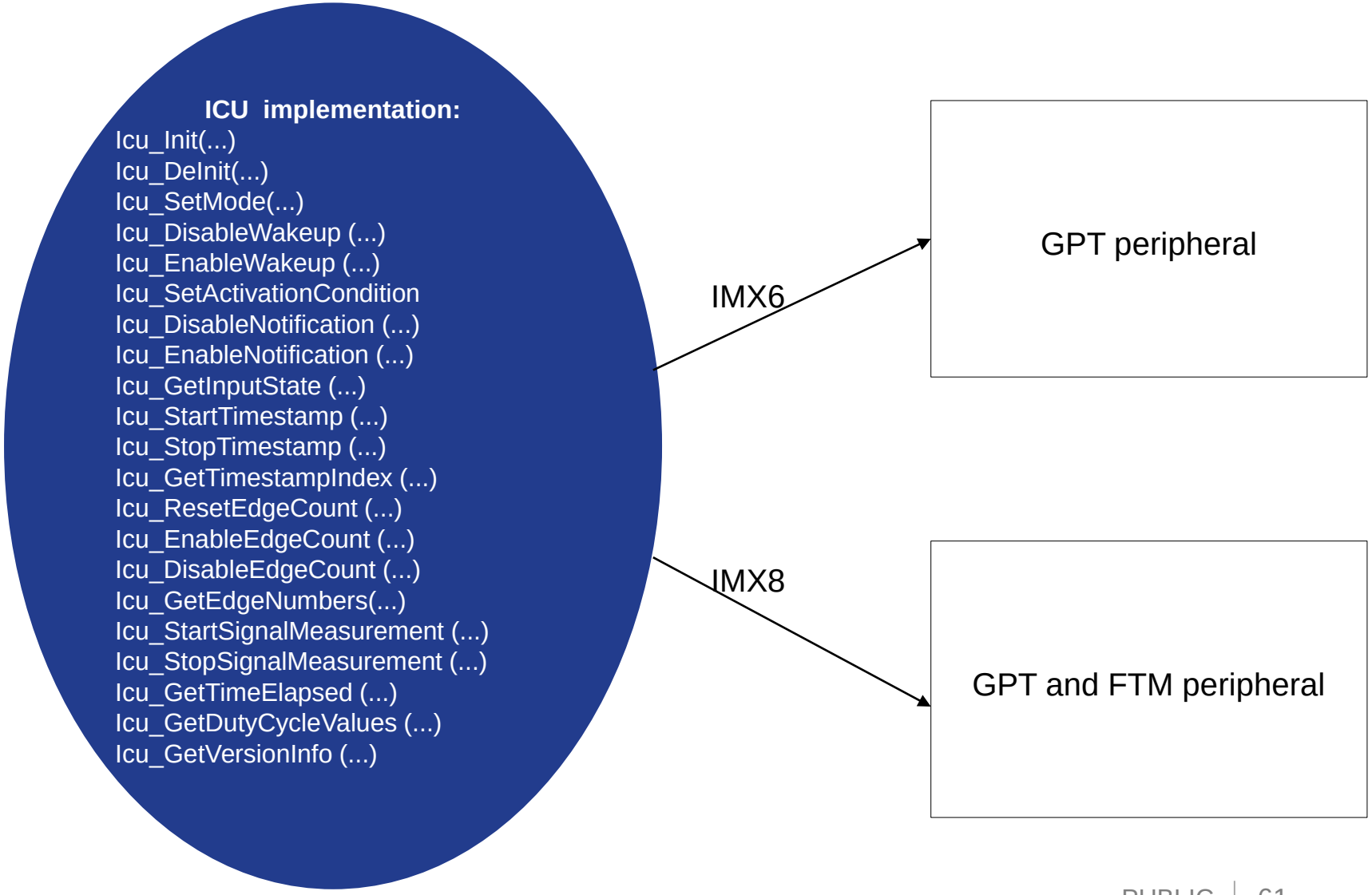


PWM Module Tips and Tricks

The user has to provide which channel and what duty cycle in Ticks “Time Units.”

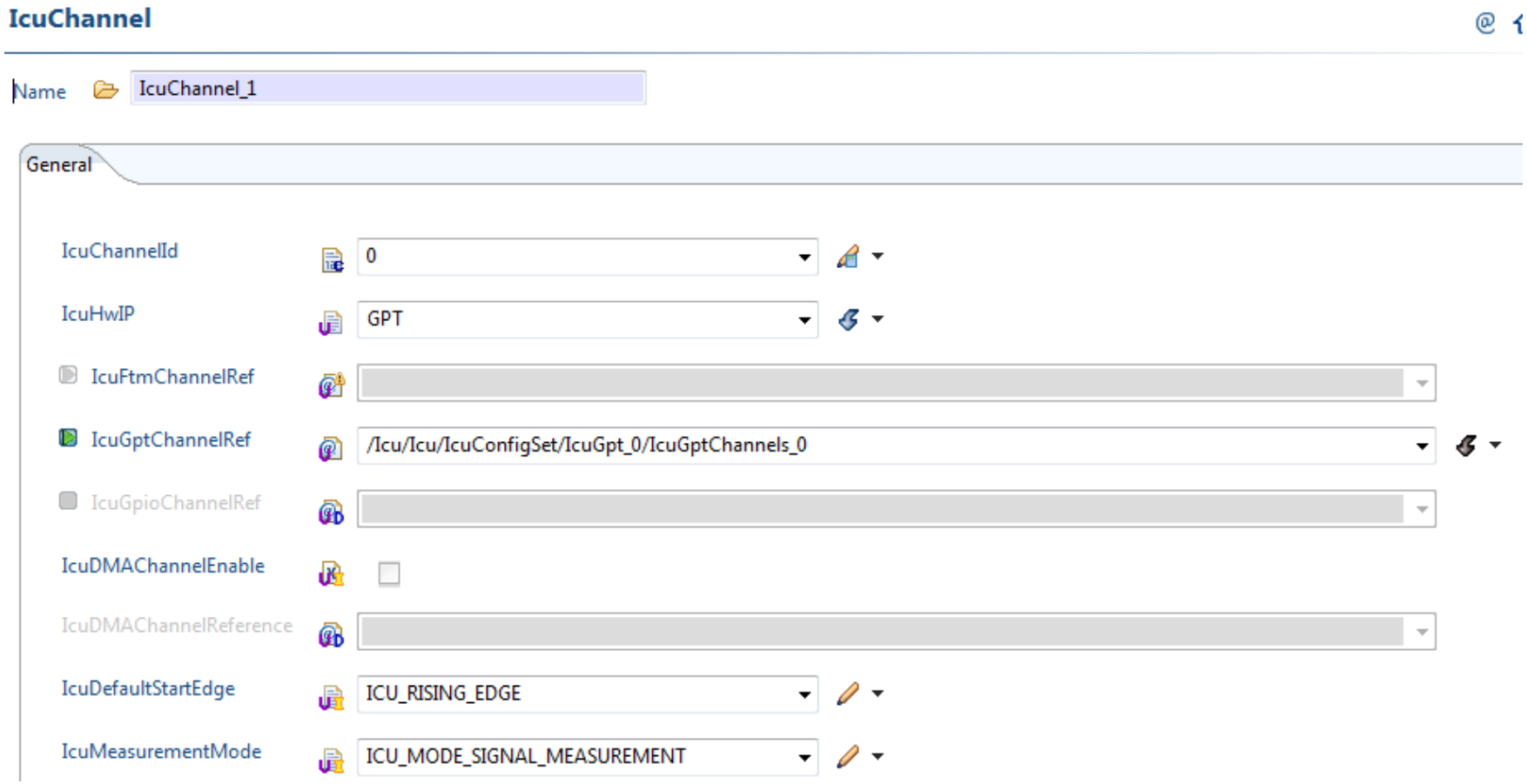
- Duty cycle ranges from 0x0000 (0%) to 0x8000(100%)
- This value will be converted to ticks internally by function Pwm_Pwm_ComputeDutycycle() based on the clock frequency and pre-scaler configured in the MCU Driver for PWM hardware or Pwm_FtmInternal_SetDutyRegs() for FTM hardware.

ICU Module – NXP Implementation



ICU Channel Configuration

Parameters are the same between IMX6 and IMX8



ICU Module Tips and Tricks

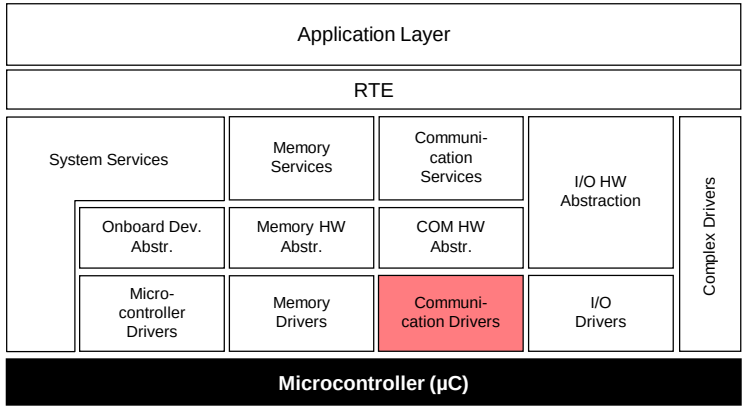
- If the register can affect several hardware modules and if it is an I/O register it shall be initialized by the PORT driver.
- If the register can affect several hardware modules and if it is not an I/O register it shall be initialized by the MCU driver.
- Therefore, Port and MCU Drivers shall be initialized first

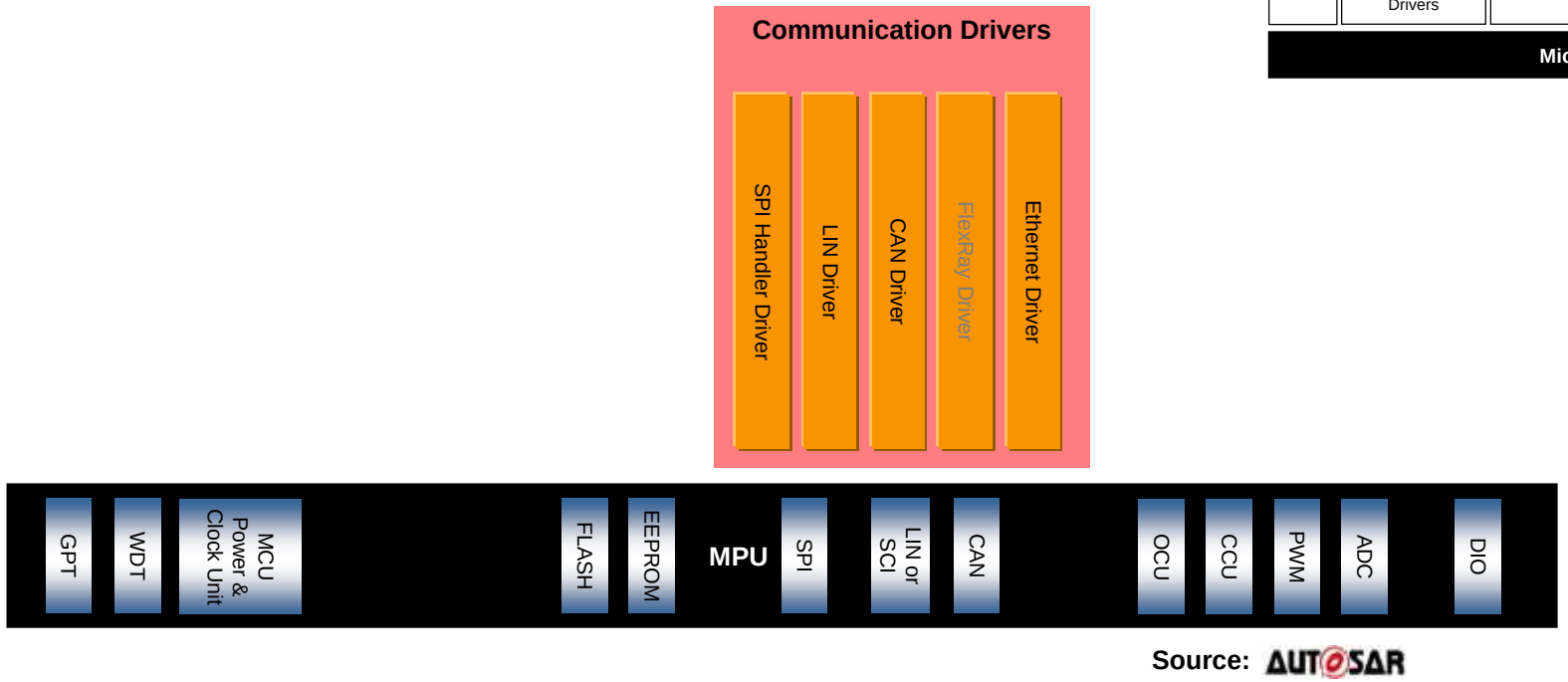
i.MX MCAL Communication Drivers

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



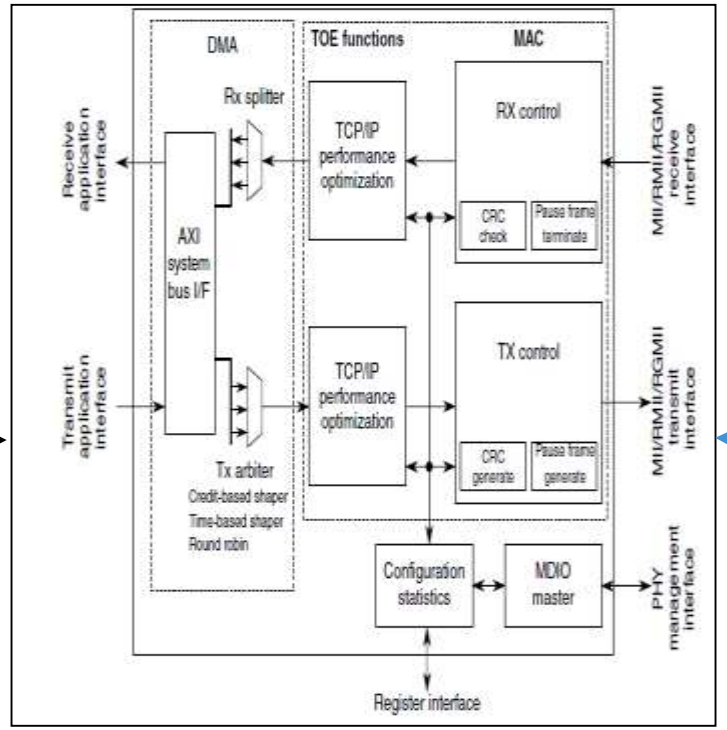


Source: AUTOSAR

Ethernet Module – NXP Implementation

Ethernet implementation:

Eth_Init (...)
Eth_ControllerInit (...)
Eth_SetControllerMode (...)
Eth_GetControllerMode (...)
Eth_GetPhysAddr (...)
Eth_WriteMii (...)
Eth_ReadMii (...)
Eth_GetCounterState (...)
Eth_ProvideTxBuffer (...)
Eth_Transmit (...)
Eth_Receive (...)
Eth_TxConfirmation (...)
Eth_GetVersionInfo (...)
Eth_SetPhysAddr (…)
Eth_UpdatePhysAddrFilter (…)
Eth_GetDropCount (…)
Eth_GetEtherStats (…)
Eth_GetCurrentTime (…)
Eth_EnableEgressTimeStamp (…)
Eth_GetEgressTimeStamp (…)
Eth_GetIngressTimeStamp (…)
Eth_SetCorrectionTime (…)
Eth_SetGlobalTime (…)

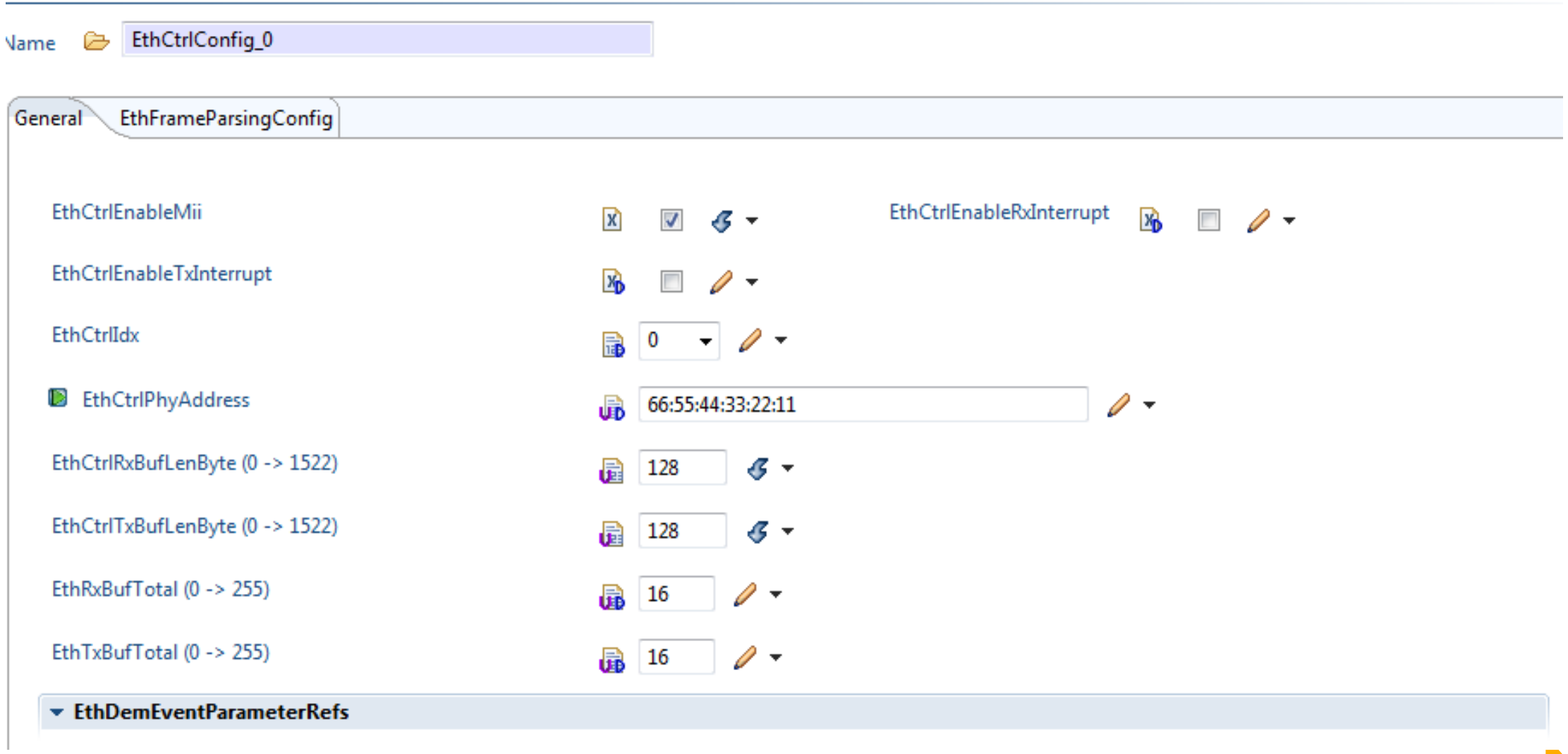


Eth_GetDropCount (…)
Eth_GetEtherStats (…)
Eth_SetCorrectionTime (…)
Eth_SetGlobalTime (…)
Eth_GetCounterValues (…)
Eth_GetRxStats (…)
Eth_GetTxStats (…)
Eth_GetTxErrorCounterValues (…)

APIs in white are for ASR4.0, ASR4.2, and ASR4.3
APIs in yellow are new APIs in ASR4.2 and ASR4.3
APIs are in red are removed from ASR4.2 and ASR4.3

As comparing with ASR4.2
APIs in red are removed from ASR4.2
APIs in yellow are new APIs in ASR4.3

ETH Controller Configuration in EB Tresos



Ethernet Tips and Tricks

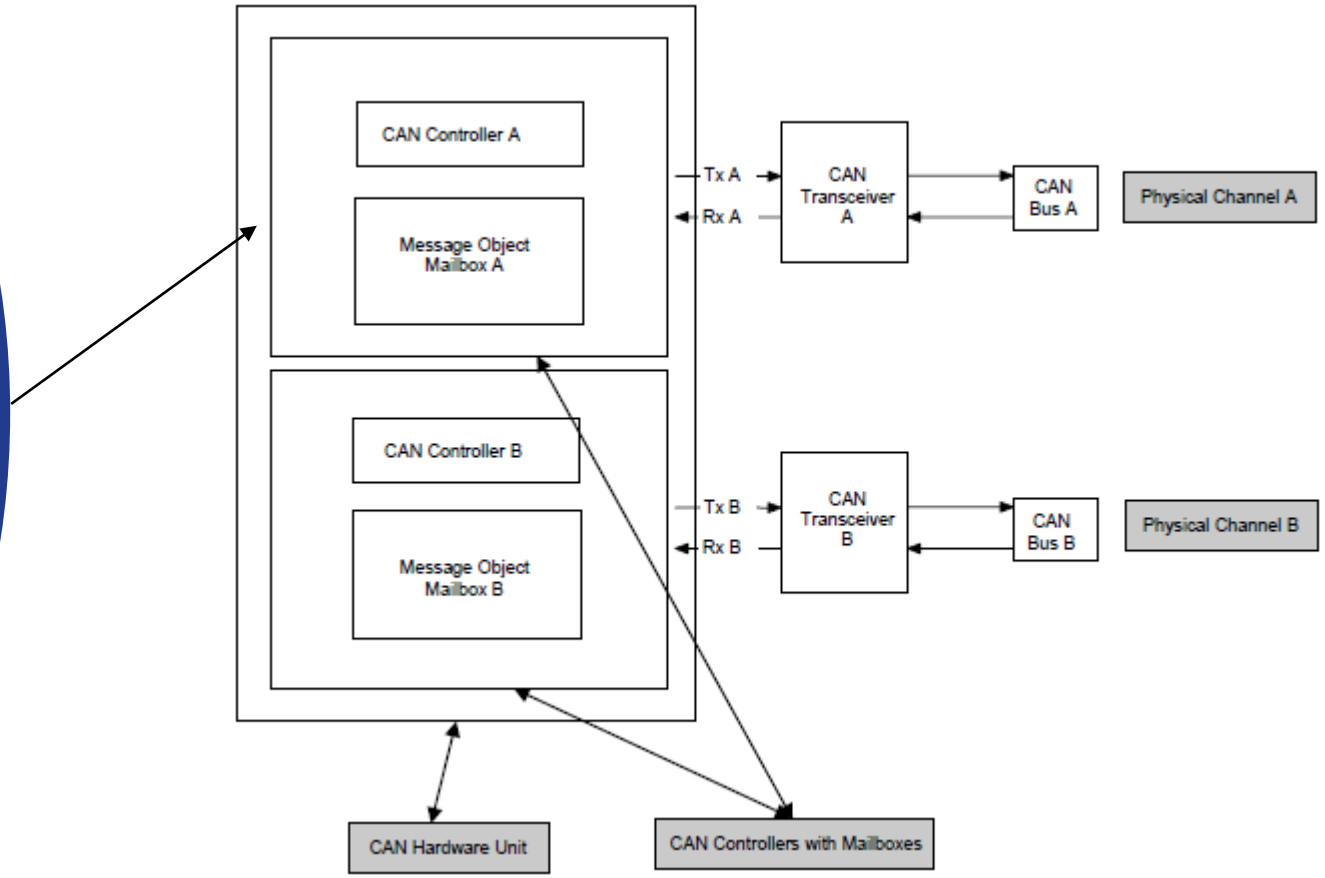
- Only one Ethernet Controller is supported since there is only one Ethernet. The Controller index is always ignored because only one Ethernet Controller is supported.

- Frame reception can be done either in the poll driven mode or in the interrupt driven mode but the mixture is not possible. It means that the function Eth_Receive shall not be called if the receive interrupt is enabled and vice versa.

- Frame transmission confirmation can be done either in the poll driven mode or in the interrupt driven mode but the mixture is not possible. It means that the function Eth_43_Fsl_TxConfirmation shall not be called if the transmit interrupt is enabled and vice versa.

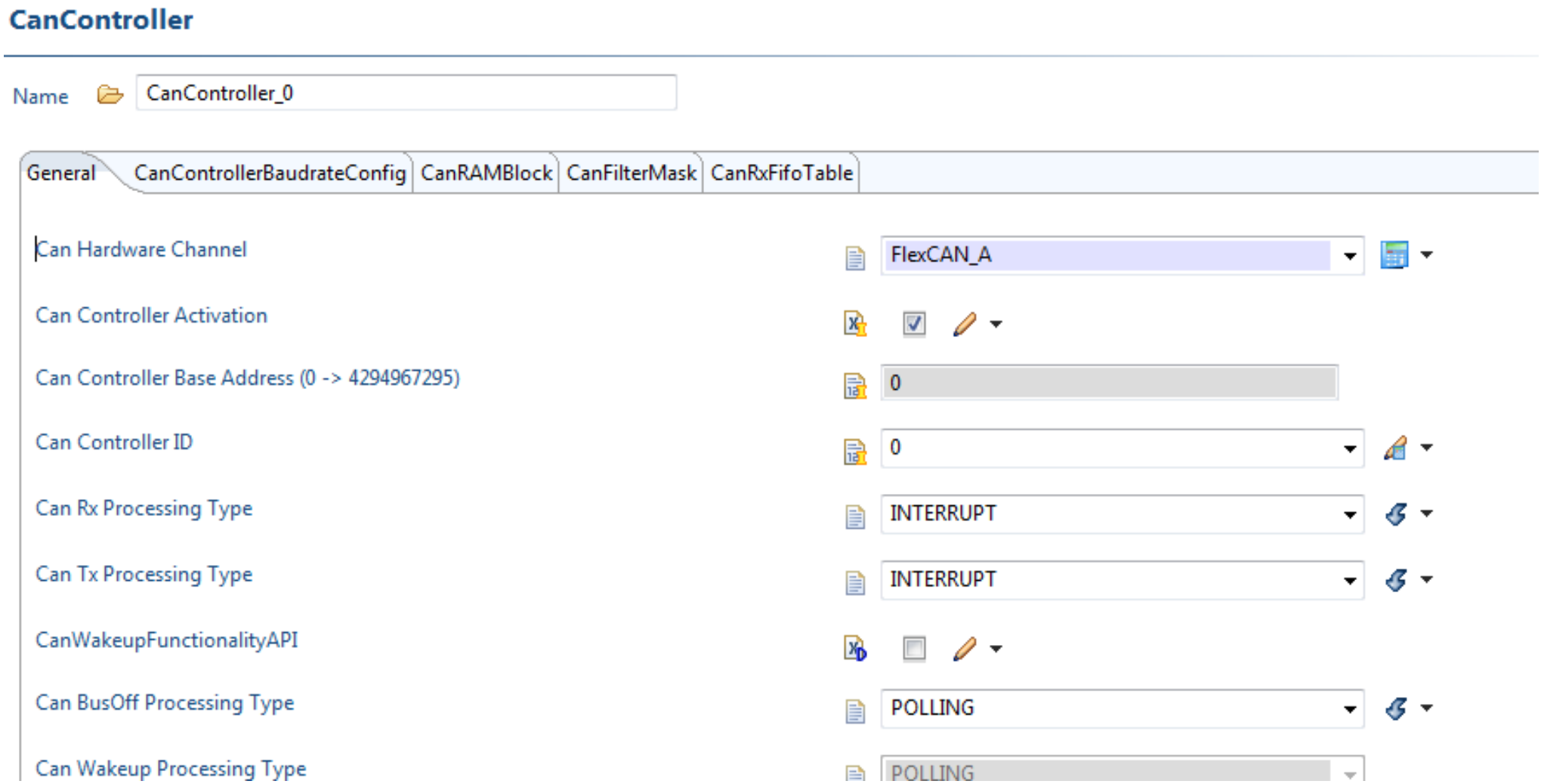
CAN Module – NXP Implementation

CAN implementation:
Can_Init (...)
Can_ChangeBaudrate (...)
Can_SetControllerMode (...)
Can_EnableControllerInterrupts (...)
Can_CheckWakeup (...)
Can_GetVersionInfo (...)
Can_CheckBaudrate (...)
Can_DisableControllerInterrupts (...)
Can_Write (...)
Can_MainFunction_Write (...)
Can_MainFunction_Read (...)
Can_MainFunction_BusOff (...)
Can_MainFunction_Wakeup (...)
Can_MainFunction_Mode (...)
Can_SetBaudRate (...)
Can_GetControllerErrorState (…)
Can_GetControllerMode (…)
Can_DeInit (…)



APIs in white are for ASR4.0, ASR4.2, and ASR4.3
APIs in yellow are new APIs in ASR4.3
APIs are in red are removed from ASR4.3

CAN Configuration in EB Tresos



New Feature in New Version FlexCAN Module – CAN FD

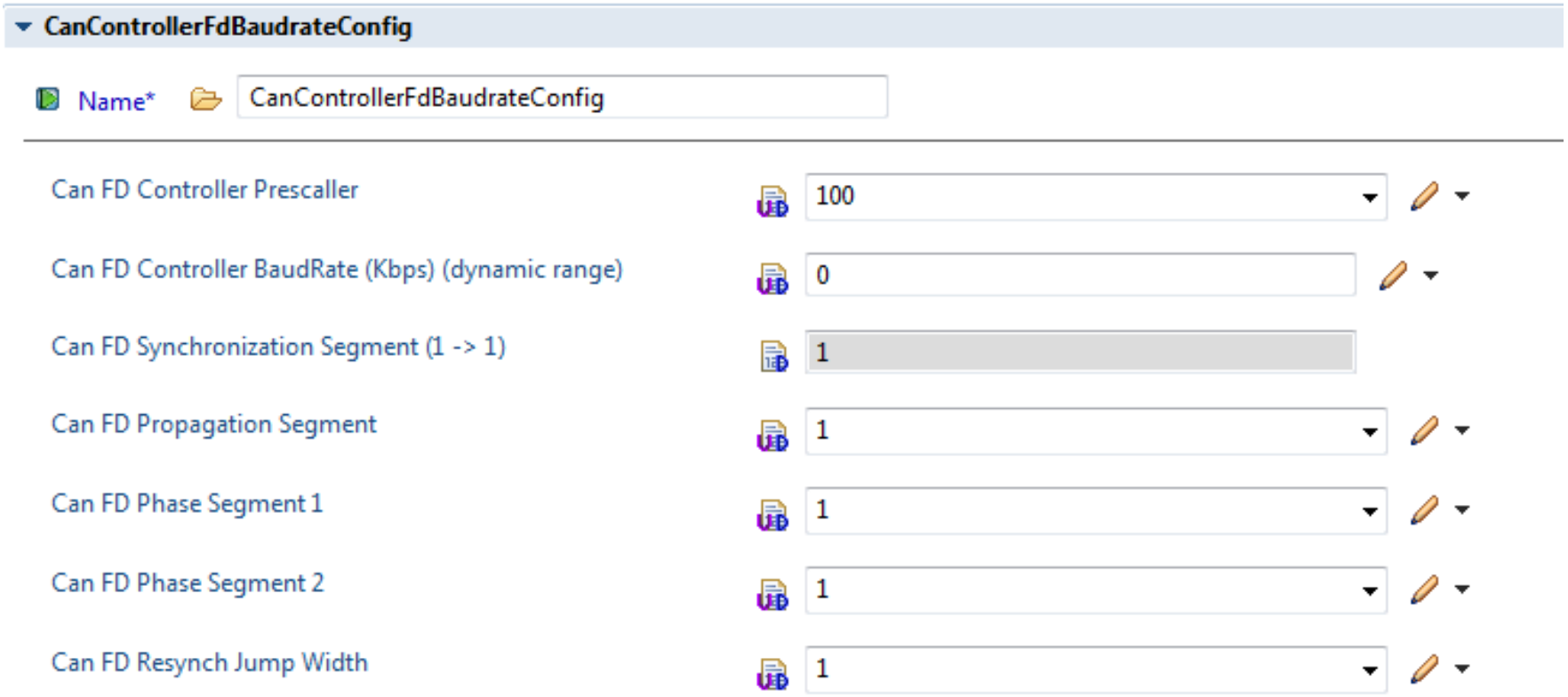
- Full implementation of:
  - CAN protocol specification and the CAN 2.0 version B protocol. Both support standard and extended message frames
  - CAN with Flexible Data Rate (CAN FD) protocol specification that support long payloads up to 64 bytes transferred at faster rates up to 8 Mbps

- Flexible mailboxes configurable to store 0 to 8, 16, 32 or 64 bytes data length
- Each mailbox configurable as receive or transmit, all supporting standard and extended messages

100% backward compatibility with previous FlexCAN version

Enable CAN FD Feature in EB Tresos

- Activate CAN FD feature by enable CanControllerFDBaudrateConfig container
- Data baudrate in CAN FD frame should be configured in this container

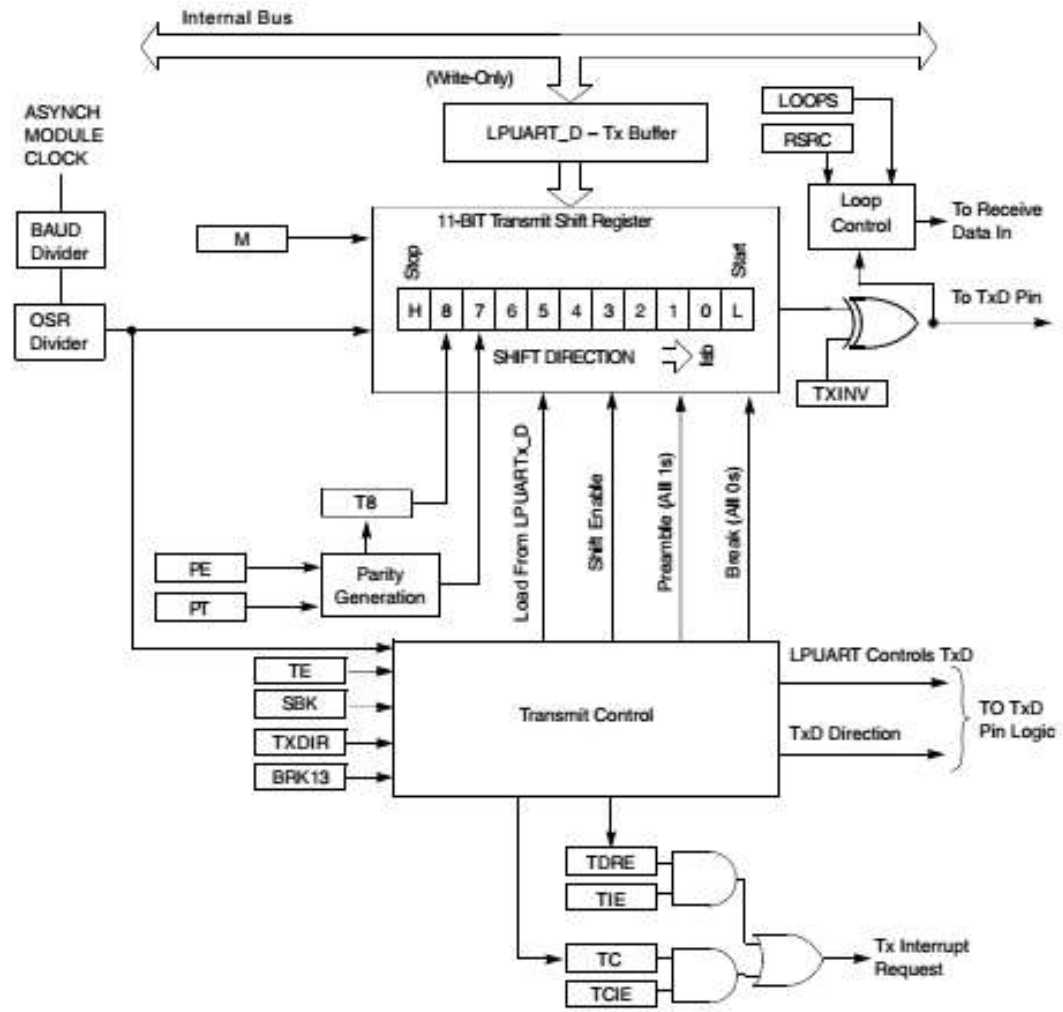


CAN Module Tips and Tricks

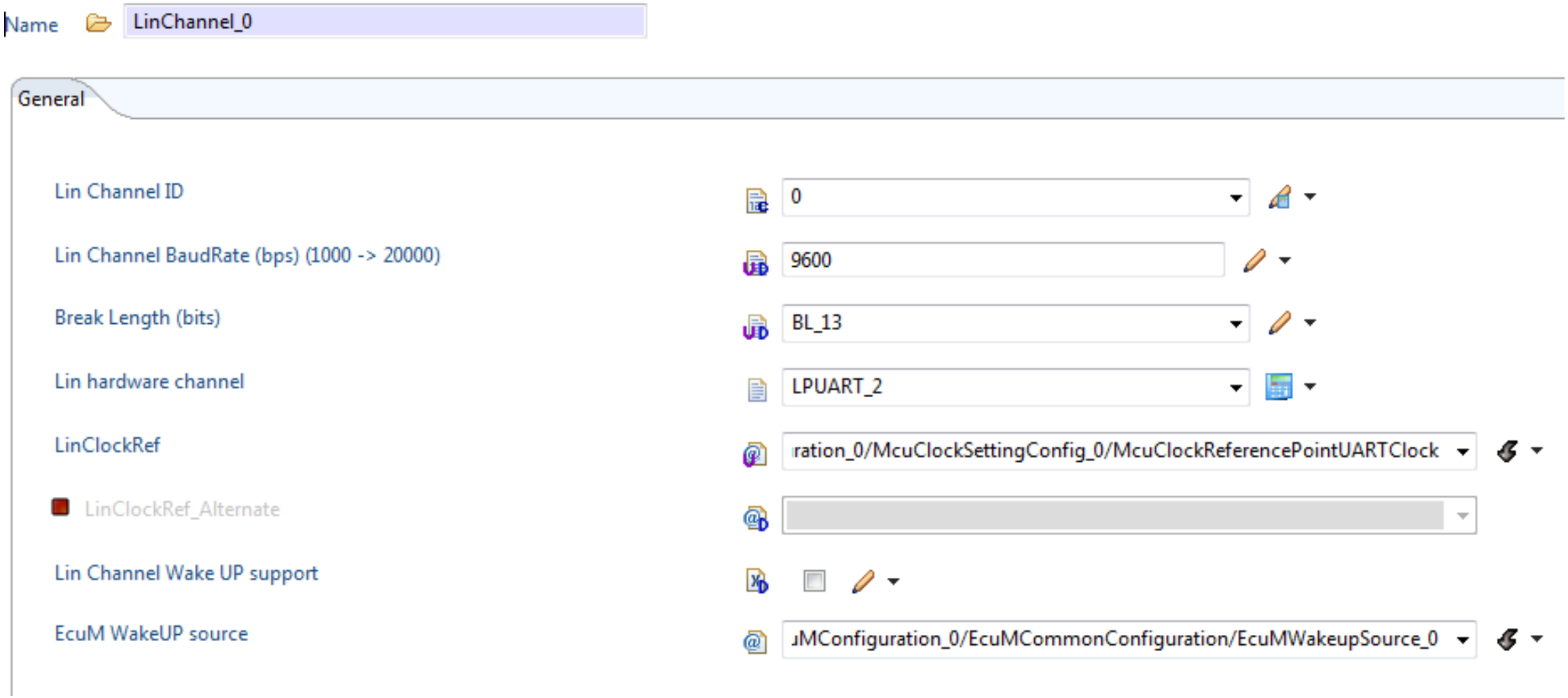
- The driver does not distinguish between “Extended” and mixed (Standard/Extended) MB types for receiving. All Rx MBs configured as MIXED type will be converted to EXTENDED type

LIN Module – NXP Implementation

LIN implementation:
Lin_Init (...)
Lin_CheckWakeup (...)
Lin_GetVersionInfo (...)
Lin_SendFrame (...)
Lin_GoToSleep (...)
Lin_GoToSleepInternal (...)
Lin_Wakeup (...)
Lin_WakeUpInternal (...)
Lin_GetStatus (...)



LIN Configuration in EB Tresos

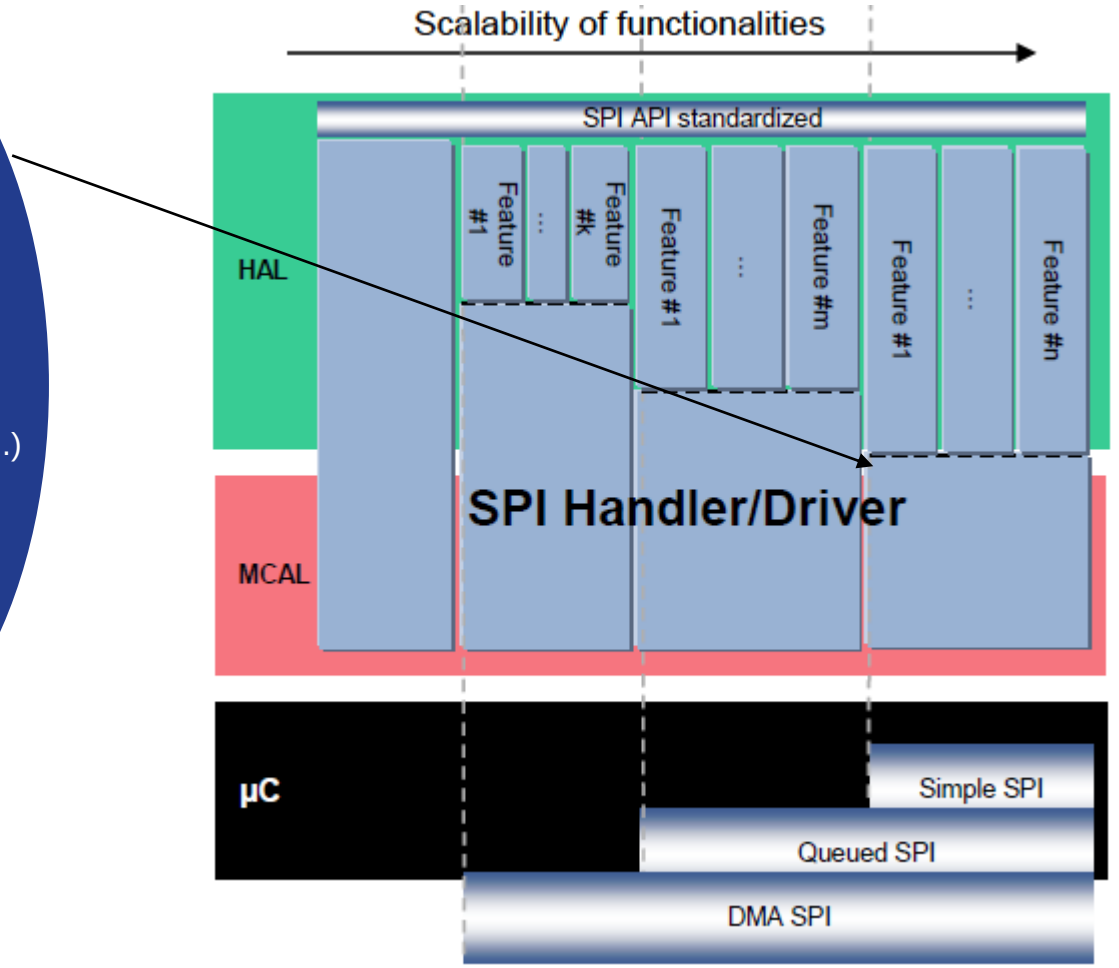


LIN Tips and Tricks

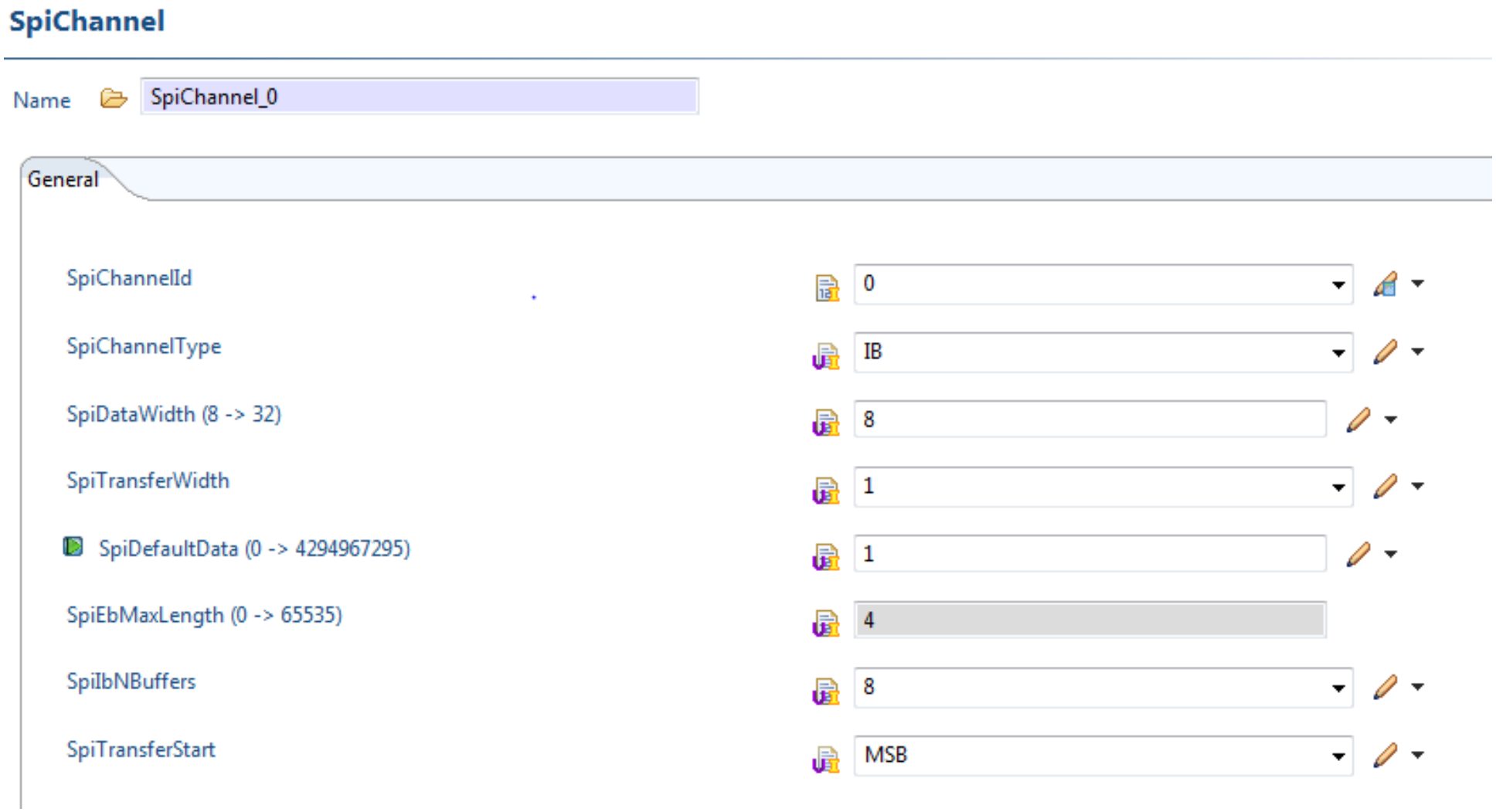
According to AUTOSAR MCAL LIN Driver requirements, LIN Driver only supports MASTER mode

SPI Module – NXP Implementation

SPI implementation:
Spi_Init(...)
Spi_DeInit (...)
Spi_ReadIB (...)
Spi_WriteIB (...)
Spi_AsyncTransmit(...)
Spi_SetupEB(...)
Spi_GetStatus (...)
Spi_GetJobResult (...)
Spi_GetSequenceResult (...)
Spi_GetVersionInfo (...)
Spi_SyncTransmit (...)
Spi_GetHWUnitStatus (...)
Spi_Cancel (...)
Spi_SetAsyncMode (...)



SPI Configuration in EB Tresos



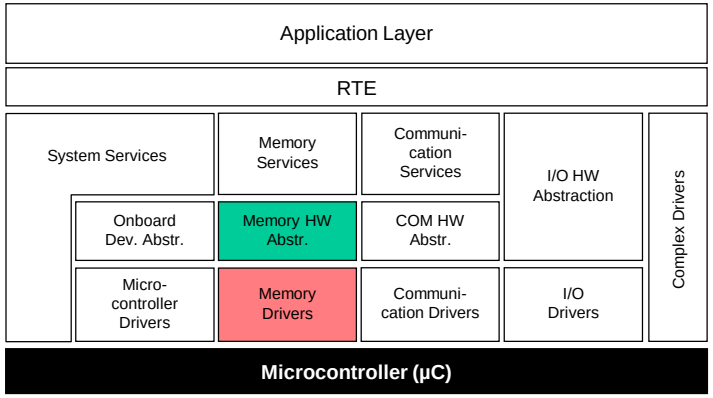
SPI Module Tips and Tricks

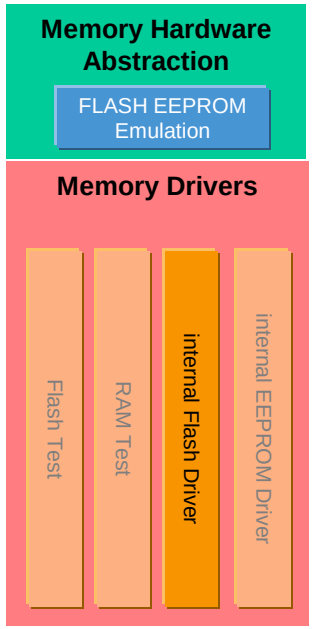
- CS Options:
  - via SPI HW Engine: default- used when CS has to be enabled during data transfer based on the burst length(8/16/32.)
  - via GPIO: used when the CS has to be enabled throughout a job to control the data transfer in SPI protocol. The user has to use SpiJobStartNotification' & 'SpiJobEndNotification' to toggle the CS (chip select pin) using DIO drivers .
- SPI Driver can be configured as
  - **Synchronous** **-** FIFO policy to handle multiple access-es.
  - **Asynchronous -** Priority policy to handle multiple access-es.
  - **Enhanced (**Synchronous/ Asynchronous**) -** Either by interrupts or polling mechanism - selectable during execution time and with a Priority policy to handle multiple accesses.

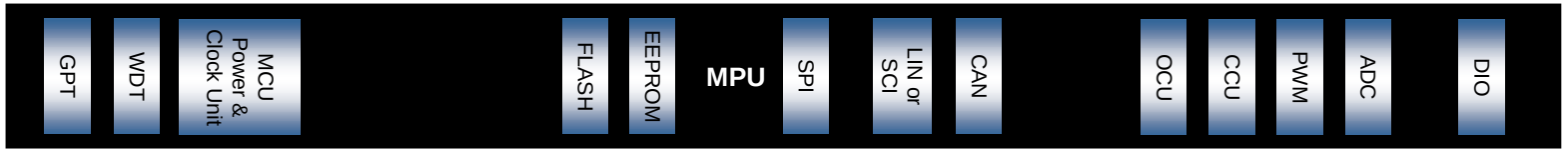
i.MX MCAL Memory Drivers

AUTOSAR — Microcontroller Abstaction Layer

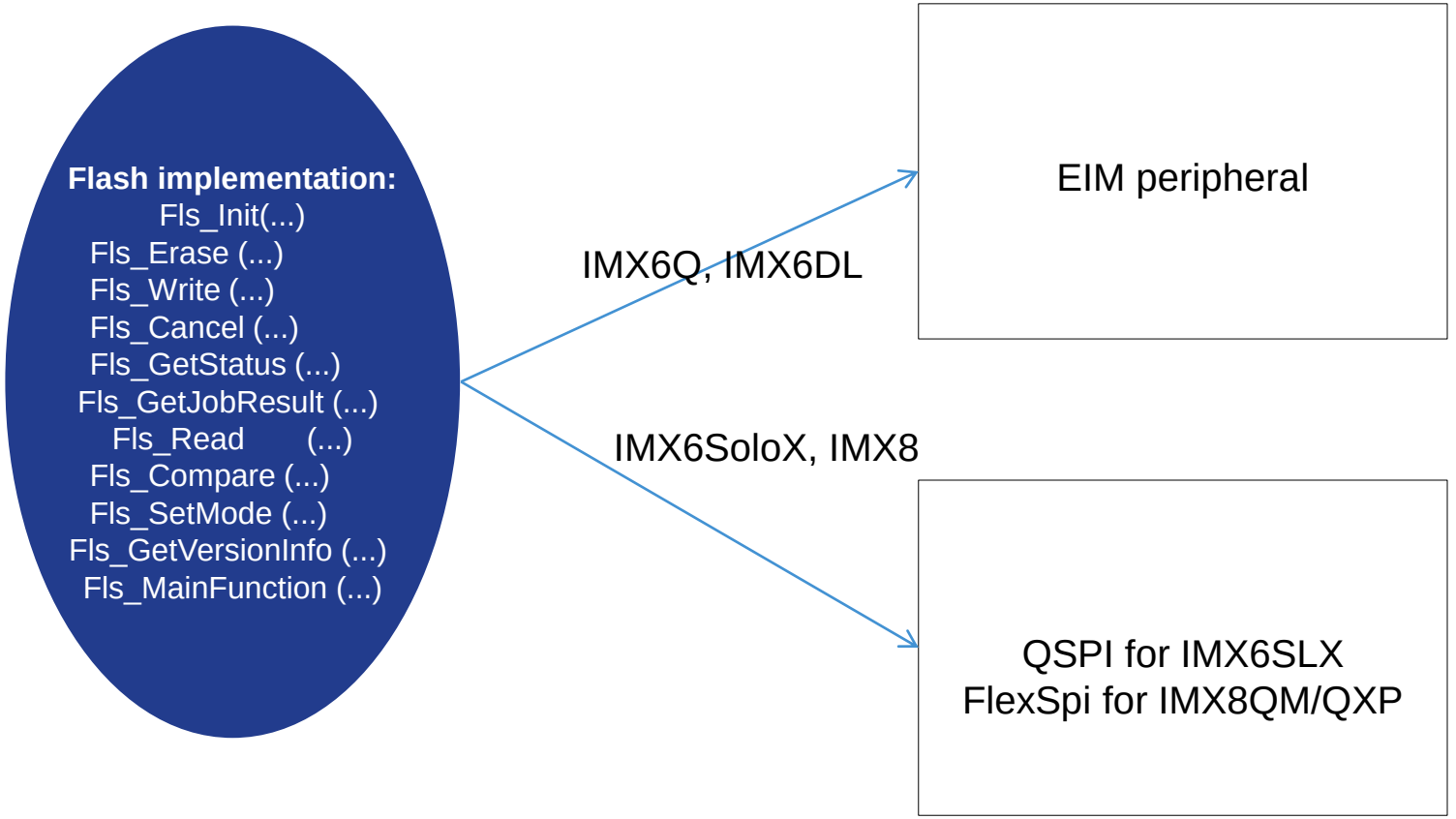
- Memory Drivers
  - The Memory Hardware Abstraction is a group of modules which abstracts from the location of peripheral memory devices (on-chip or on-board) and the ECU hardware layout
  - The Memory Drivers are accessed via memory specific abstraction/emulation modules (e.g. EEPROM Abstraction)



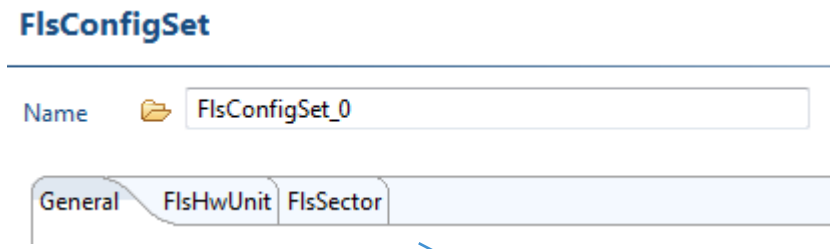




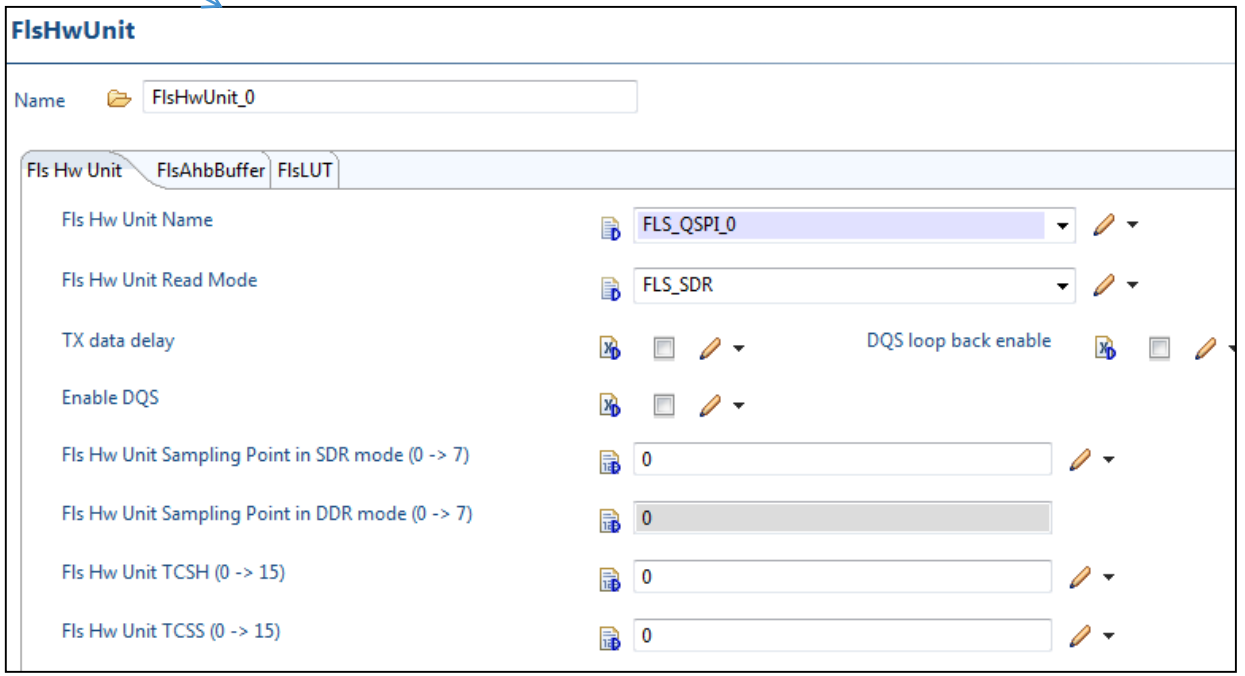
Flash Module – NXP Implementation Per External Memory



Flash Config Set



FlsHwUnit is used for QSPI configuration

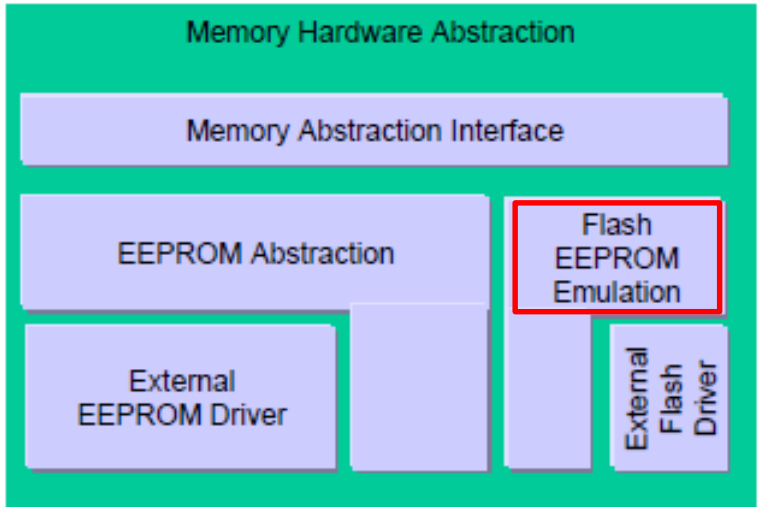


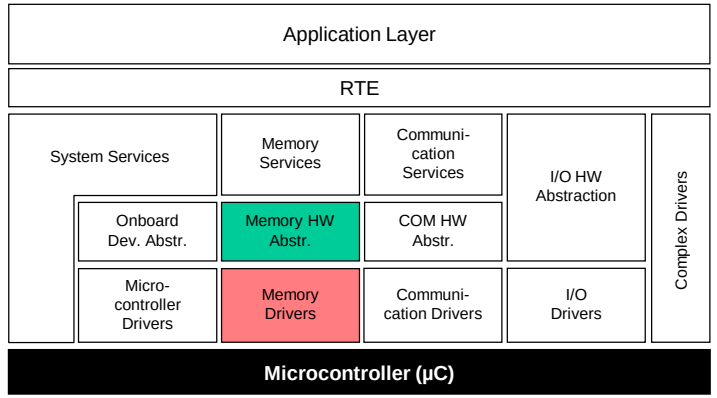
Flash Module Tips and Tricks

- Implementation per AUTOSAR is to erases only one sector per cycle but the HW allows erasing of more physical sectors in parallel.
- For EIM hardware, FLS driver is designed to communicate between FLS driver and NOR flash via command APIs
- For QSPI hardware, FLS driver is designed to communicate between FLS driver and NOR flash via command tables (LUT) of user configuration under EB tresos tool.

AUTOSAR — Microcontroller Abstaction Layer

- Memory Drivers
  - FEE is located in the “Memory Hardware Abstraction” layer which falls under the ECU Abstraction Layer





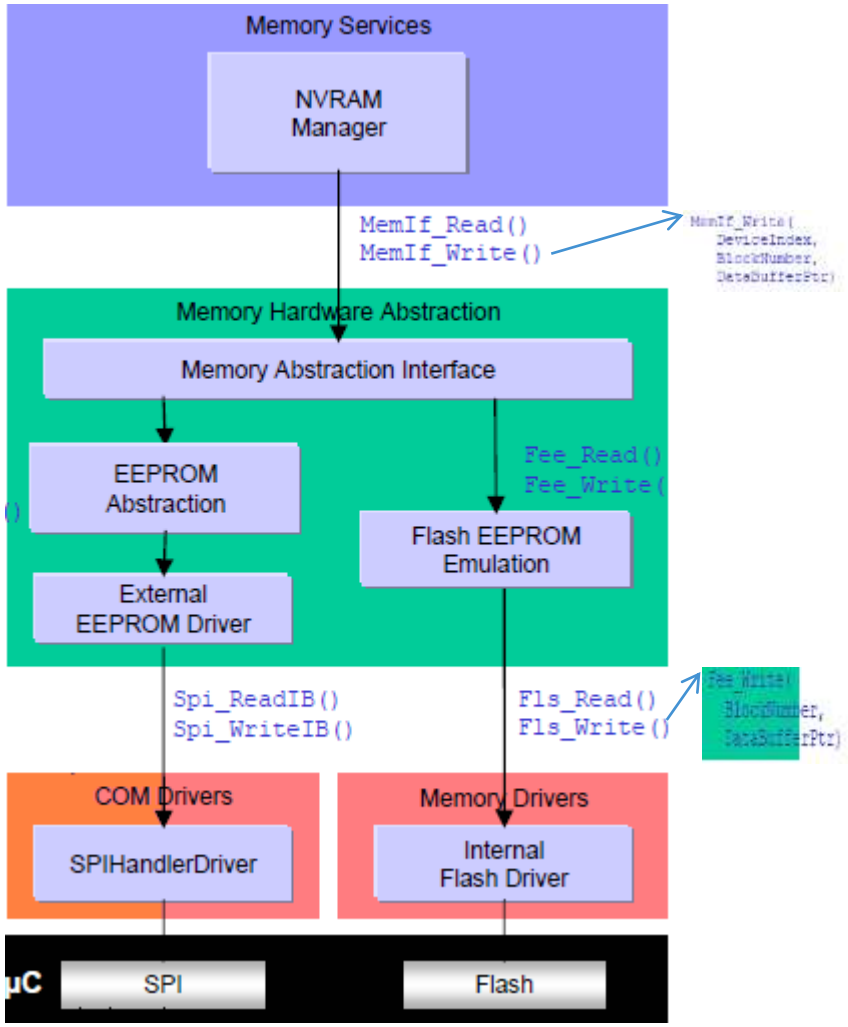
Source: AUTOSAR

FEE Module – Functional Overview

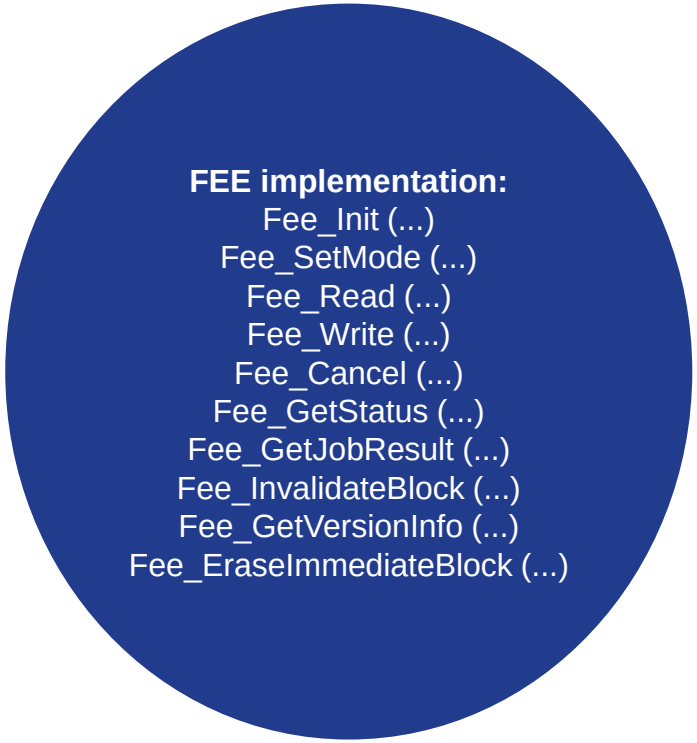
- NVRAM Manager accesses drivers via Memory Hardware Abstraction Interface. It addresses different memory devices using a device index.

The Memory Hardware Abstraction Interface performs the following operations.

- Routing during runtime based on device index (Int\Ext)
- Routing during runtime based on block index
- The Memory Drivers are low level drivers like FLS which performs the actual write/read/erase operations on microcontroller hardware



FEE Module – NXP Implementation Per Flash Memory



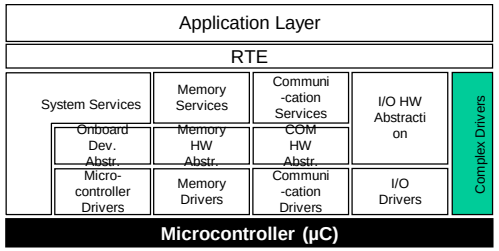
FEE Module Tips and Tricks

- FLS Module should be initialized before usage of any FEE services
- FEE is a Hardware independent module and it does not directly interact with microcontroller hardware(s) instead calls will be executed by lower layer FLS module

i.MX MCAL Complex Drivers

AUTOSAR — Complex Device Drivers

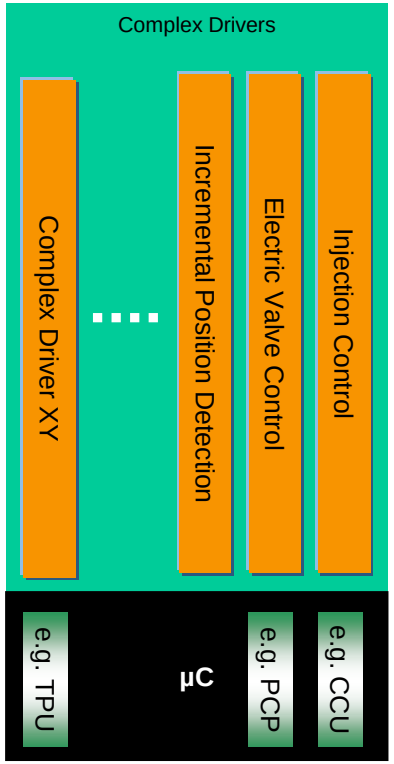
A **Complex Driver** is a module which implements non-standardized functionality within the basic software stack.



Examples:

- USB
- I2C
- I2S
- RPMSG
- SDIO
- Fault Monitoring Drivers
- Core and Peripheral Self Tests
- MicroController Library (MCL) – eDMA management
- CRC Driver

Example:



Source: AUTOSAR

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

Complex Drivers/Services

Autosar Complex Drivers / Custom Software Development, Migration, Testing, Integration, Consulting

Complex Driver Development

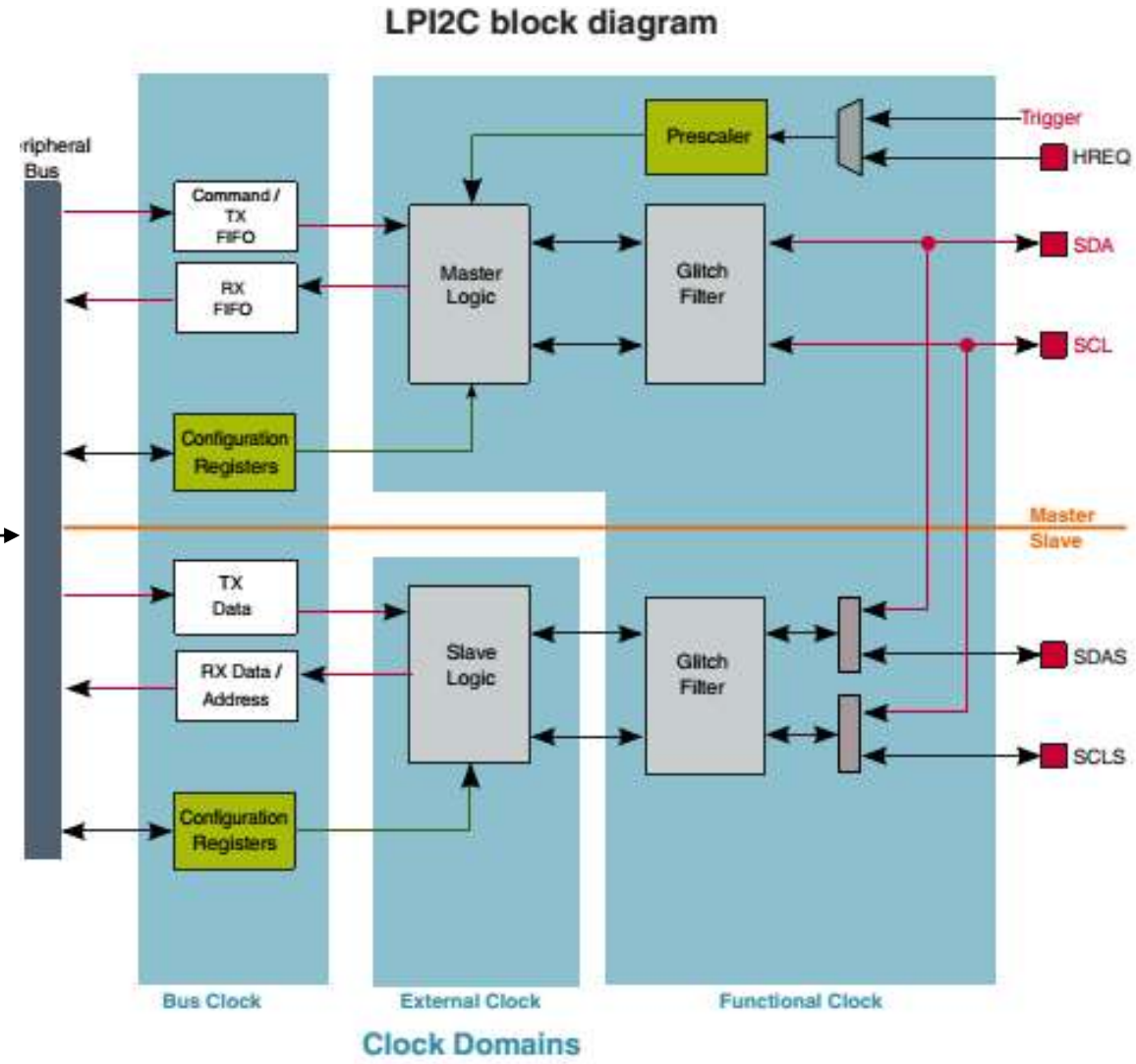
- Automotive Software Services Engagement Process:
  - Gather Customer Requirements
  - Scope the Engagement Effort
  - Create SOW Proposal for Review and Approval
  - Execute the Project and Test the Deliverables per the Agreed Test Plan
  - Deliver on Time and on Budget

Autosar Service Offerings

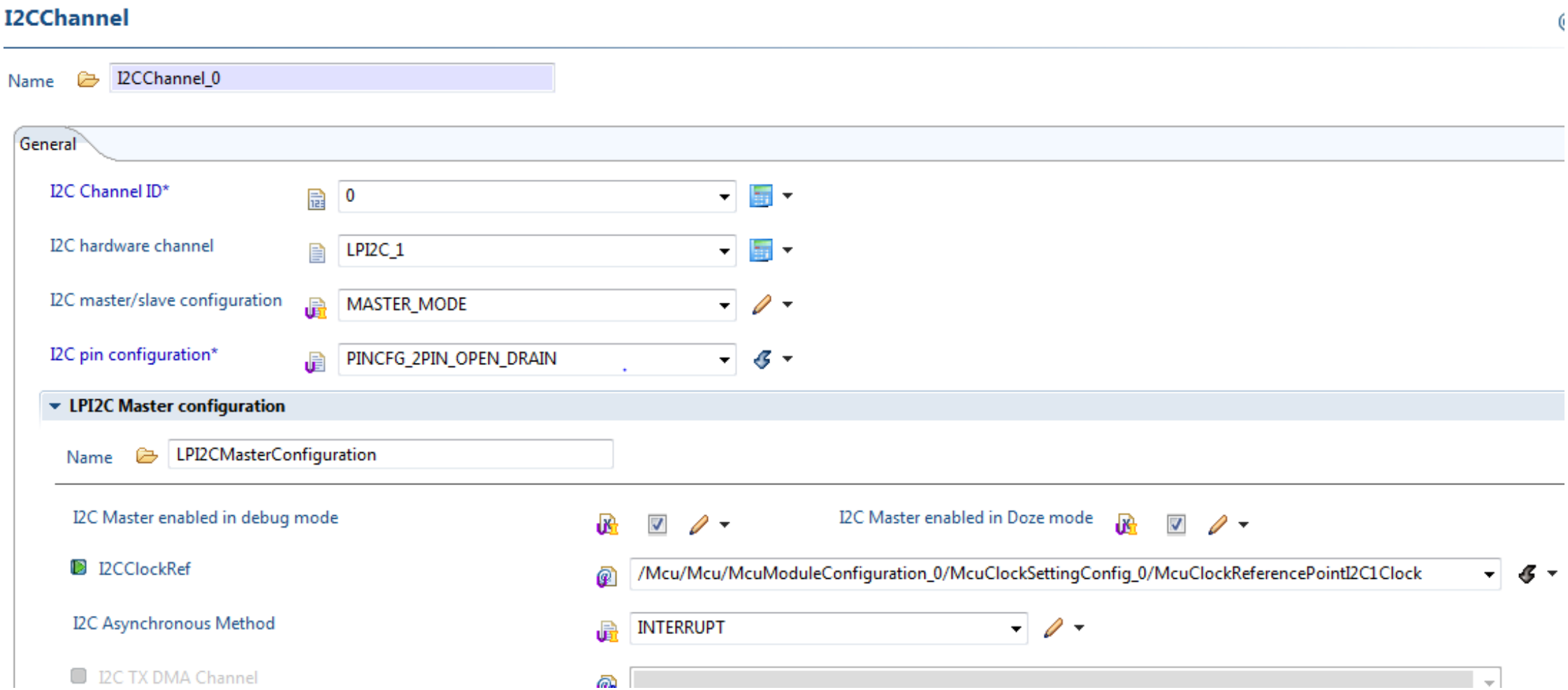
- System Consulting
- Integration
- Configuration
- Custom Development
- Migration
- Modeling and Automatic Code Generation
- Performance Analysis & Optimization

I²C Module – NXP Implementation

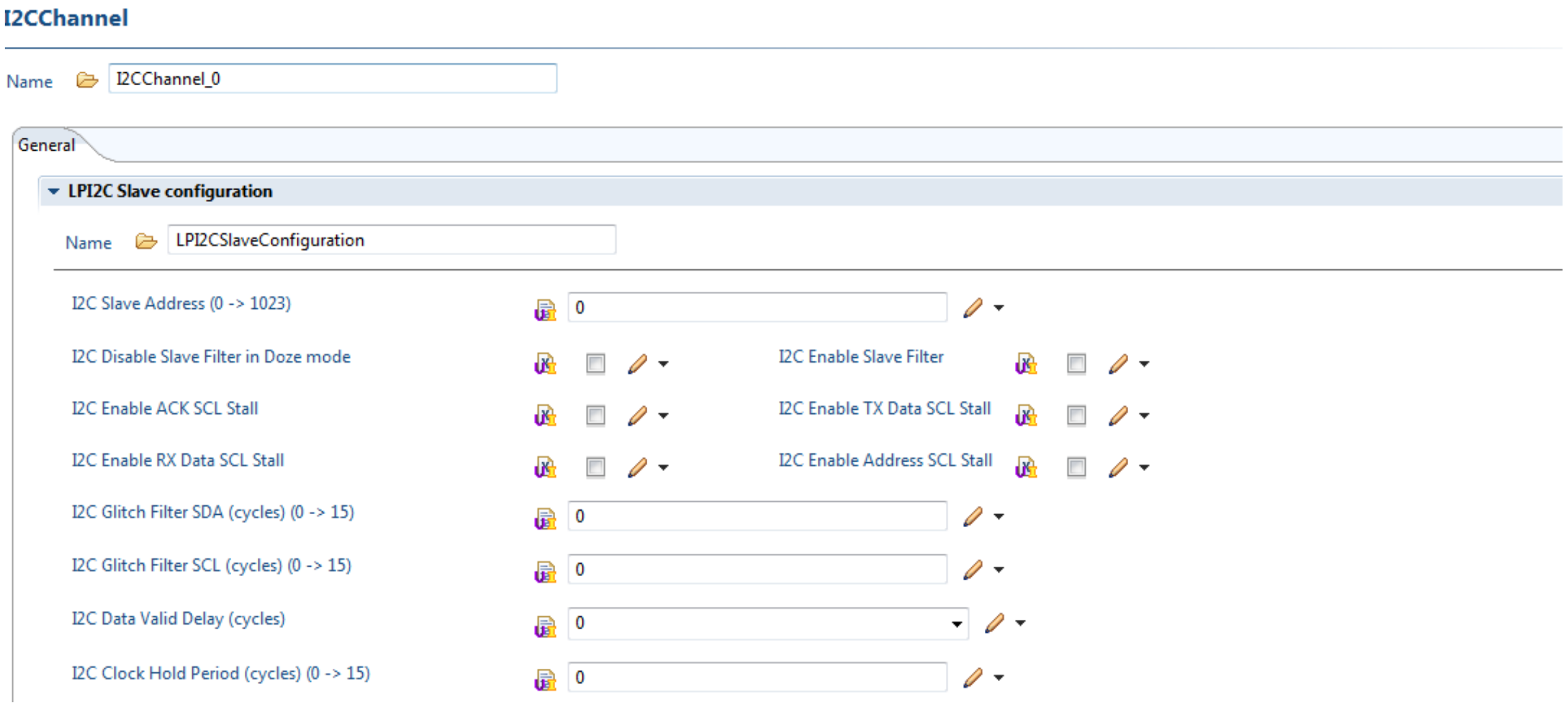
I2C implementation:
I2c_43_Fsl_Init(...)
I2c_43_Fsl_DeInit(...)
I2c_43_Fsl_SyncTransmit(...)
I2c_43_Fsl_AsyncTransmit(...)
I2c_43_Fsl_GetStatus(...)
I2c_43_Fsl_PrepareSlaveBuffer(...)
I2c_43_Fsl_GetVersionInfo(...)



I²C Master Configuration



I²C Slave Configuration



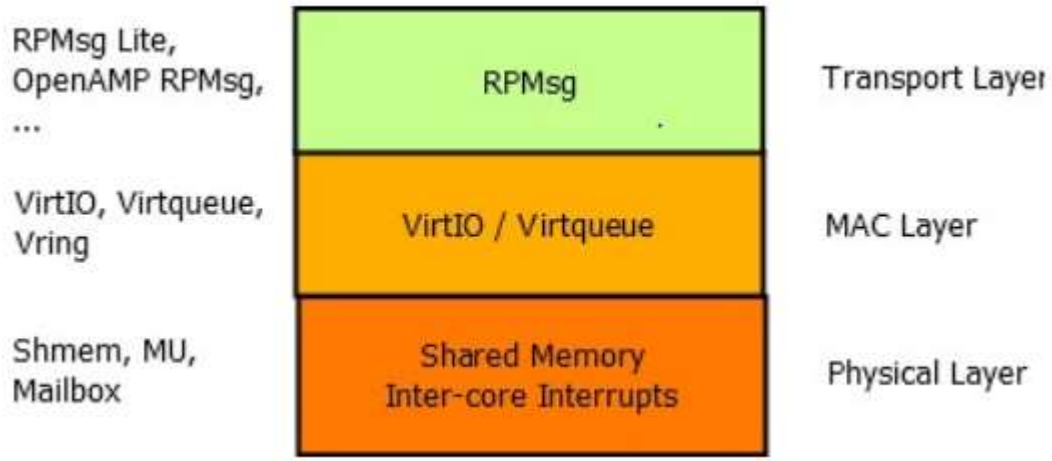
I²C Module Tips and Tricks

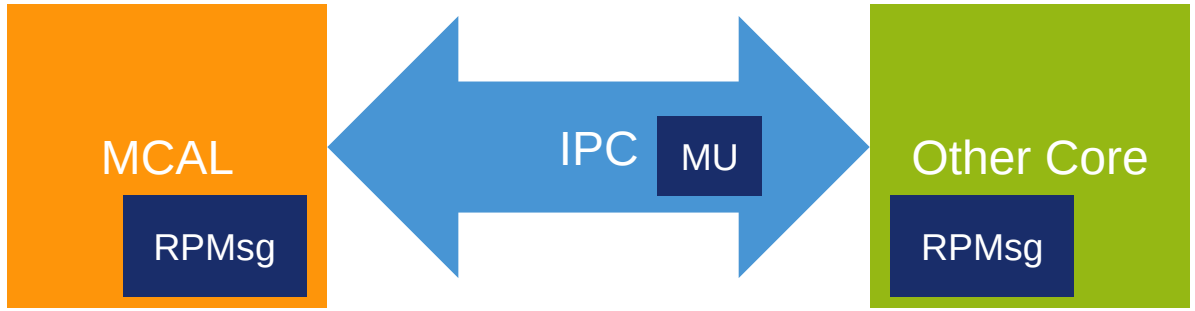
- I²C Driver supports both MASTER and SLAVE mode
  - In MASTER mode, DMA can be enabled to reduce CPU overhead when transmitting and receiving data

RPMsg Module – Functional Overview

- **RPMsg** (Remote Processor Messaging) is a protocol enabling inter-core communication in heterogeneous multicore systems

- In i.MX8 families, The Messaging Unit acts as a mean of physical medium to enables two processors within the SoC to communicate and coordinate by passing messages (e.g. data, status and control) through the MU interface.
- The MU also provides the ability for one processor to signal the other processor using interrupts.





RPMsg Module – NXP Implementation

Rpmsg implementation:
Rpmsg_43_Nxp_Init(...)
Rpmsg_43_Nxp_DeInit(...)
Rpmsg_43_Nxp_Announce(...)
Rpmsg_43_Nxp_GetStatus(...)
Rpmsg_43_Nxp_Send(...)
Rpmsg_43_Nxp_SendNoCopy(...)
Rpmsg_43_Nxp_CreateEndpoint(...)
Rpmsg_43_Nxp_DestroyEndpoint(...)
Rpmsg_43_Nxp_WakeupRemote(...)

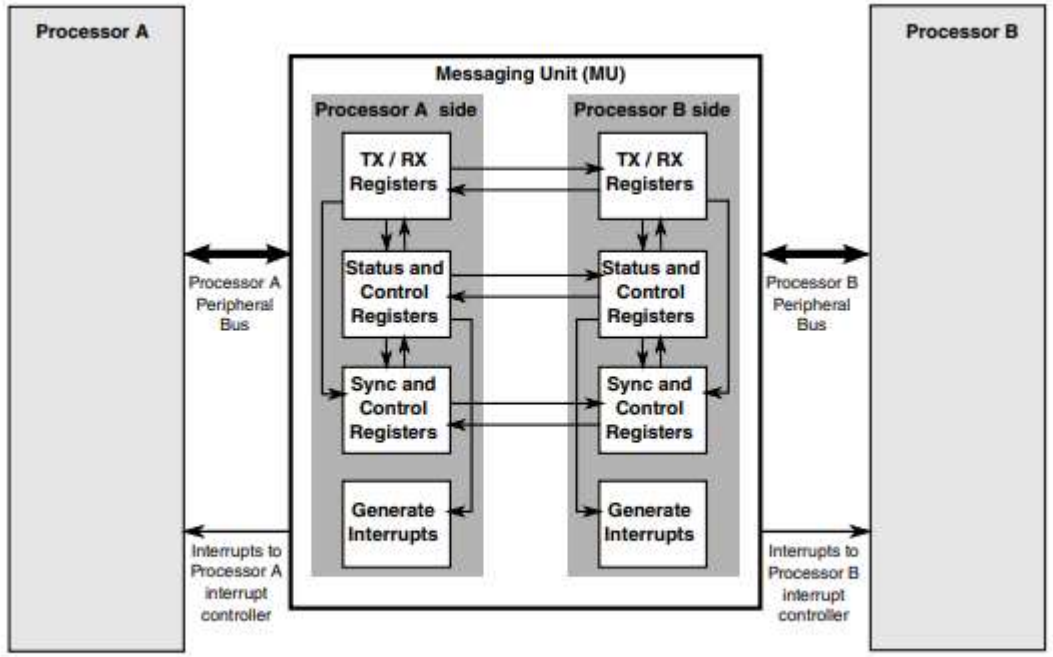
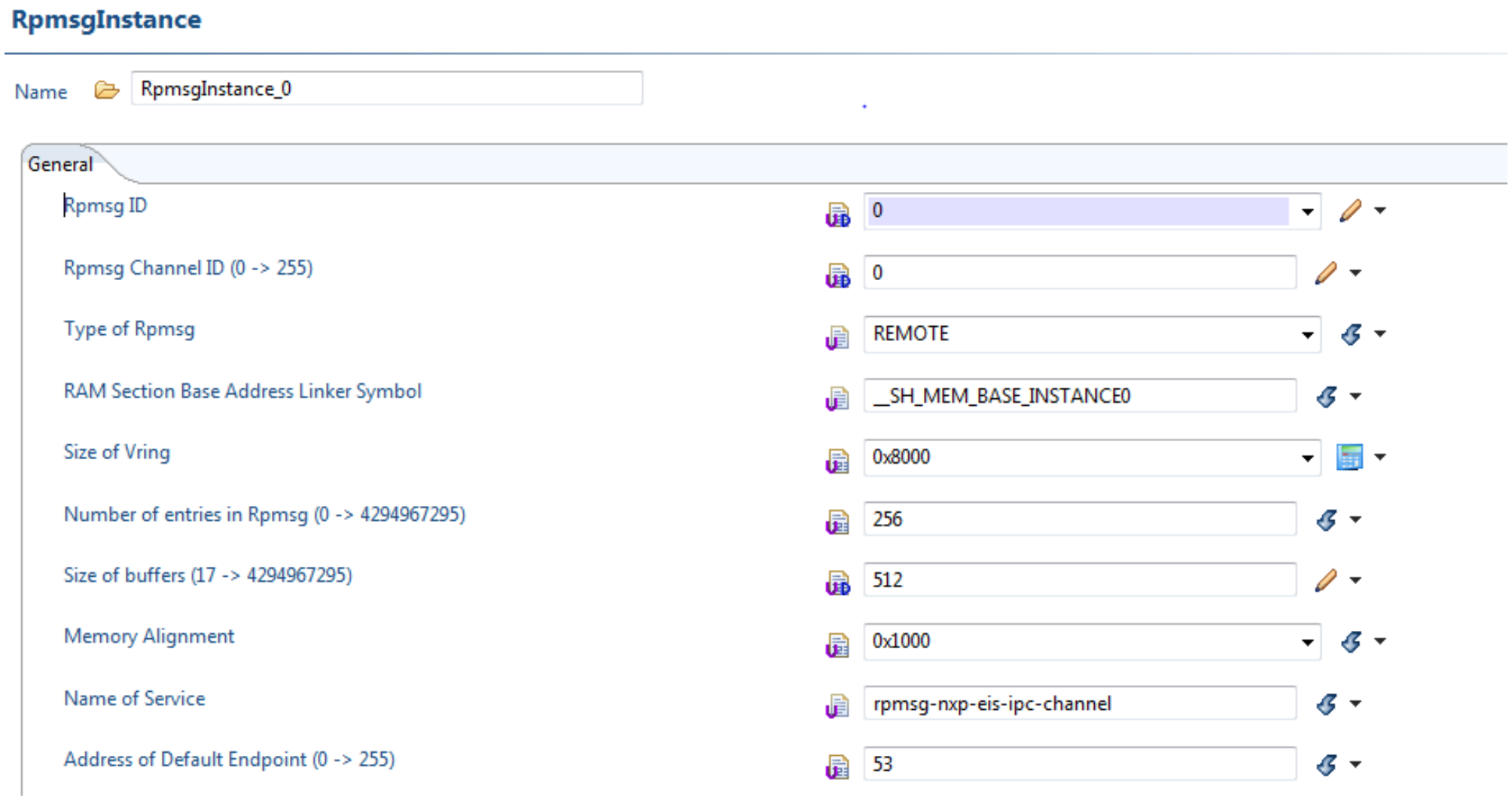


Figure 5-8. MU Block Diagram

RPMsg Configuration

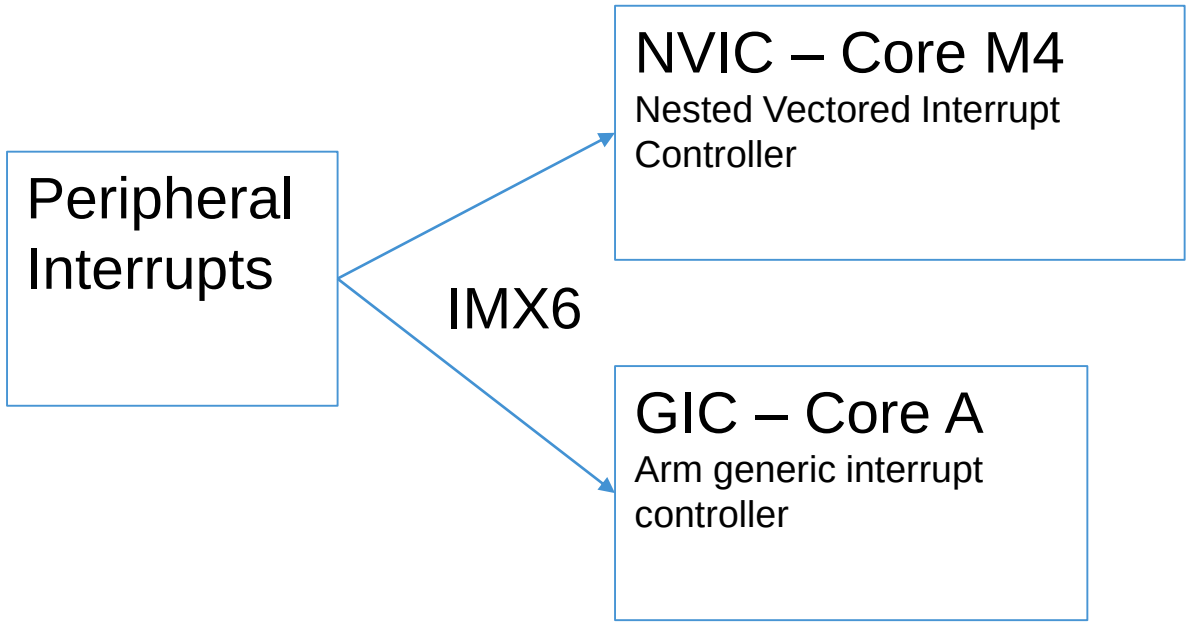


RPMSG Module Tips and Tricks

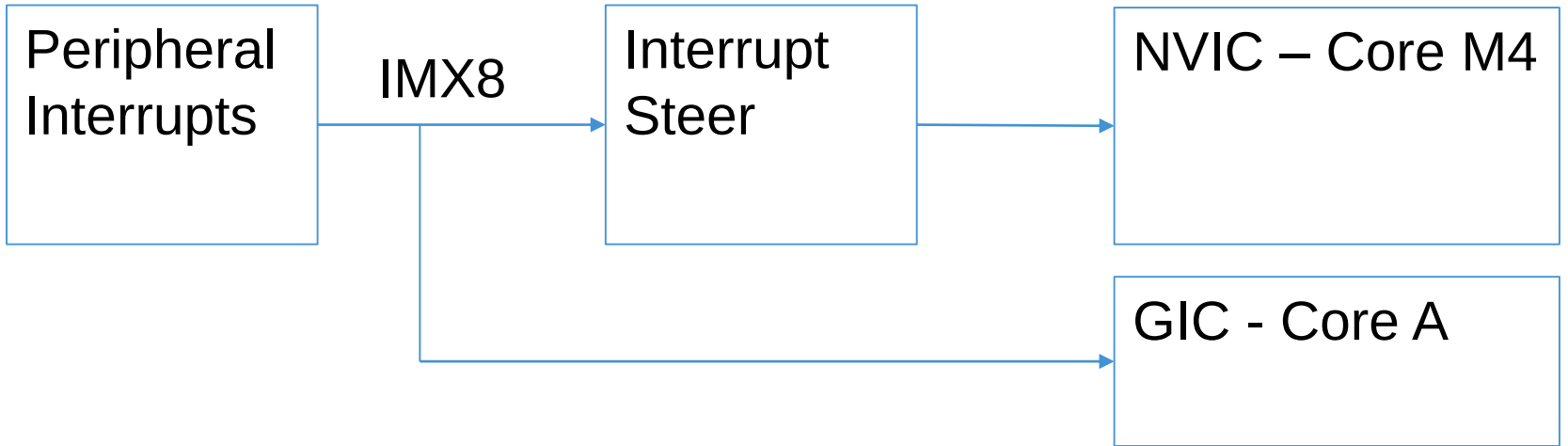
RPMsg Driver supports both MASTER and SLAVE mode

i.MX Interrupts Services

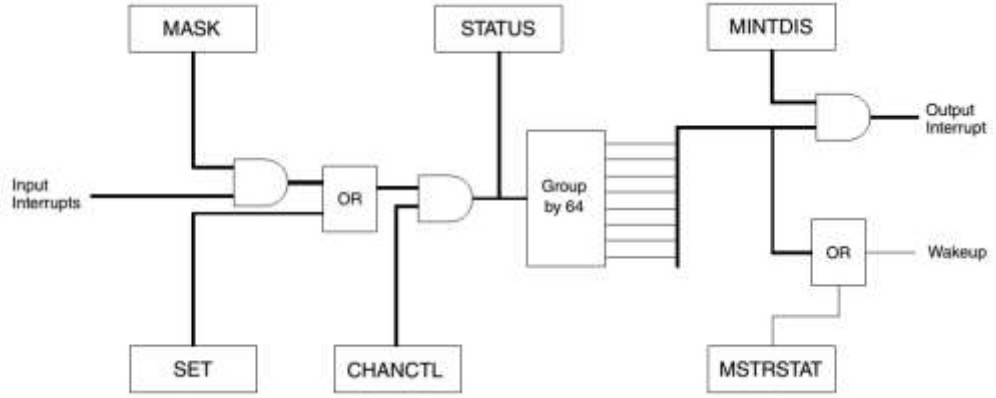
Interrupt Handler

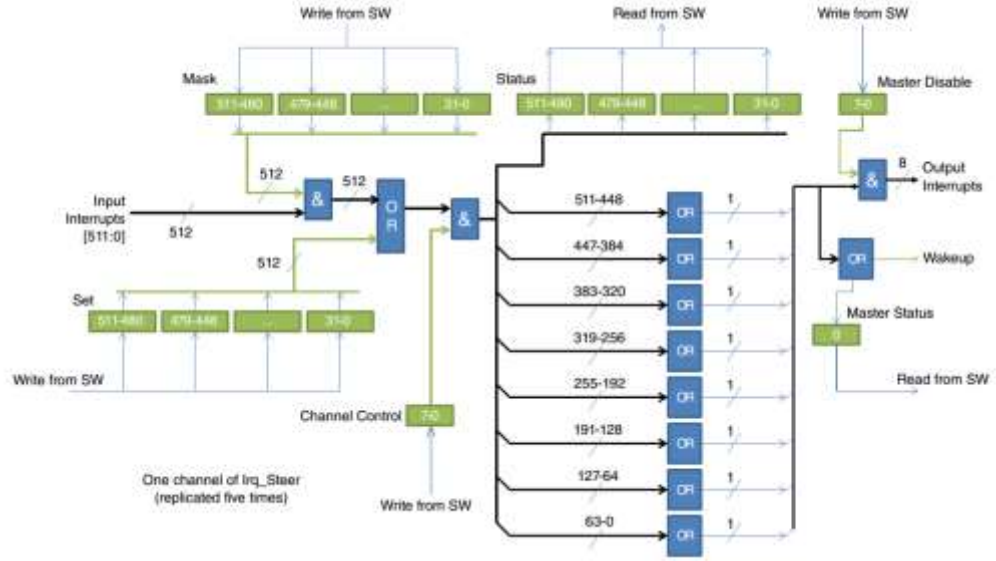


- Interrupt Steer: is redirects/steers the incoming interrupts to output interrupts.
- Interrupt steer is being controlled by MCL driver



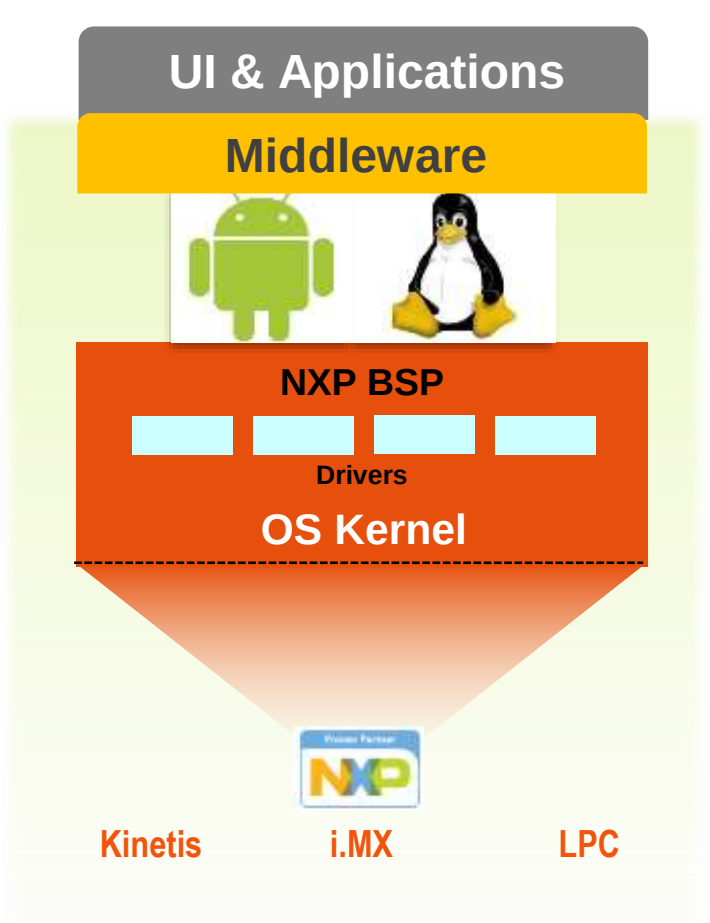
Interrupt Steer





The figure below is a detailed diagram of one 512-interrupt channel of the irq_steer module

Professional Engineering Services Technical Competency



Primary Focus Area

Customer Skill Set

Service Skill Set & Knowledge Value

Platform Provider, with Reach to the Middleware and Graphics Layers

SECURE CONNECTIONS
FOR A SMARTER WORLD

www.nxp.com