MBD Approach with MATLAB & SIMULINK in Powertrain Application Development Using NXP Microprocessor S32Kxx

Daniel Popa, Razvan Ionescu

NXP Automotive Microcontroller & Processors Tools

October 2019 | Session #AMF-AUT-T3836

SECURE CONNECTIONS FOR A SMARTER WORLD

Agenda

- MBD Development Process with MATLAB Ecosystem – Value Proposition
- NXP MBD Toolbox SW Development Flow
- NXP MBD Toolbox for S32K Features
  - S32K Core & Systems Support
  - S32K Motor Control Support
  - S32K Communication Support
  - Various S32K MCU Configuration Modes
  - Code Generation and Cross-compiler Support
  - Simulation Mode Support (Normal, SIL, PIL)
  - MBDT S32K Utilities: AMMCLIB, FreeMASTER, BootLoader, Profiler, Registers

Model-Based Design Development Process with MATLAB Ecosystem

Simulate, Test, Build and Deploy

Model-Based Design (MBD) Introduction

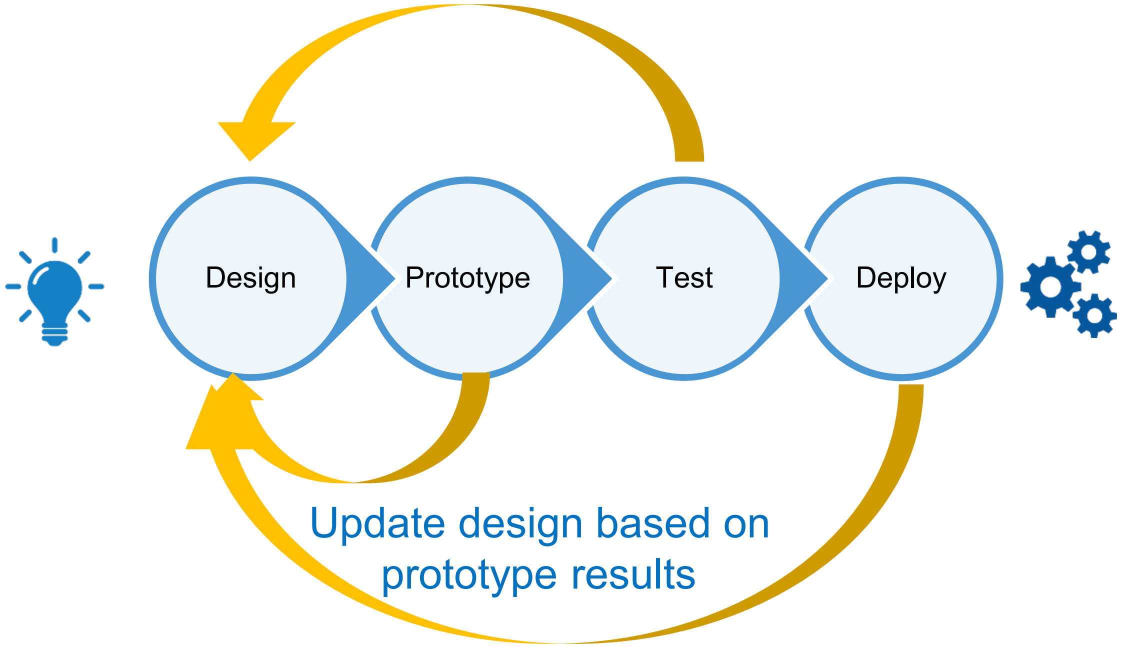
- MBD is becoming more common during the normal course of software development to explain and implement the desired behavior of a system. The challenge is to take advantage of this approach and get an executable that can be simulated and implemented directly from the model to help you get the product to market in less time and with higher quality. This is especially true for electric motor controls development in this age of hybrid/electric vehicles and the industrial motor control application space.

- Many companies model their controller algorithm and the target motor or plant so they can use a simulation environment to accelerate their algorithm development.

- The final stage of this type of development is the integration of the control algorithm software with target MCU hardware. This is often done using hand code or a mix of hand code and model-generated code. NXP’s Model Based Design Toolbox allows this stage of the development to generate 100% of the code from the model.

Traditional vs. Model-Based Design Dev. Process

Redefine the design based on test insights



Design

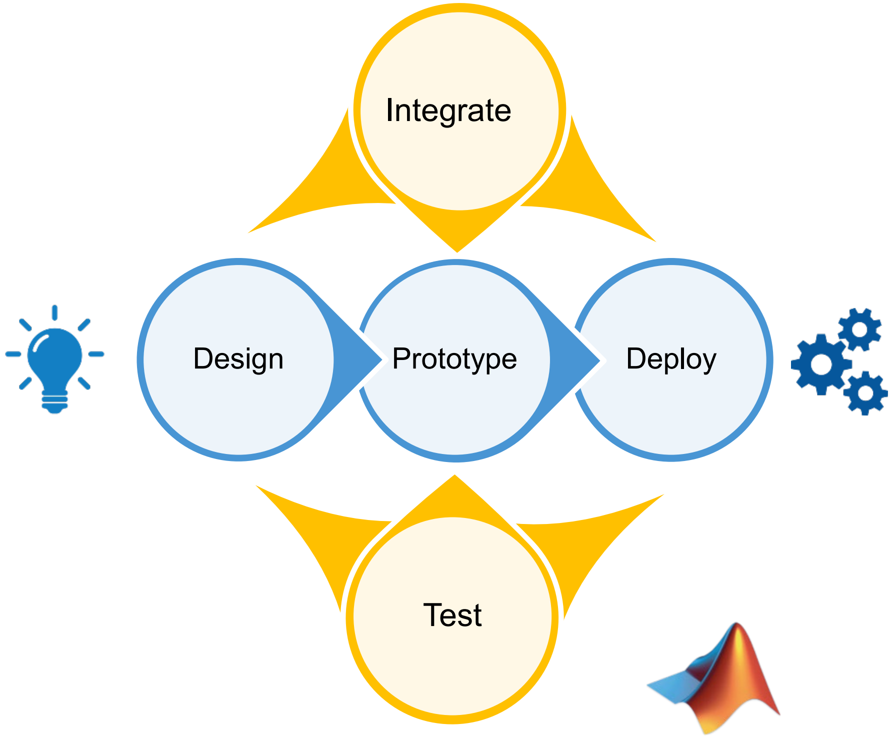
Prototype

Test

Deploy

Update design based on prototype results

Integrate tested existing code to new design



Integrate

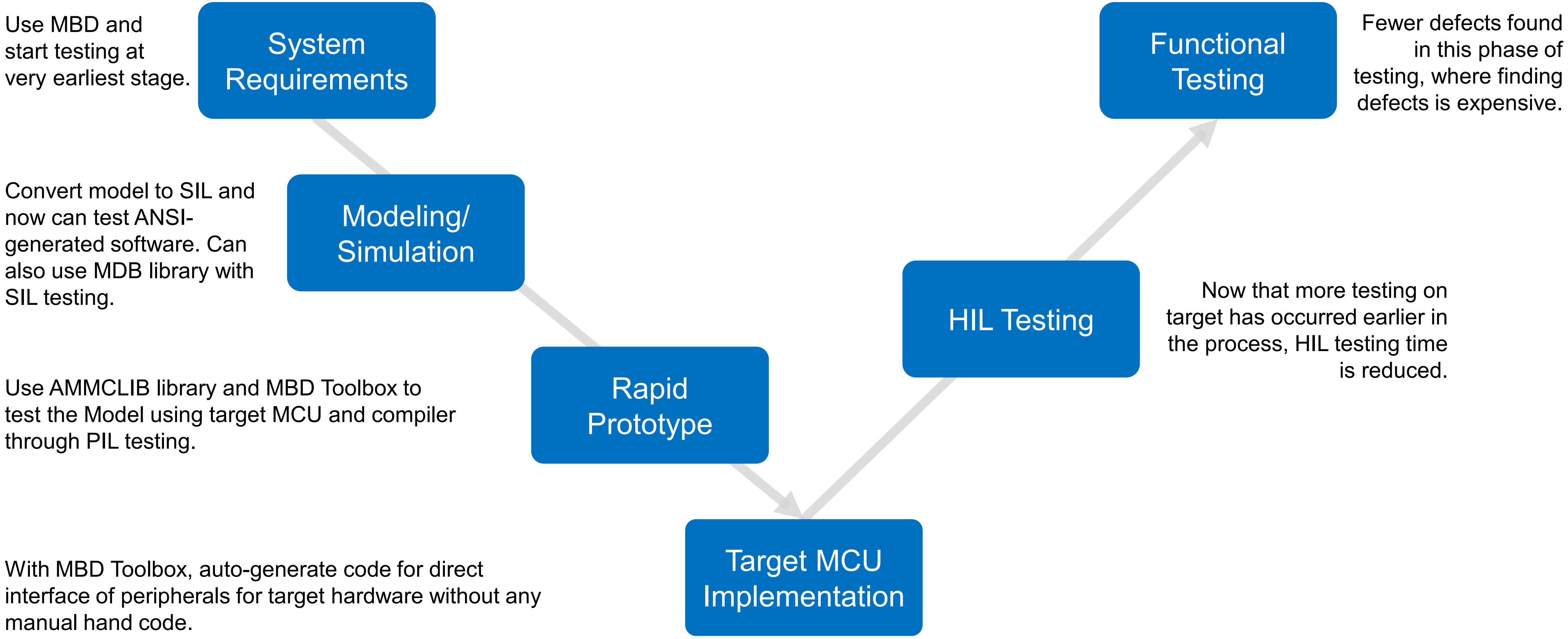
Design

Prototype

Deploy

Test

Reduce TTM With Model-Based Design (V-Model SW Dev)



System Requirements

Use MBD and start testing at very earliest stage.

Modeling/ Simulation

Convert model to SIL and now can test ANSI-generated software. Can also use MDB library with SIL testing.

Rapid Prototype

Use AMMCLIB library and MBD Toolbox to test the Model using target MCU and compiler through PIL testing.

Target MCU Implementation

With MBD Toolbox, auto-generate code for direct interface of peripherals for target hardware without any manual hand code.

HIL Testing

Now that more testing on target has occurred earlier in the process, HIL testing time is reduced.

Functional Testing

Fewer defects found in this phase of testing, where finding defects is expensive.

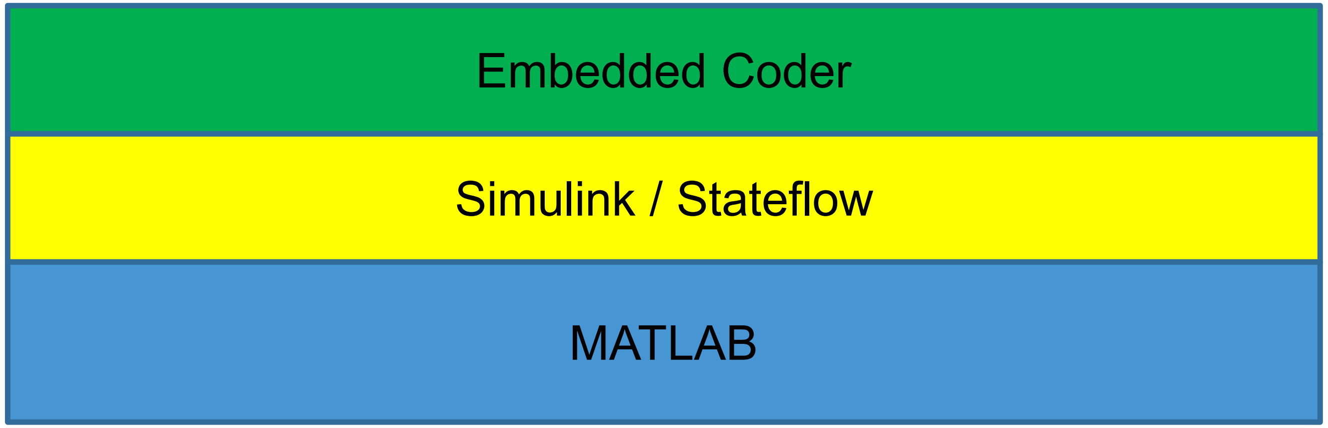
MATHWORKS Model-Based Design Environment

Embedded Coder is production code generation environment from MATLAB/Simulink models that generates ANSI C source code.

Simulink is graphical environment for building controls algorithms as well as simulation of these algorithms. Stateflow is a special case of Simulink blocks for state based design and flow chart controls of execution.

Simulink allows for a basic solver to execute either in discrete time or continuous time modes.

MATLAB is the base tool environment, very powerful and scriptable and allows access to complex functionalities



Embedded Coder

Simulink / Stateflow

MATLAB

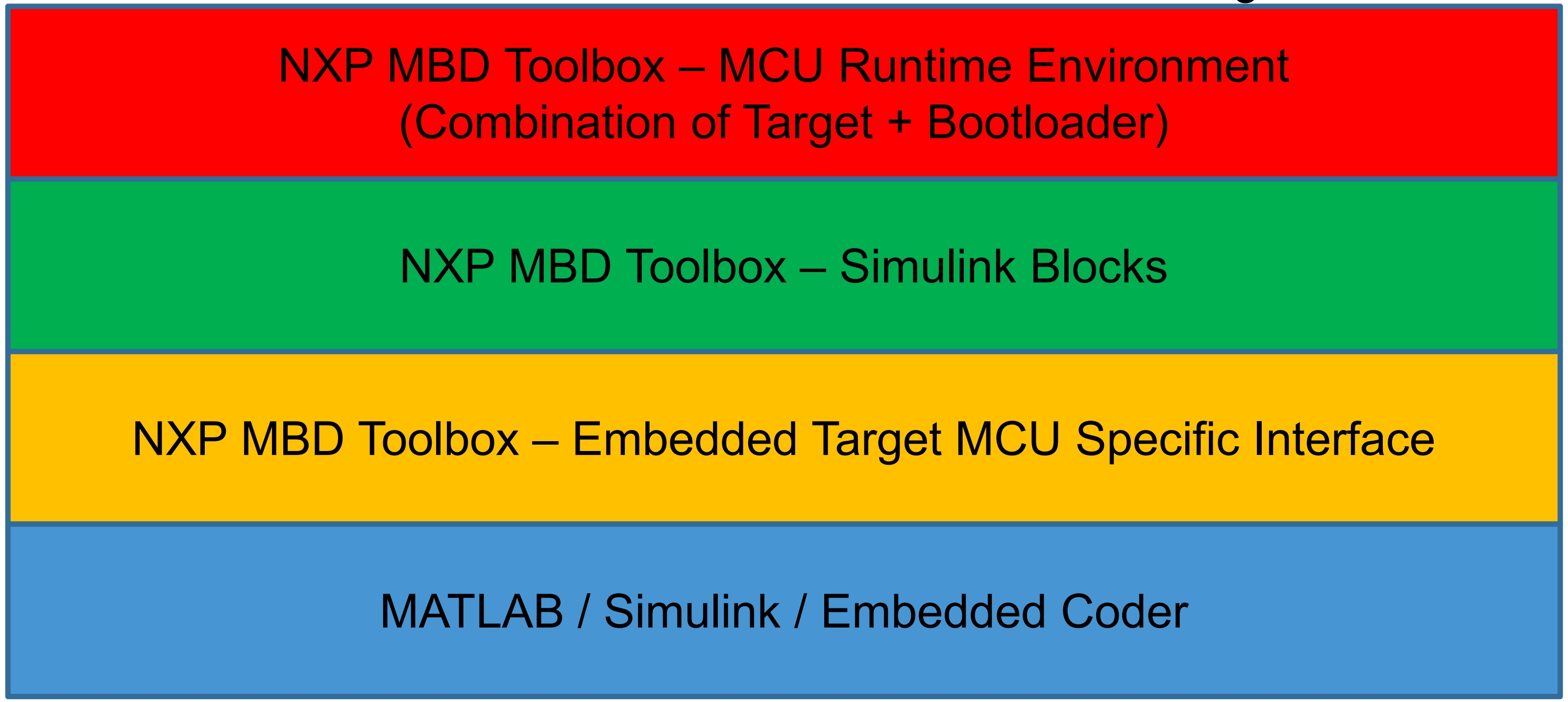
NXP Model-Based Design Toolbox Environment

MCU embedded target and bootloader support

NXP Simulink Blocks for algorithm components, place holders for peripheral drivers, and other functional utility block interfaces.

NXP Embedded Target, basic main functions + MCU specific infrastructure code and make file generation to build .elf/s-record/mot files on back end of code generation process.

Standard MathWorks Tools for Model-Based Design



NXP MBD Toolbox – MCU Runtime Environment
(Combination of Target + Bootloader)

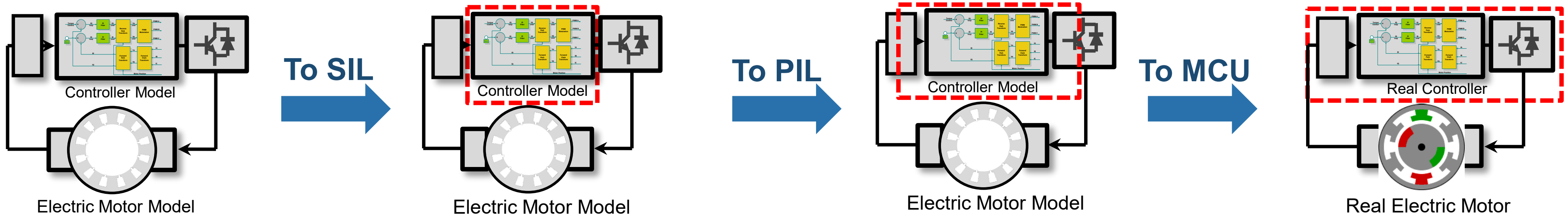
NXP MBD Toolbox – Simulink Blocks

NXP MBD Toolbox – Embedded Target MCU Specific Interface

MATLAB / Simulink / Embedded Coder

NXP Model-Based Design Steps

Idea incubation	Code Generation	Code Validation	Final Product



PC Environment

PC Environment

PC Environment + MCU

MCU with Embedded Control Module (ECM)

Step 1 – System Requirements:

Model-in-the-Loop

- Software requirements
- Control system requirements
- Overall application control strategy

Step 2 – Modeling/Simulation:

Software-in-the-Loop

- Control algorithm design
- Code generation preparation
- Control system design
- Start testing implementation approach

Step 3 – Rapid Prototype:

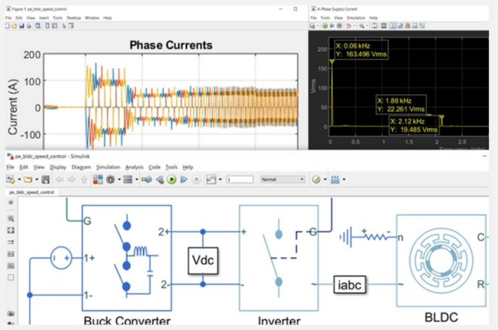
Processor-in-the-Loop

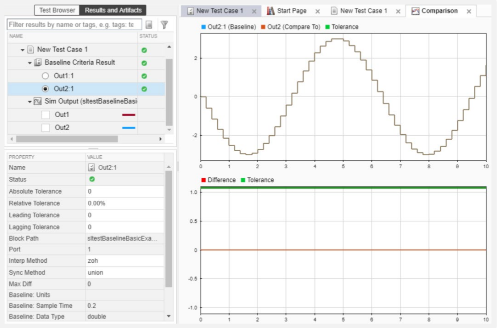
- Controller code generation
- Determine execution time on MCU
- Verify algorithm on MCU
- See memory/stack usage on MCU

Step 4 – Target MCU Implementation

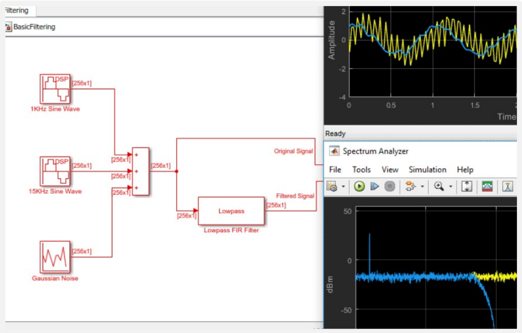
MCU Final Application

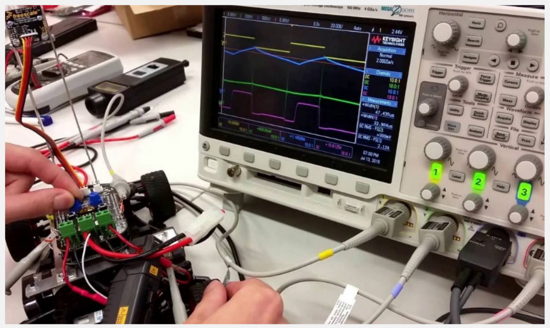
- Validation/verification phase
- Controller code generation
- Test system in target environment using tools for data logging and parameter tuning





Name: Out2:1
Status
Absolute Tolerance: 0
Relative Tolerance: 0.00%
Leading Tolerance: 0
Lagging Tolerance: 0
Block Path: sltestBaselineBasicExa...
Port: 1
Interp Method: zoh
Sync Method: union
Max Diff: 0
Baseline: Units
Baseline: Sample Time: 0.2
Baseline: Data Type: double



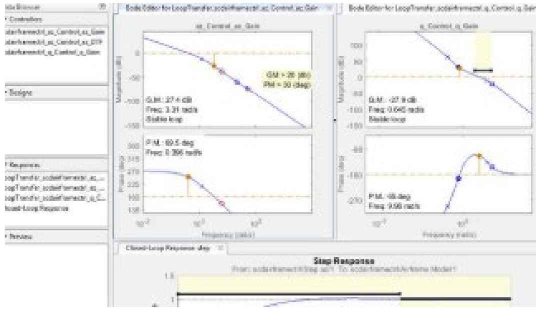


MathWorks Ecosystem

Control & Design

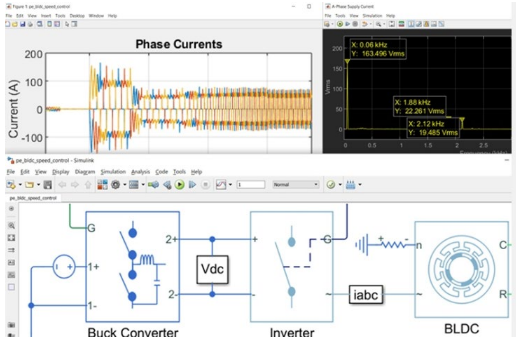
- Control Design

  Tuning & Auto tuning, Frequency Response



- Simscape

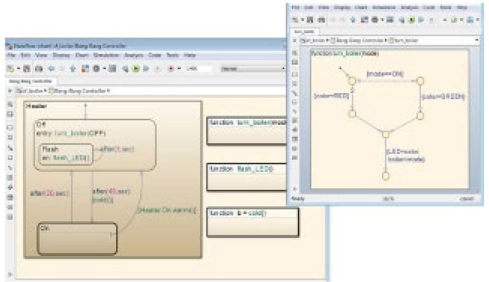
  Motors, IGBT, Diodes, Thermal, Spice

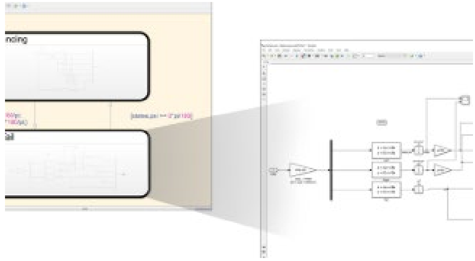


Event Based Modeling

- Stateflow

  Logic design, Scheduling

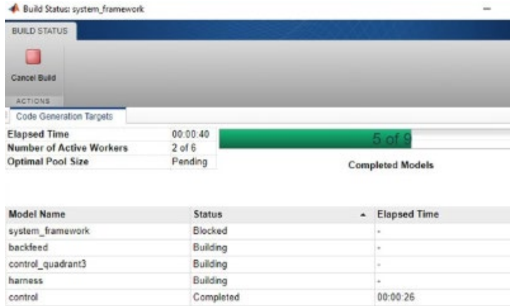




Code Generation

- Simulink Coder

  Code generation, legacy code integration, XCP, HIL, PIL, External mode



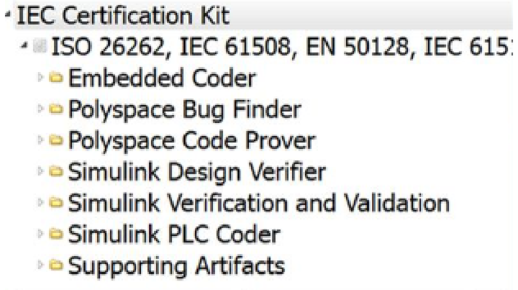
Elapsed Time	00:00:40
Number of Active Workers	2 of 6
Optimal Pool Size	Pending

Model Name	Status	Elapsed Time
system_framework	Blocked	-
backfeed	Building	-
control_quadrant3	Building	-
harness	Building	-
control	Completed	00:00:26

Certification & Validation

- IEC Certification Kit

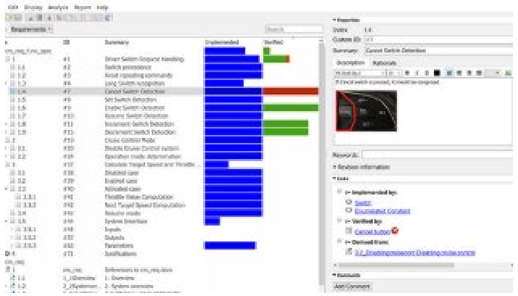
  IEC 61508, ISO26262, EN50128, IEC62304



- IEC Certification Kit
  - ISO 26262, IEC 61508, EN 50128, IEC 615
    - Embedded Coder
    - Polyspace Bug Finder
    - Polyspace Code Prover
    - Simulink Design Verifier
    - Simulink Verification and Validation
    - Simulink PLC Coder
    - Supporting Artifacts

- Simulink Requirements

  Requirements, Trace PRD, Track Status

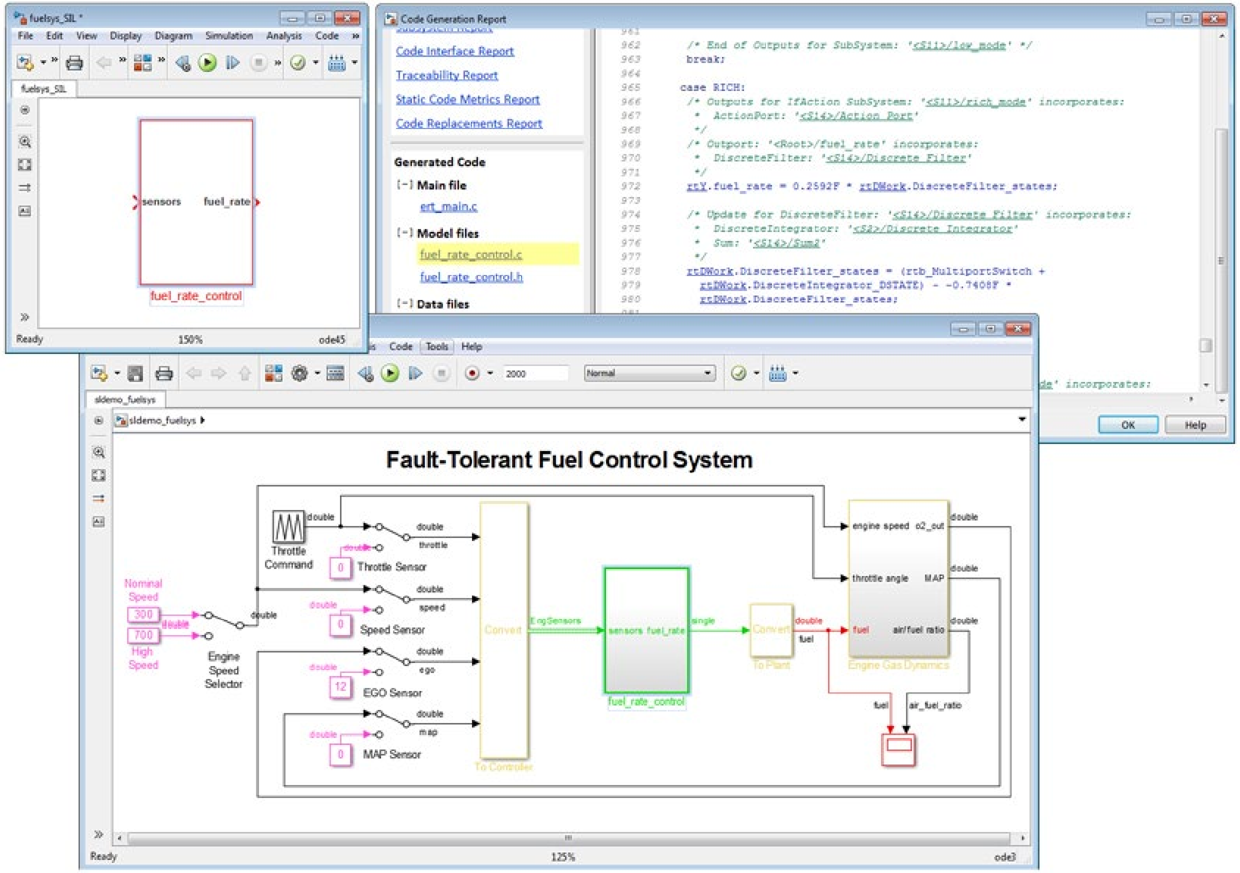


Embedded Coder

Automatic code generation

Embedded Coder Introduction

- A set of optimization and code configuration options that extend MATLAB Coder and Simulink Coder to generate code for a specific MCU.
- Storage class, type, and alias definition using Simulink® data dictionary capabilities.
- MCU specific code optimizations
- Code verification, including SIL and PIL testing, custom comments, and code reports with tracing of models to and from code and requirements
- Standards support, including ISO 26262, and MISRA-C



Code Generation Report

Code Interface Report

Traceability Report

Static Code Metrics Report

Code Replacements Report

Generated Code

[-] Main file

ert_main.c

[-] Model files

fuel_rate_control.c

fuel_rate_control.h

[-] Data files

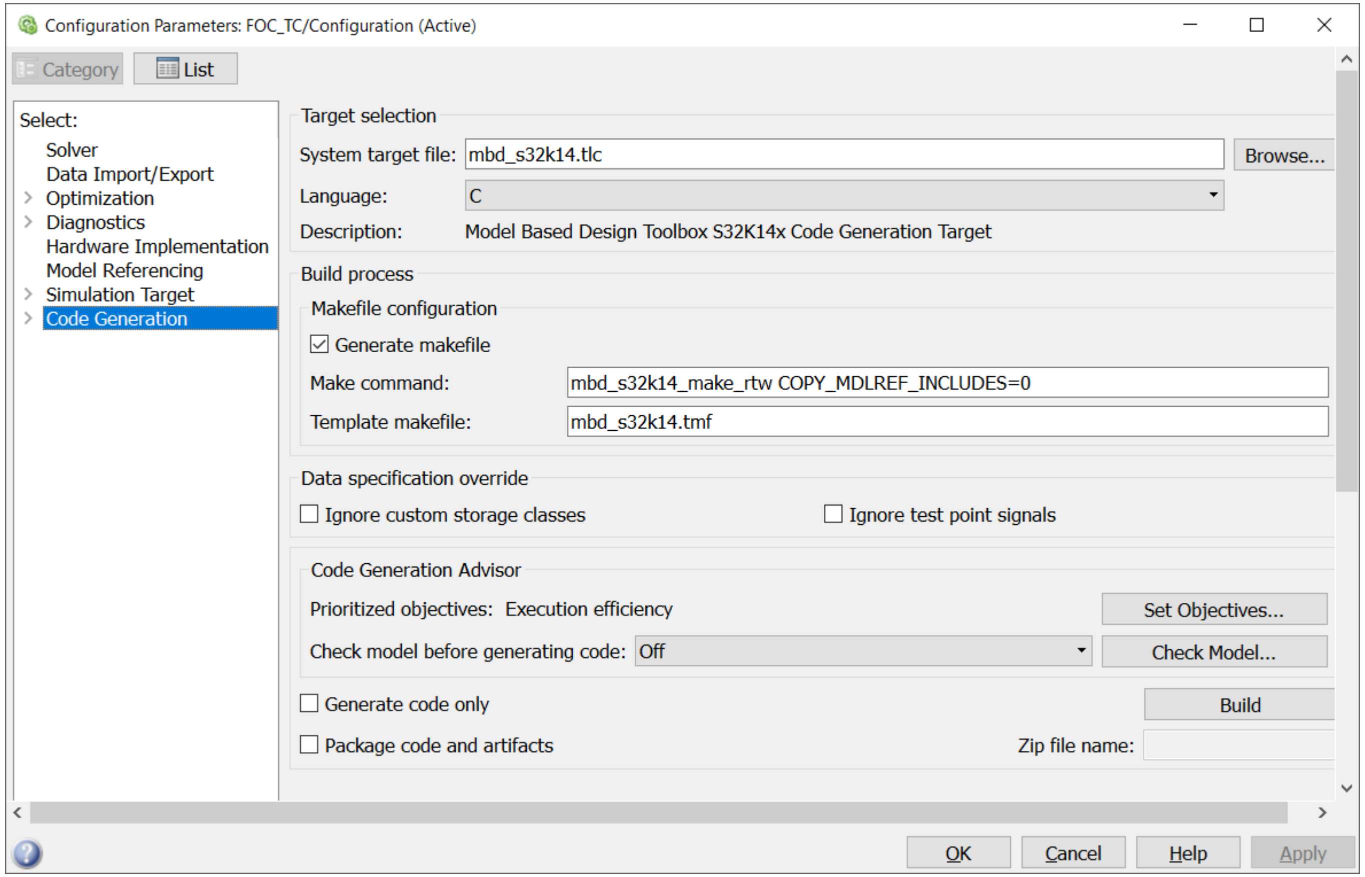
```
  /* End of Outputs for SubSystem: '<S11>/low_mode' */
  break;

 case RICH:
  /* Outputs for IfAction SubSystem: '<S11>/rich_mode' incorporates:
   *  ActionPort: '<S14>/Action Port'
   */
  /* Outport: '<Root>/fuel_rate' incorporates:
   *  DiscreteFilter: '<S14>/Discrete Filter'
   */
  rtY.fuel_rate = 0.2592F * rtDWork.DiscreteFilter_states;

  /* Update for DiscreteFilter: '<S14>/Discrete Filter' incorporates:
   *  DiscreteIntegrator: '<S2>/Discrete Integrator'
   *  Sum: '<S14>/Sum2'
   */
  rtDWork.DiscreteFilter_states = (rtb_MultiportSwitch +
    rtDWork.DiscreteIntegrator_DSTATE) - -0.7408F *
    rtDWork.DiscreteFilter_states;
```

Embedded Target Code Generation Requirements

- Simulink Model must use a discrete time solver
- Must NOT contain any continuous/special blocks
- Code Generation Add-on: To build MCU target executable must have embedded target selected (NXP MBD Toolbox provides all Embedded Coder configuration files for the S32K MCUs).
- May generate code only or generate code and compile for a target environment .exe, .dll, .s19, .elf…).



Target selection

System target file: mbd_s32k14.tlc

Language: C

Description: Model Based Design Toolbox S32K14x Code Generation Target

Build process

Makefile configuration

☑ Generate makefile

Make command: mbd_s32k14_make_rtw COPY_MDLREF_INCLUDES=0

Template makefile: mbd_s32k14.tmf

Data specification override

□ Ignore custom storage classes □ Ignore test point signals

Code Generation Advisor

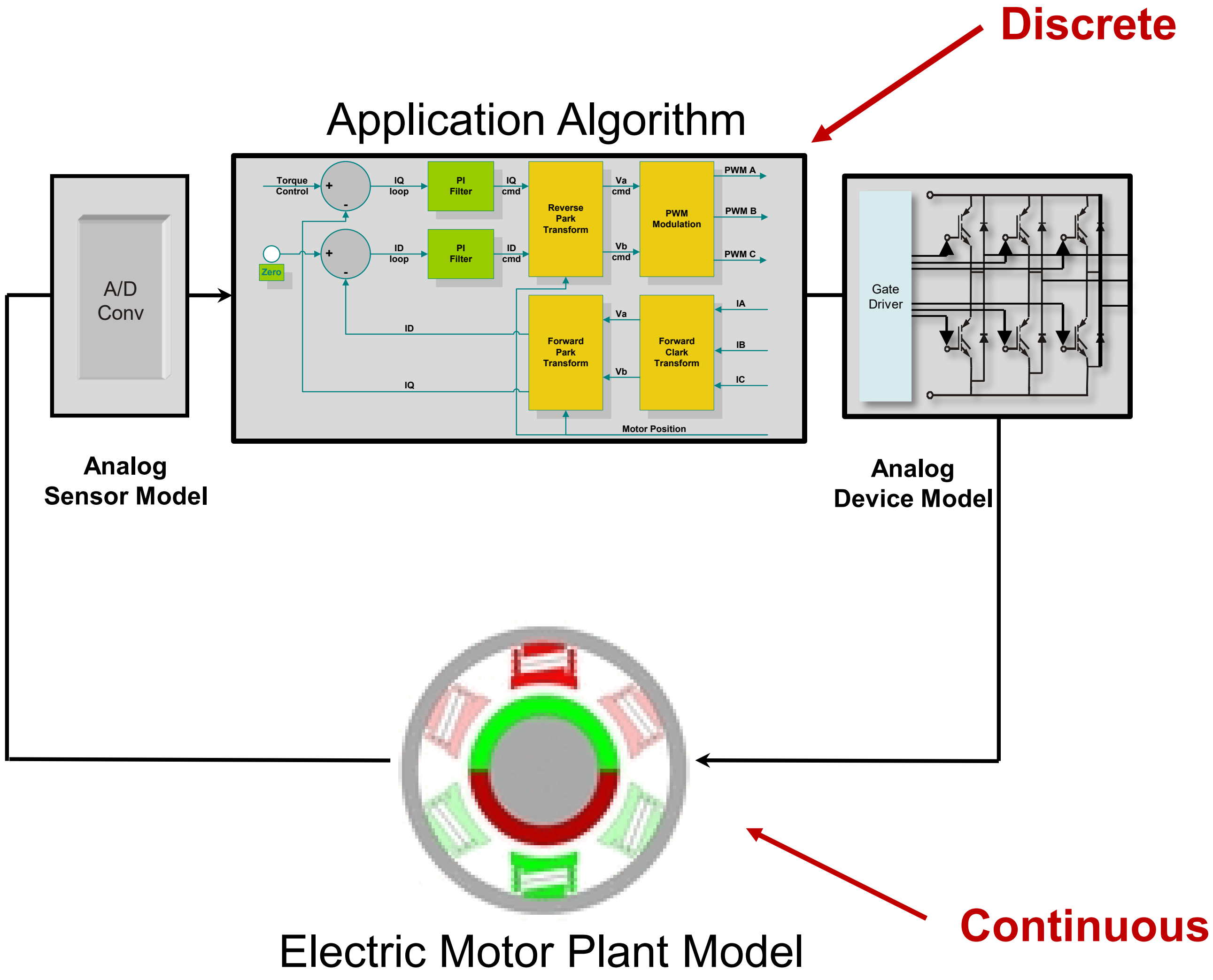
Prioritized objectives: Execution efficiency

Check model before generating code: Off

□ Generate code only

□ Package code and artifacts Zip file name:

Control System Anatomy



Discrete

Application Algorithm

Analog Sensor Model

Analog Device Model

Electric Motor Plant Model

Continuous

Embedded Target Requirements for Application Algorithm

- Start from Idealized model (double precision unconstrained mathematics).
- Need to move to target constrained model (integer code, fixed point mathematics, or single precision mathematics).
- Tradeoffs are made in this process depending on how the user chooses to implement the controller on the target MCU.
- The simulation environment supports simulation in any of the target data types also code generation is supported when targeting the MCU.
- What is executed in simulation should be exactly the same as on the embedded target.

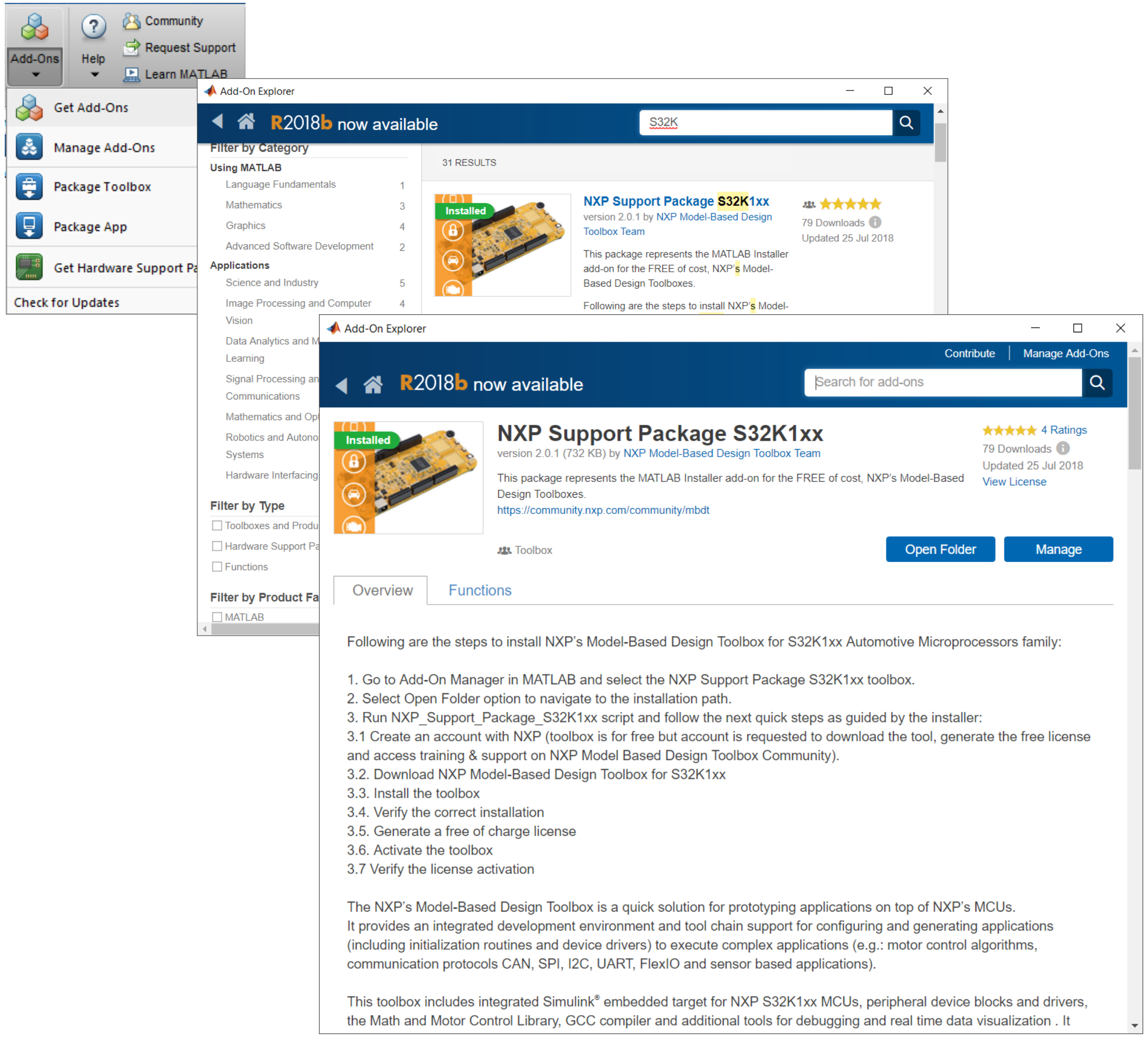
Embedded Target Code Generation Considerations

- Must have modeling style guidelines that takes into account code generation and target software architecture
- Plan code generation to meet target software architecture
- Use of an interface Data Dictionary to minimize software integration issues is industry best practice
- Design/Refine models for code generation to target MCU
  - Learn and understand code generator optimizations (Know the Tools)
  - Optimize model for code generation with target MCU in mind
    - Data Types
    - Optimize Function/File Partitions
    - Utilize target optimized functions for key bottlenecks (custom)
    - Utilize target optimized block sets whenever possible

NXP MBD Toolbox for S32K Development Flow

Quick Introduction

NXP Model-Based Design Toolbox – Installation



NXP Support Package S32K1xx

version 2.0.1 by NXP Model-Based Design Toolbox Team

79 Downloads
Updated 25 Jul 2018

This package represents the MATLAB Installer add-on for the FREE of cost, NXP's Model-Based Design Toolboxes.

Following are the steps to install NXP's Model-

NXP Support Package S32K1xx

version 2.0.1 (732 KB) by NXP Model-Based Design Toolbox Team

4 Ratings
79 Downloads
Updated 25 Jul 2018
View License

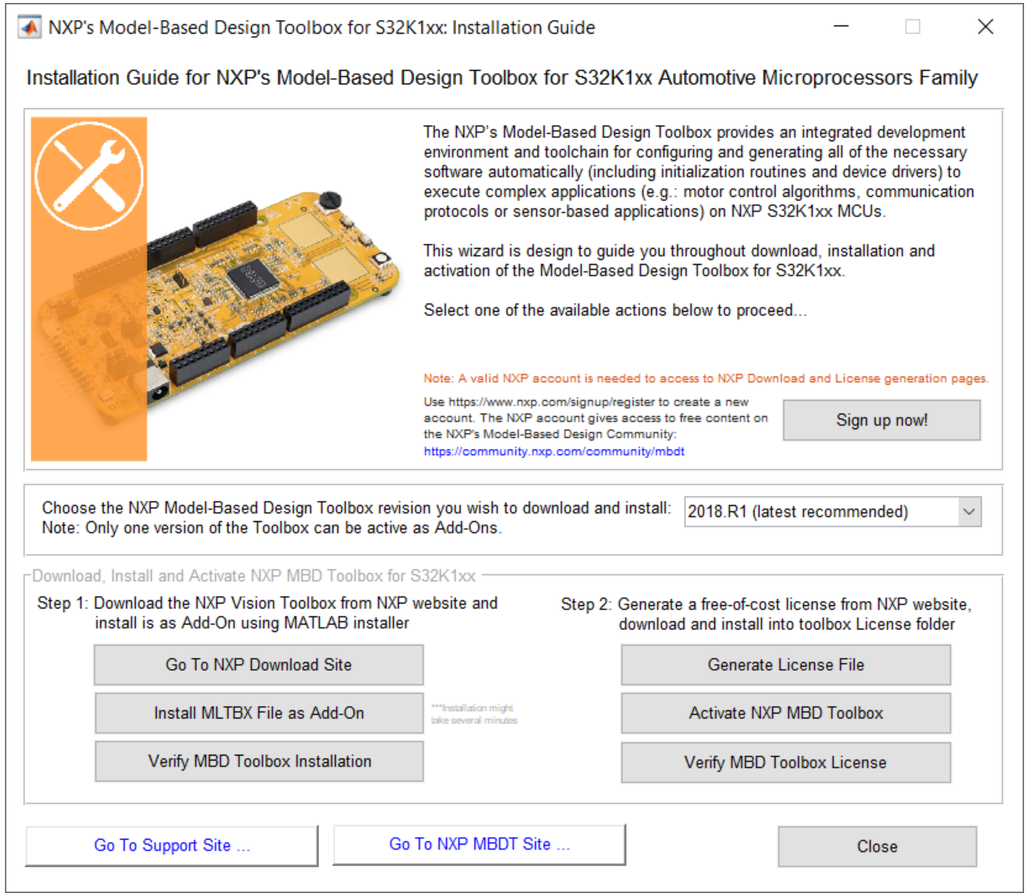
This package represents the MATLAB Installer add-on for the FREE of cost, NXP's Model-Based Design Toolboxes.
https://community.nxp.com/community/mbdt

Following are the steps to install NXP's Model-Based Design Toolbox for S32K1xx Automotive Microprocessors family:

1. Go to Add-On Manager in MATLAB and select the NXP Support Package S32K1xx toolbox.
2. Select Open Folder option to navigate to the installation path.
3. Run NXP_Support_Package_S32K1xx script and follow the next quick steps as guided by the installer:
3.1 Create an account with NXP (toolbox is for free but account is requested to download the tool, generate the free license and access training & support on NXP Model Based Design Toolbox Community).
3.2. Download NXP Model-Based Design Toolbox for S32K1xx
3.3. Install the toolbox
3.4. Verify the correct installation
3.5. Generate a free of charge license
3.6. Activate the toolbox
3.7 Verify the license activation

The NXP's Model-Based Design Toolbox is a quick solution for prototyping applications on top of NXP's MCUs.
It provides an integrated development environment and tool chain support for configuring and generating applications (including initialization routines and device drivers) to execute complex applications (e.g.: motor control algorithms, communication protocols CAN, SPI, I2C, UART, FlexIO and sensor based applications).

This toolbox includes integrated Simulink® embedded target for NXP S32K1xx MCUs, peripheral device blocks and drivers, the Math and Motor Control Library, GCC compiler and additional tools for debugging and real time data visualization . It


NXP's Model-Based Design Toolbox for S32K1xx: Installation Guide
Sign up now!
Go To NXP Download Site
Install MLTBX File as Add-On
**Installation might take several minutes
Verify MBD Toolbox Installation
Generate License File
Activate NXP MBD Toolbox
Verify MBD Toolbox License
Go To Support Site ...
Go To NXP MBDT Site ...
Close

Installation Guide for NXP's Model-Based Design Toolbox for S32K1xx Automotive Microprocessors Family

The NXP's Model-Based Design Toolbox provides an integrated development environment and toolchain for configuring and generating all of the necessary software automatically (including initialization routines and device drivers) to execute complex applications (e.g.: motor control algorithms, communication protocols or sensor-based applications) on NXP S32K1xx MCUs.

This wizard is design to guide you throughout download, installation and activation of the Model-Based Design Toolbox for S32K1xx.

Select one of the available actions below to proceed...

Note: A valid NXP account is needed to access to NXP Download and License generation pages.

Use https://www.nxp.com/signup/register to create a new account. The NXP account gives access to free content on the NXP's Model-Based Design Community:
https://community.nxp.com/community/mbdt

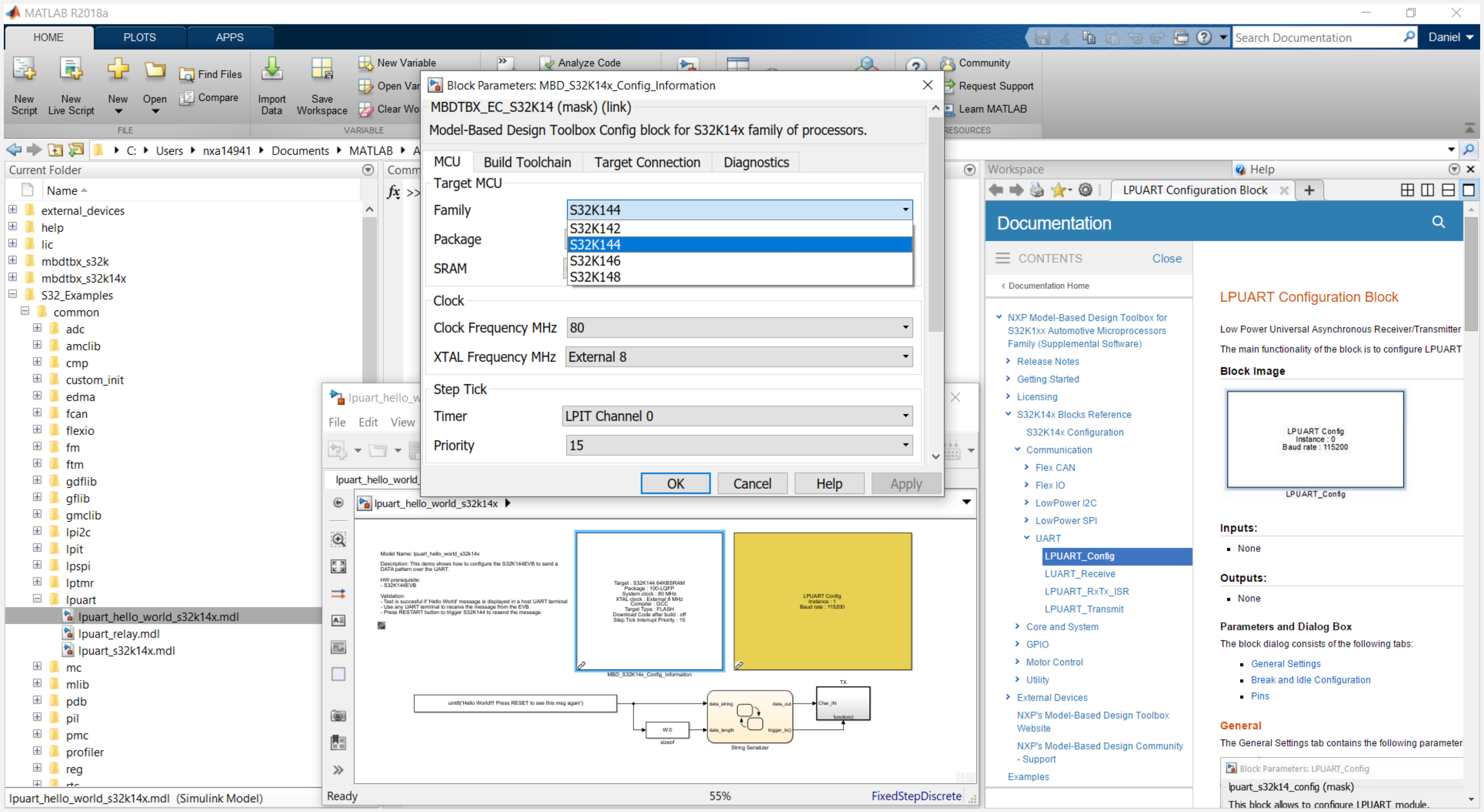
Choose the NXP Model-Based Design Toolbox revision you wish to download and install: 2018.R1 (latest recommended)
Note: Only one version of the Toolbox can be active as Add-Ons.

Download, Install and Activate NXP MBD Toolbox for S32K1xx

Step 1: Download the NXP Vision Toolbox from NXP website and install is as Add-On using MATLAB installer

Step 2: Generate a free-of-cost license from NXP website, download and install into toolbox License folder

NXP Model-Based Design Toolbox – Preview



Block Parameters: MBD_S32K14x_Config_Information

MBDTBX_EC_S32K14 (mask) (link)

Model-Based Design Toolbox Config block for S32K14x family of processors.

MCU | Build Toolchain | Target Connection | Diagnostics

Target MCU

Family: S32K144 (S32K142, S32K144, S32K146, S32K148)

Package

SRAM

Clock

Clock Frequency MHz: 80

XTAL Frequency MHz: External 8

Step Tick

Timer: LPIT Channel 0

Priority: 15

OK | Cancel | Help | Apply

lpuart_hello_world_s32k14x

Model Name: lpuart_hello_world_s32k14x

Description: This demo shows how to configure the S32K144EVB to send a DATA pattern over the UART.

HW prerequisite:
- S32K144EVB

Validation:
- Test is succesful if 'Hello World' message is displayed in a host UART terminal
- Use any UART terminal to receive the message from the EVB.
- Press RESTART button to trigger S32K144 to resend the message.

Target : S32K144 64KBSRAM
Package : 100-LQFP
System clock : 80 MHz
XTAL clock : External 8 MHz
Compiler : GCC
Target Type : FLASH
Download Code after build : off
Step Tick Interrupt Priority : 15

MBD_S32K14x_Config_Information

LPUART Config
Instance : 1
Baud rate : 115200

uint8('Hello World!!! Press RESET to see this msg again')

Documentation

CONTENTS Close

‹ Documentation Home

- NXP Model-Based Design Toolbox for S32K1xx Automotive Microprocessors Family (Supplemental Software)
  - Release Notes
  - Getting Started
  - Licensing
  - S32K14x Blocks Reference
    - S32K14x Configuration
    - Communication
      - Flex CAN
      - Flex IO
      - LowPower I2C
      - LowPower SPI
      - UART
        - LPUART_Config
        - LUART_Receive
        - LPUART_RxTx_ISR
        - LPUART_Transmit
    - Core and System
    - GPIO
    - Motor Control
    - Utility
  - External Devices
  - NXP's Model-Based Design Toolbox Website
  - NXP's Model-Based Design Community - Support
- Examples

LPUART Configuration Block

Low Power Universal Asynchronous Receiver/Transmitter

The main functionality of the block is to configure LPUART

Block Image

LPUART Config
Instance : 0
Baud rate : 115200

LPUART_Config

Inputs:

- None

Outputs:

- None

Parameters and Dialog Box

The block dialog consists of the following tabs:

- General Settings
- Break and Idle Configuration
- Pins

General

The General Settings tab contains the following parameter

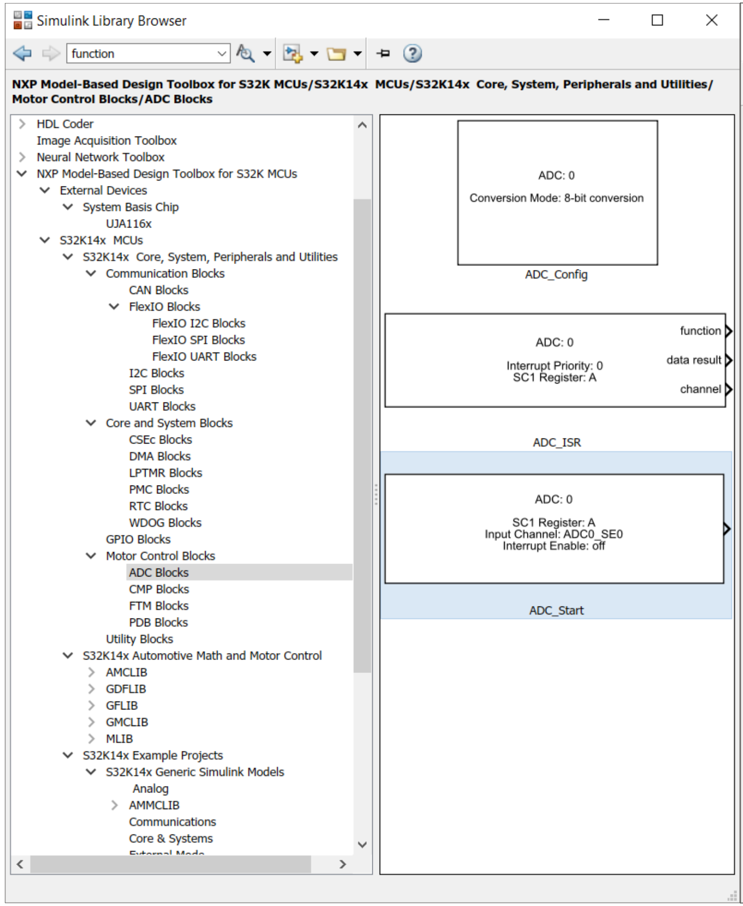
Block Parameters: LPUART_Config

lpuart_s32k14_config (mask)

This block allows to configure LPUART module.

S32K MBDT – SW Development Flow

Step 1: Open Simulink Library



NXP Model-Based Design Toolbox for S32K MCUs/S32K14x MCUs/S32K14x Core, System, Peripherals and Utilities/ Motor Control Blocks/ADC Blocks

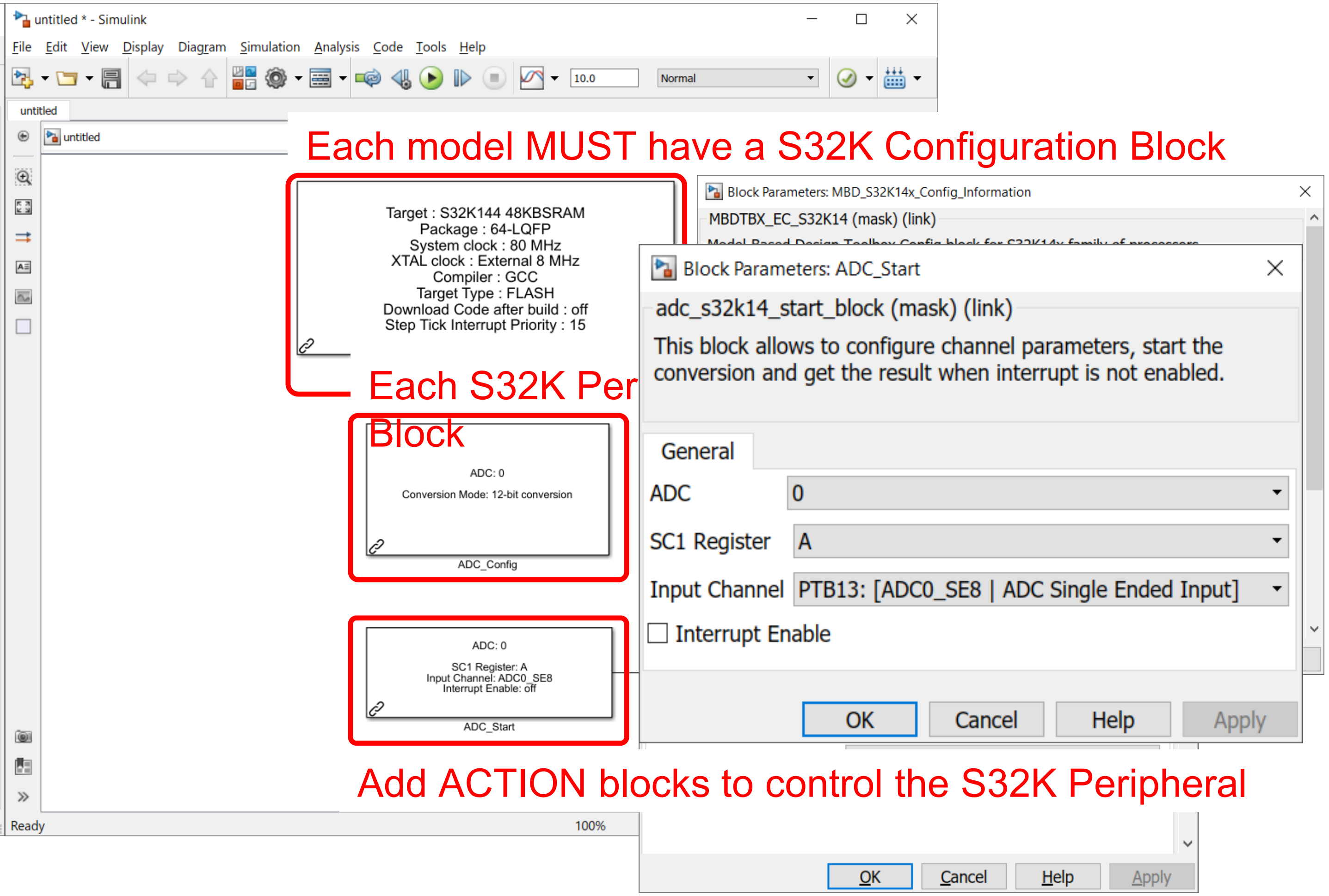
- HDL Coder
- Image Acquisition Toolbox
- Neural Network Toolbox
- NXP Model-Based Design Toolbox for S32K MCUs
  - External Devices
    - System Basis Chip
      - UJA116x
  - S32K14x MCUs
    - S32K14x Core, System, Peripherals and Utilities
      - Communication Blocks
        - CAN Blocks
        - FlexIO Blocks
          - FlexIO I2C Blocks
          - FlexIO SPI Blocks
          - FlexIO UART Blocks
        - I2C Blocks
        - SPI Blocks
        - UART Blocks
      - Core and System Blocks
        - CSEc Blocks
        - DMA Blocks
        - LPTMR Blocks
        - PMC Blocks
        - RTC Blocks
        - WDOG Blocks
      - GPIO Blocks
      - Motor Control Blocks
        - ADC Blocks
        - CMP Blocks
        - FTM Blocks
        - PDB Blocks
      - Utility Blocks
    - S32K14x Automotive Math and Motor Control
      - AMCLIB
      - GDFLIB
      - GFLIB
      - GMCLIB
      - MLIB
    - S32K14x Example Projects
      - S32K14x Generic Simulink Models
        - Analog
        - AMMCLIB
        - Communications
        - Core & Systems

ADC_Config: ADC: 0, Conversion Mode: 8-bit conversion

ADC_ISR: ADC: 0, Interrupt Priority: 0, SC1 Register: A

ADC_Start: ADC: 0, SC1 Register: A, Input Channel: ADC0_SE0, Interrupt Enable: off

Step 2: Open a new Simulink Canvas. Drag&Drop Blocks


untitled * - Simulink
File Edit View Display Diagram Simulation Analysis Code Tools Help
10.0
Normal
untitled
Ready
100%

Each model MUST have a S32K Configuration Block

Target : S32K144 48KBSRAM
Package : 64-LQFP
System clock : 80 MHz
XTAL clock : External 8 MHz
Compiler : GCC
Target Type : FLASH
Download Code after build : off
Step Tick Interrupt Priority : 15

Block Parameters: MBD_S32K14x_Config_Information

MBDTBX_EC_S32K14 (mask) (link)

Each S32K Per Block

ADC_Config: ADC: 0, Conversion Mode: 12-bit conversion

ADC_Start: ADC: 0, SC1 Register: A, Input Channel: ADC0_SE8, Interrupt Enable: off

Block Parameters: ADC_Start

adc_s32k14_start_block (mask) (link)

This block allows to configure channel parameters, start the conversion and get the result when interrupt is not enabled.

General

ADC: 0

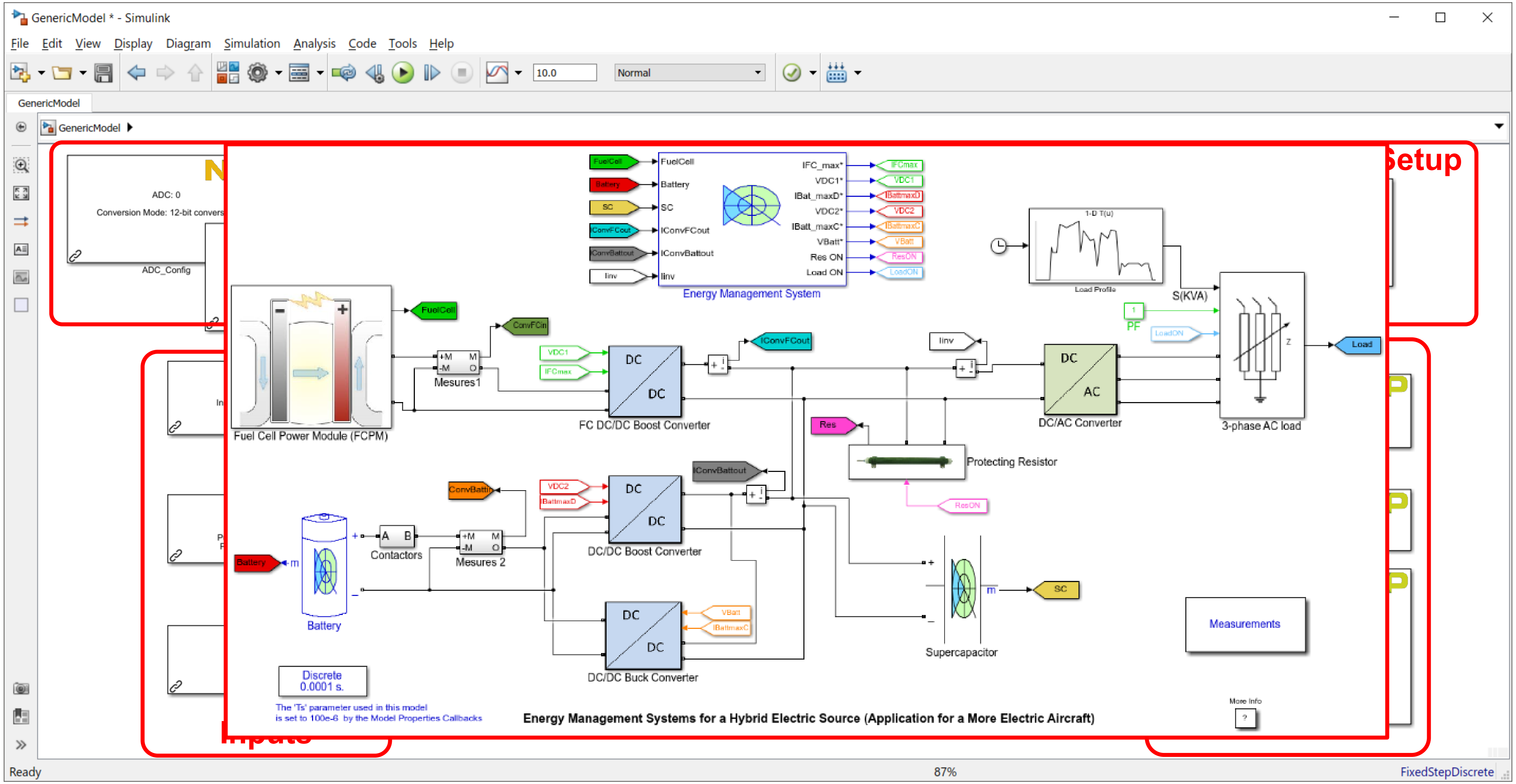
SC1 Register: A

Input Channel: PTB13: [ADC0_SE8 | ADC Single Ended Input]

☐ Interrupt Enable

Add ACTION blocks to control the S32K Peripheral

S32K MBDT – Limitless Design Options

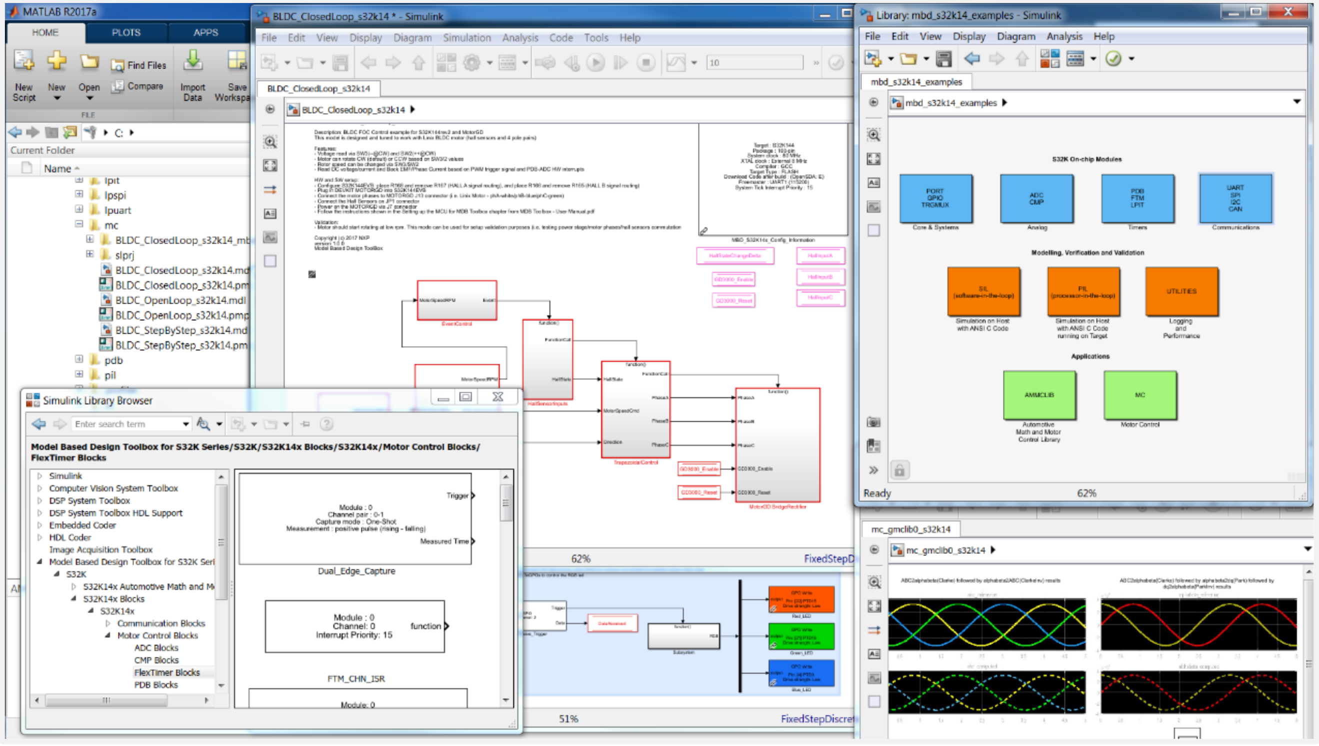


NXP MBD Toolbox for S32K Features

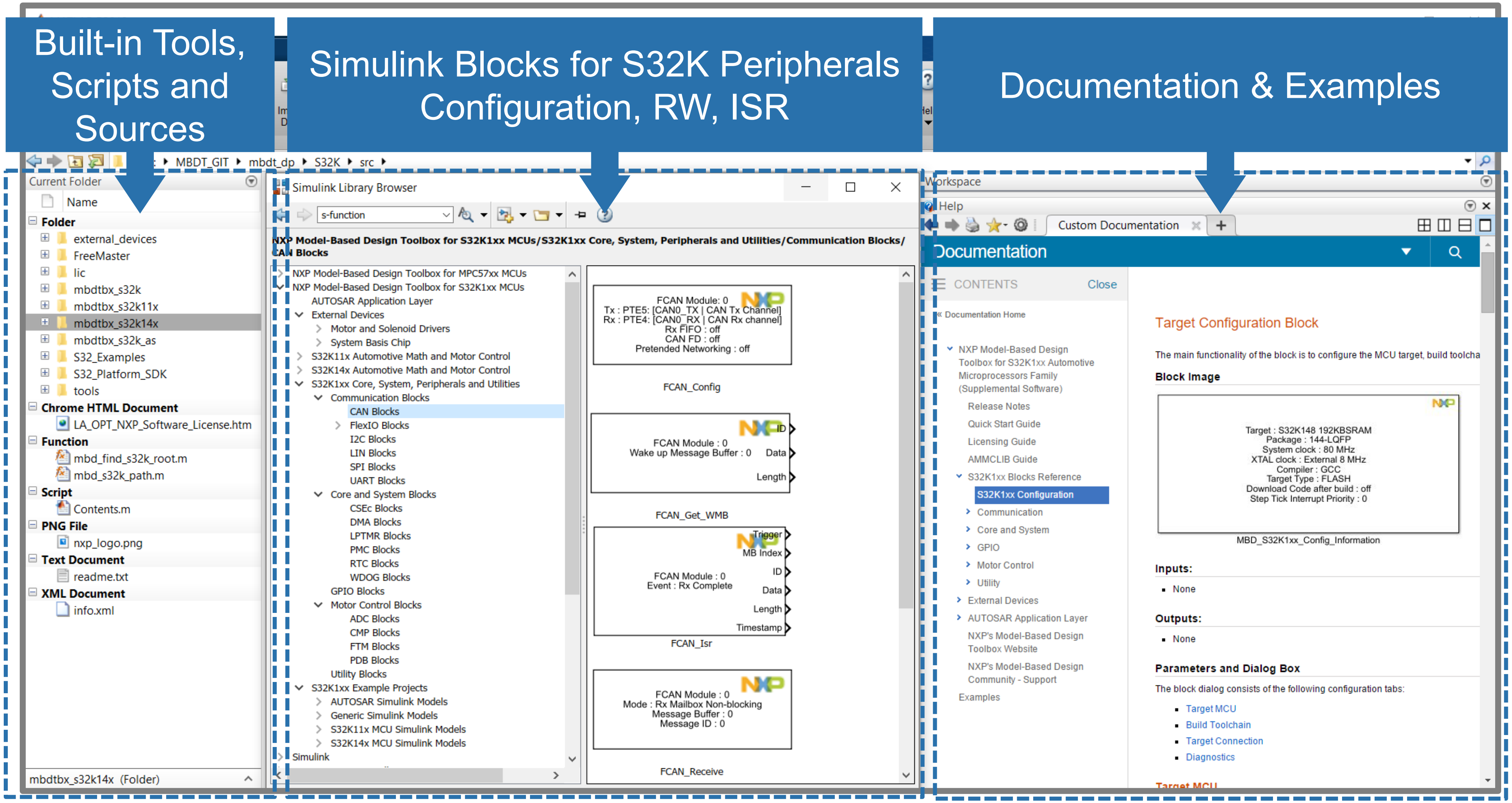
One common frameworks for simulation, testing and deploying embedded applications for S32K processors without any hand-written code

NXP Model-Based Design Toolbox – Overview

- Collection of Tools & Libraries designed to **Assist** customers with prototyping and accelerate algorithm development on NXP MCUs
- MCU Peripherals **Initialization & Configuration** through GUI from a Model-Based Design environment like Simulink®
- Supported **Platforms** for automatic Code Generation: MagniV S12ZVMx/S12ZVC, MPC56xx, MPC57xx, Arm®-based S32K, DSC and Kinetis families
- Customer **Support** and **Training**: https://community.nxp.com/community/mbdt


MATLAB R2017a
HOME
PLOTS
APPS
BLDC_ClosedLoop_s32k14 * - Simulink
Library: mbd_s32k14_examples - Simulink
Simulink Library Browser
Model Based Design Toolbox for S32K Series/S32K/S32K14x Blocks/S32K14x/Motor Control Blocks/FlexTimer Blocks
S32K On-chip Modules
Modelling, Verification and Validation
Applications
Dual_Edge_Capture
FTM_CHN_ISR
mc_gmclib0_s32k14
Ready
62%
51%

NXP Model-Based Design Toolbox – Preview



Built-in Tools, Scripts and Sources

Simulink Blocks for S32K Peripherals Configuration, RW, ISR

Documentation & Examples

FCAN Module: 0
Tx : PTE5: [CAN0_TX | CAN Tx Channel]
Rx : PTE4: [CAN0_RX | CAN Rx channel]
Rx FIFO : off
CAN FD : off
Pretended Networking : off

FCAN_Config

FCAN Module : 0
Wake up Message Buffer : 0

ID
Data
Length

FCAN_Get_WMB

FCAN Module : 0
Event : Rx Complete

Trigger
MB Index
ID
Data
Length
Timestamp

FCAN_Isr

FCAN Module : 0
Mode : Rx Mailbox Non-blocking
Message Buffer : 0
Message ID : 0

FCAN_Receive

Target Configuration Block

The main functionality of the block is to configure the MCU target, build toolcha

Block Image

Target : S32K148 192KBSRAM
Package : 144-LQFP
System clock : 80 MHz
XTAL clock : External 8 MHz
Compiler : GCC
Target Type : FLASH
Download Code after build : off
Step Tick Interrupt Priority : 0

MBD_S32K1xx_Config_Information

Inputs:

- None

Outputs:

- None

Parameters and Dialog Box

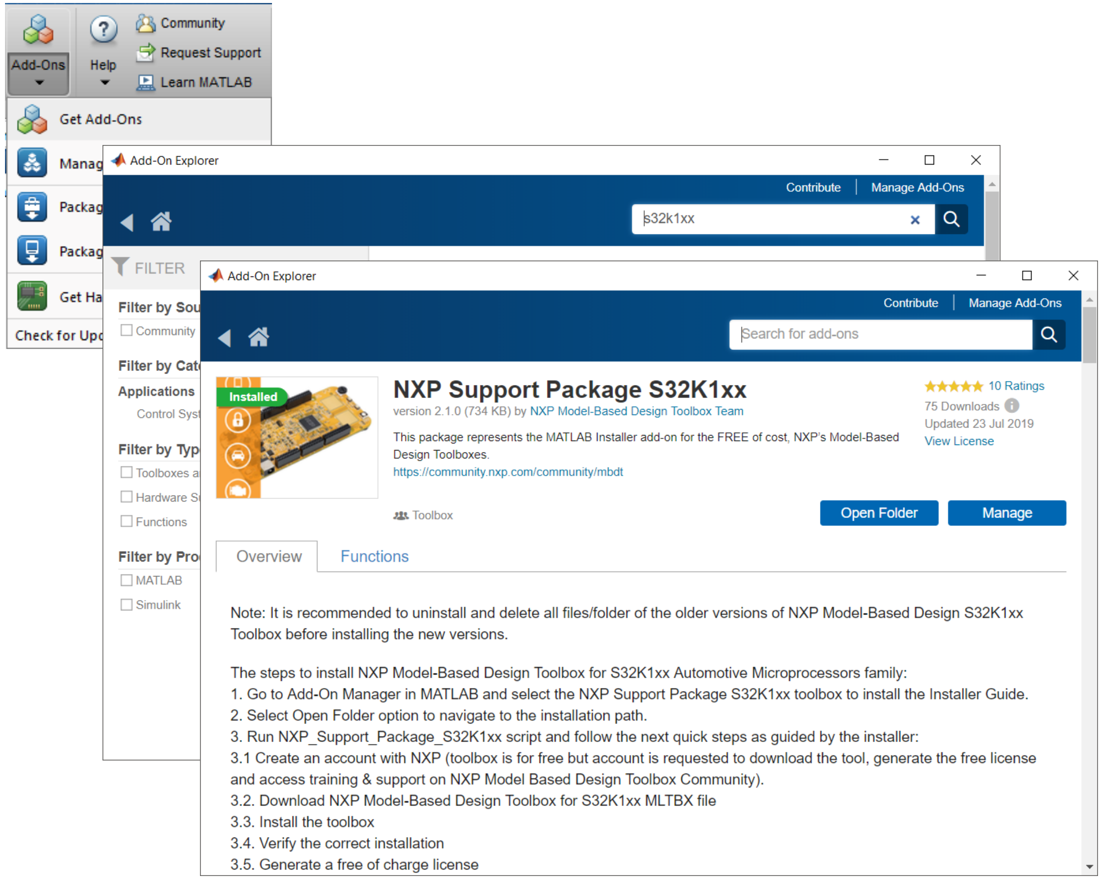
The block dialog consists of the following configuration tabs:

- Target MCU
- Build Toolchain
- Target Connection
- Diagnostics

Target MCU

NXP Model-Based Design Toolbox – Installation

Step 1: Get Add-on



NXP Support Package S32K1xx

version 2.1.0 (734 KB) by NXP Model-Based Design Toolbox Team

★★★★★ 10 Ratings
75 Downloads
Updated 23 Jul 2019
View License

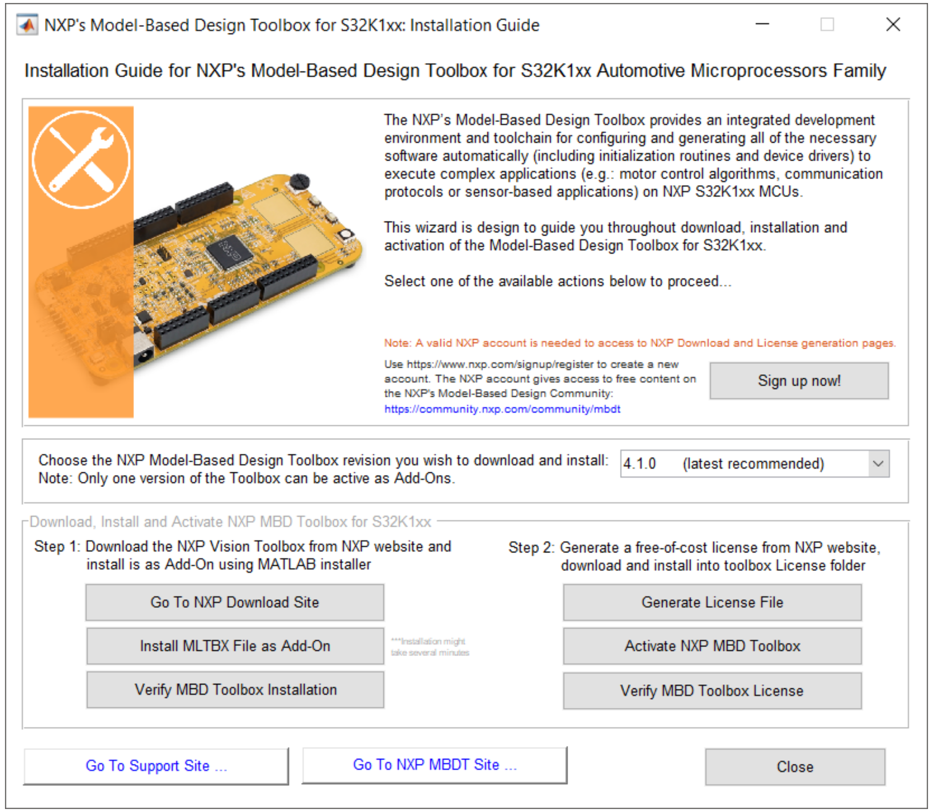
This package represents the MATLAB Installer add-on for the FREE of cost, NXP's Model-Based Design Toolboxes.
https://community.nxp.com/community/mbdt

Toolbox

Note: It is recommended to uninstall and delete all files/folder of the older versions of NXP Model-Based Design S32K1xx Toolbox before installing the new versions.

The steps to install NXP Model-Based Design Toolbox for S32K1xx Automotive Microprocessors family:
1. Go to Add-On Manager in MATLAB and select the NXP Support Package S32K1xx toolbox to install the Installer Guide.
2. Select Open Folder option to navigate to the installation path.
3. Run NXP_Support_Package_S32K1xx script and follow the next quick steps as guided by the installer:
3.1 Create an account with NXP (toolbox is for free but account is requested to download the tool, generate the free license and access training & support on NXP Model Based Design Toolbox Community).
3.2. Download NXP Model-Based Design Toolbox for S32K1xx MLTBX file
3.3. Install the toolbox
3.4. Verify the correct installation
3.5. Generate a free of charge license

Step 2: Install & Validate


NXP's Model-Based Design Toolbox for S32K1xx: Installation Guide
Sign up now!
Go To NXP Download Site
Install MLTBX File as Add-On
Verify MBD Toolbox Installation
Generate License File
Activate NXP MBD Toolbox
Verify MBD Toolbox License
Go To Support Site ...
Go To NXP MBDT Site ...
Close

Installation Guide for NXP's Model-Based Design Toolbox for S32K1xx Automotive Microprocessors Family

The NXP's Model-Based Design Toolbox provides an integrated development environment and toolchain for configuring and generating all of the necessary software automatically (including initialization routines and device drivers) to execute complex applications (e.g.: motor control algorithms, communication protocols or sensor-based applications) on NXP S32K1xx MCUs.

This wizard is design to guide you throughout download, installation and activation of the Model-Based Design Toolbox for S32K1xx.

Select one of the available actions below to proceed...

Note: A valid NXP account is needed to access to NXP Download and License generation pages.

Use https://www.nxp.com/signup/register to create a new account. The NXP account gives access to free content on the NXP's Model-Based Design Community:
https://community.nxp.com/community/mbdt

Choose the NXP Model-Based Design Toolbox revision you wish to download and install: 4.1.0 (latest recommended)
Note: Only one version of the Toolbox can be active as Add-Ons.

Download, Install and Activate NXP MBD Toolbox for S32K1xx

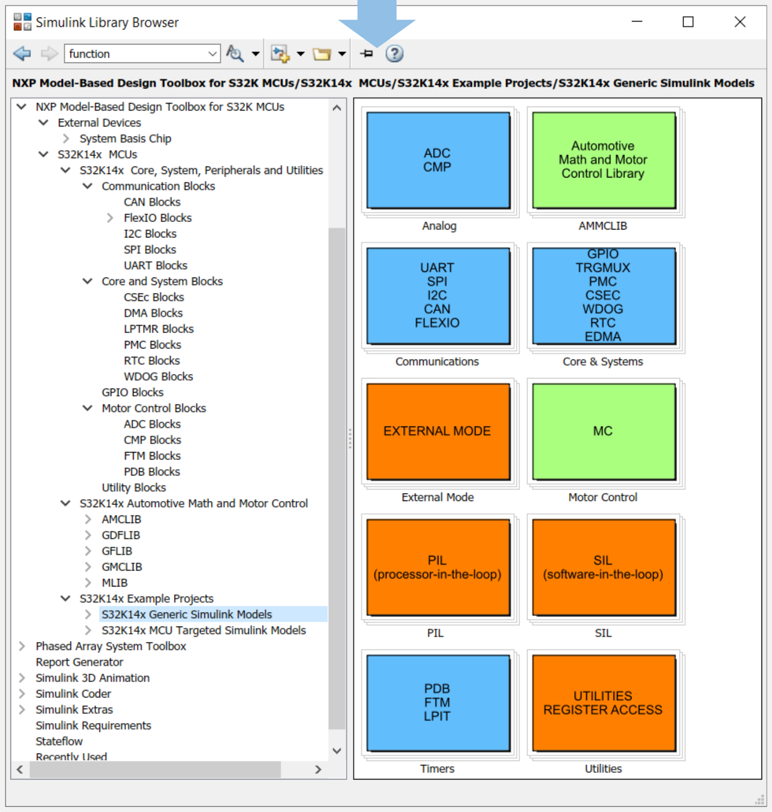
Step 1: Download the NXP Vision Toolbox from NXP website and install is as Add-On using MATLAB installer

***Installation might take several minutes

Step 2: Generate a free-of-cost license from NXP website, download and install into toolbox License folder

NXP Model-Based Design Toolbox – Approach

1 Basic/Advanced Building Blocks



Simulink Library Browser

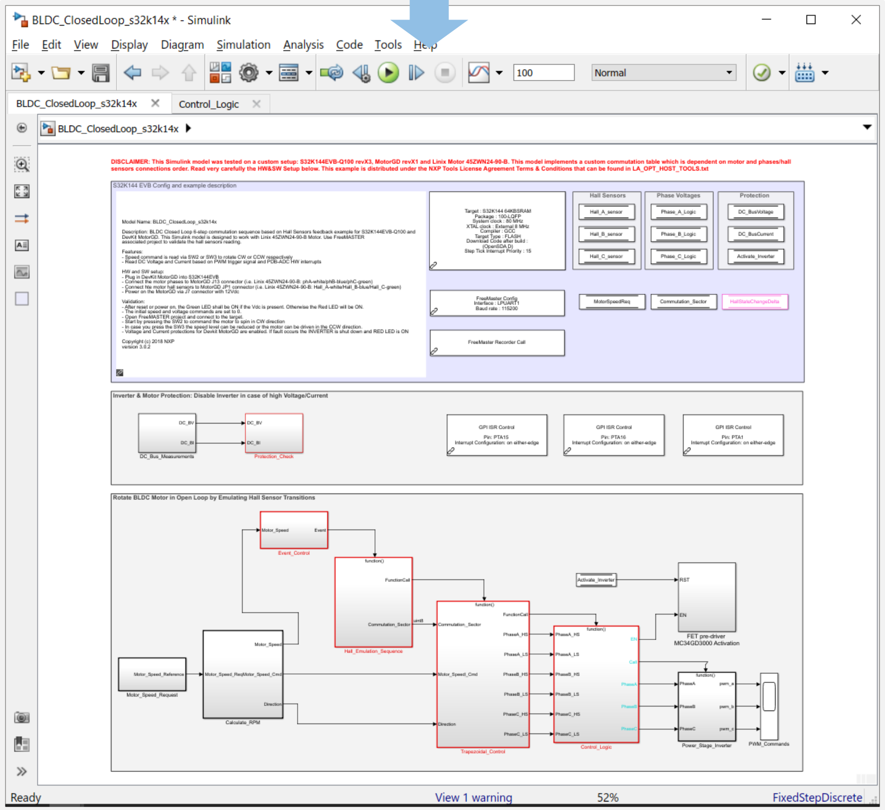
function

NXP Model-Based Design Toolbox for S32K MCUs/S32K14x MCUs/S32K14x Example Projects/S32K14x Generic Simulink Models

- NXP Model-Based Design Toolbox for S32K MCUs
  - External Devices
    - System Basis Chip
  - S32K14x MCUs
    - S32K14x Core, System, Peripherals and Utilities
      - Communication Blocks
        - CAN Blocks
        - FlexIO Blocks
        - I2C Blocks
        - SPI Blocks
        - UART Blocks
      - Core and System Blocks
        - CSEc Blocks
        - DMA Blocks
        - LPTMR Blocks
        - PMC Blocks
        - RTC Blocks
        - WDOG Blocks
      - GPIO Blocks
      - Motor Control Blocks
        - ADC Blocks
        - CMP Blocks
        - FTM Blocks
        - PDB Blocks
      - Utility Blocks
    - S32K14x Automotive Math and Motor Control
      - AMCLIB
      - GDFLIB
      - GFLIB
      - GMCLIB
      - MLIB
    - S32K14x Example Projects
      - S32K14x Generic Simulink Models
      - S32K14x MCU Targeted Simulink Models
- Phased Array System Toolbox
- Report Generator
- Simulink 3D Animation
- Simulink Coder
- Simulink Extras
- Simulink Requirements
- Stateflow
- Recently Used

Block	Label
ADC CMP	Analog
Automotive Math and Motor Control Library	AMMCLIB
UART SPI I2C CAN FLEXIO	Communications
GPIO TRGMUX PMC CSEC WDOG RTC EDMA	Core & Systems
EXTERNAL MODE	External Mode
MC	Motor Control
PIL (processor-in-the-loop)	PIL
SIL (software-in-the-loop)	SIL
PDB FTM LPIT	Timers
UTILITIES REGISTER ACCESS	Utilities

2 Ideas & Designs



BLDC_ClosedLoop_s32k14x * - Simulink

DISCLAIMER: This Simulink model was tested on a custom setup: S32K144EVB-Q100 revX3, MotorGD revX1 and Linix Motor 45ZWN24-90-B. This model implements a custom commutation table which is dependent on motor and phases/hall sensors connections order. Read very carefully the HW&SW Setup below. This example is distributed under the NXP Tools License Agreement Terms & Conditions that can be found in LA_OPT_HOST_TOOLS.txt

S32K144 EVB Config and example description

Model Name: BLDC_ClosedLoop_s32k14x

Description: BLDC Closed Loop 6-step commutation sequence based on Hall Sensors feedback example for S32K144EVB-Q100 and DevKit MotorGD. This Simulink model is designed to work with Linix 45ZWN24-90-B Motor. Use FreeMASTER associated project to validate the hall sensors reading.

Features:
- Speed command is read via SW2 or SW3 to rotate CW or CCW respectively
- Read DC Voltage and Current based on PWM trigger signal and PDB-ADC HW interrupts

HW and SW setup:
- Plug in DevKit MotorGD into S32K144EVB
- Connect the motor phases to MotorGD J13 connector (i.e. Linix 45ZWN24-90-B: phA-white/phB-blue/phC-green)
- Connect hte motor hall sensors to MotorGD JP1 connector (i.e. Linix 45ZWN24-90-B: Hall_A-white/Hall_B-blue/Hall_C-green)
- Power on the MotorGD via J7 connector with 12Vdc

Validation:
- After reset or power on, the Green LED shall be ON if the Vdc is present. Otherwise the Red LED will be ON.
- The initial speed and voltage commands are set to 0.
- Open FreeMASTER project and connect to the target.
- Start by pressing the SW2 to command the motor to spin in CW direction
- In case you press the SW3 the speed level can be reduced or the motor can be driven in the CCW direction.
- Voltage and Current protections for Devkit MotorGD are enabled. If fault occurs the INVERTER is shut down and RED LED is ON

Copyright (c) 2018 NXP
version 3.0.2

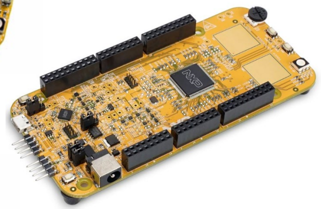
Target : S32K144 64KBSRAM
Package : 100-LQFP
System clock : 80 MHz
XTAL clock : External 8 MHz
Compiler : GCC
Target Type : FLASH
Download Code after build : (OpenSDA D)
Step Tick Interrupt Priority : 15

FreeMaster Config
Interface : LPUART1
Baud rate : 115200

FreeMaster Recorder Call

3 Easy Prototyping

S32K142 EVB



S32K144 EVB

S32K146 EVB

S32K148 EVB

NXP Model-Based Design Toolbox – S32K Features

Peripherals

- **Generic**
  - Digital & Analogue I/Os
  - HW Timers (normal/low power)
  - Various ISR sources
- **Communication**
  - CAN and CAN-FD
  - SPI, I2C, UART, FlexIO, **LIN**
- **Motor Control**
  - PWM (edge/center/combined, **SW**)
  - ADC & CMP
  - FTM (Flex Timer Module)
  - PDB (Programmable Delay Block)
- **Core & Subsystems**
  - CSEC
  - DMA
  - PMC (Power Management)
  - RTC
  - WDOG

Configuration/Modes

- **MCU Options**
  - Multiple packages (LQFP and BGA)
  - Multiple crystal frequencies
- **Compilers**
  - GCC 6.3
  - Green Hills MULTI
  - IAR
  - RAM/FLASH support
  - Custom Memory Map
- **Simulation Modes**
  - Model in the Loop (MIL)
  - Software in the Loop (SIL)
  - Processor in the Loop (PIL)
  - External Mode
  - AUTOSAR (SIL/PIL/Profiling)

Utilities

- **AMMCLIB Library 1.1.7**
  - MLIB
  - GFLIB/GDFLIB
  - GMCLIB/AMCLIB
- **S32 FreeMASTER**
  - Data acquisition / Calibration
  - Customized GUI
- **S32 Bootloader**
  - Program application into MCU Flash/RAM
- **Profiler Function**
  - Exec. time measurement
  - Available in PIL
  - Available in standalone
- **Read and Write Blocks**
  - Memory
  - Registers

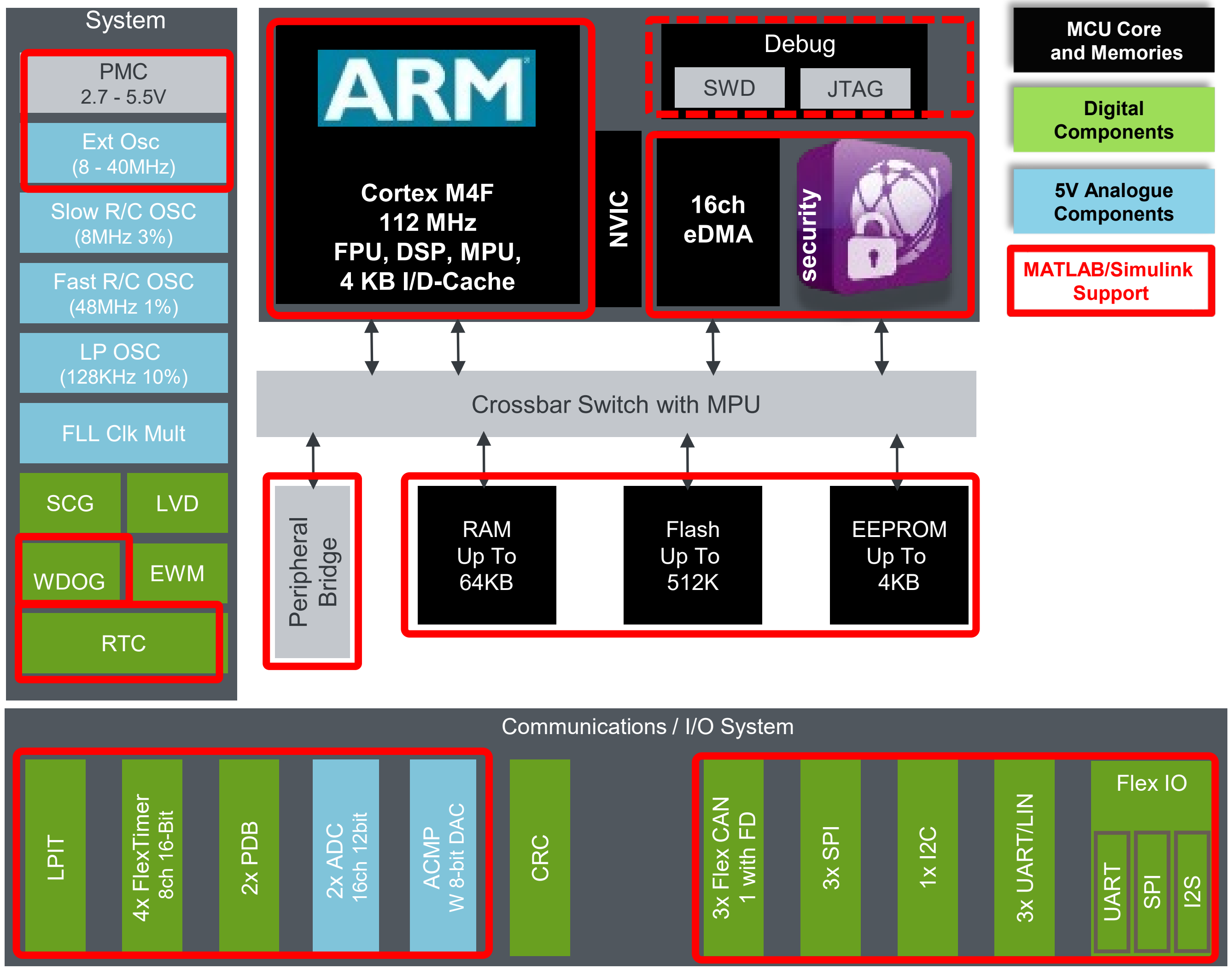
MCUs Supported

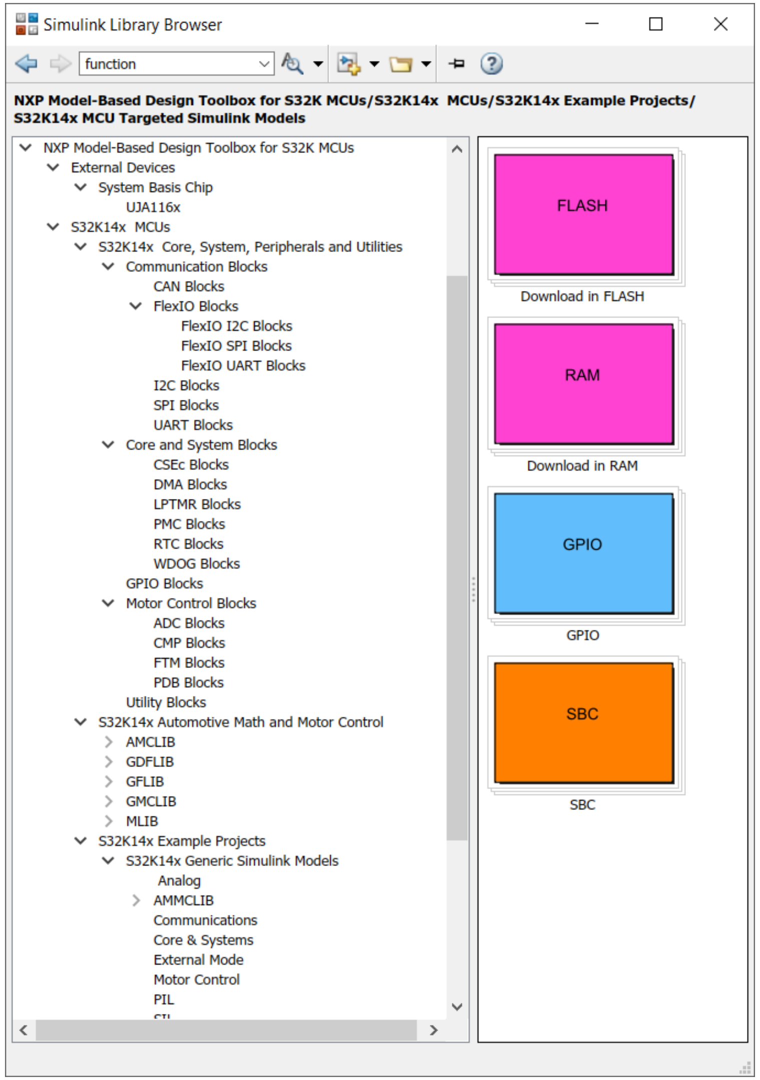
- S32K142
- S32K144
- S32K146
- S32K148
- S32K116
- S32K118

External Devices

- SBC UJA 113x
- SBC UJA 116x
- MC33GD3000
- MC34GD3000
- MC33937
- MC34937

NXP MBDT 2018.R1 release – S32K Peripheral Mapping



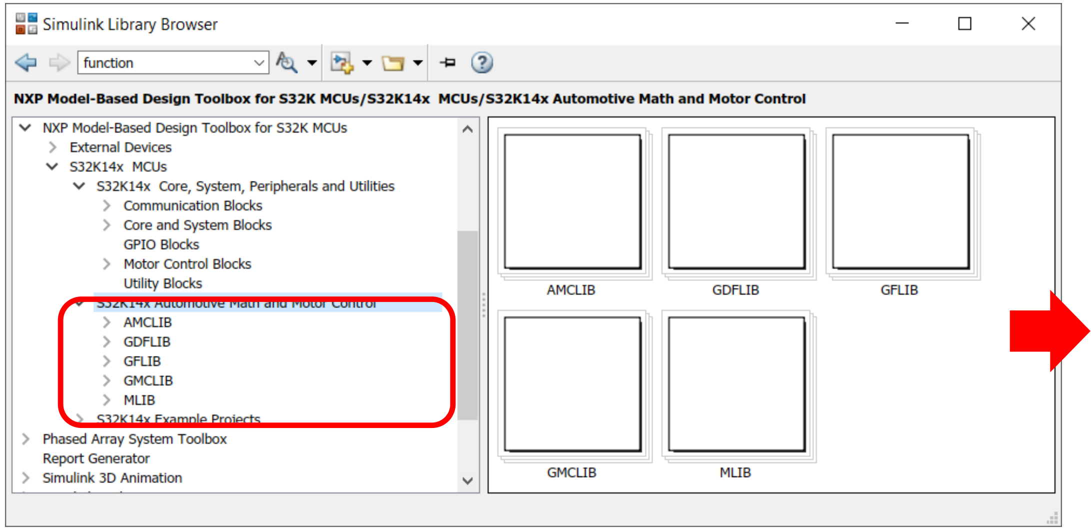


NXP Model-Based Design Toolbox for S32K MCUs/S32K14x MCUs/S32K14x Example Projects/ S32K14x MCU Targeted Simulink Models

```
NXP Model-Based Design Toolbox for S32K MCUs
  External Devices
    System Basis Chip
      UJA116x
  S32K14x  MCUs
    S32K14x  Core, System, Peripherals and Utilities
      Communication Blocks
        CAN Blocks
        FlexIO Blocks
          FlexIO I2C Blocks
          FlexIO SPI Blocks
          FlexIO UART Blocks
        I2C Blocks
        SPI Blocks
        UART Blocks
      Core and System Blocks
        CSEc Blocks
        DMA Blocks
        LPTMR Blocks
        PMC Blocks
        RTC Blocks
        WDOG Blocks
      GPIO Blocks
      Motor Control Blocks
        ADC Blocks
        CMP Blocks
        FTM Blocks
        PDB Blocks
      Utility Blocks
    S32K14x Automotive Math and Motor Control
      AMCLIB
      GDFLIB
      GFLIB
      GMCLIB
      MLIB
    S32K14x Example Projects
      S32K14x Generic Simulink Models
         Analog
        AMMCLIB
        Communications
        Core & Systems
        External Mode
        Motor Control
        PIL
```

Automotive & Math and Motor Control Library

- Precompiled software library containing building blocks for a wide range of motor control applications
- Easy migration between platforms with minimized effort
- Production ready SW (SPICE Level 3, CMMI and ISO9001/TS16949)
- Control loop modeling with MATLAB/Simulink® models and examples

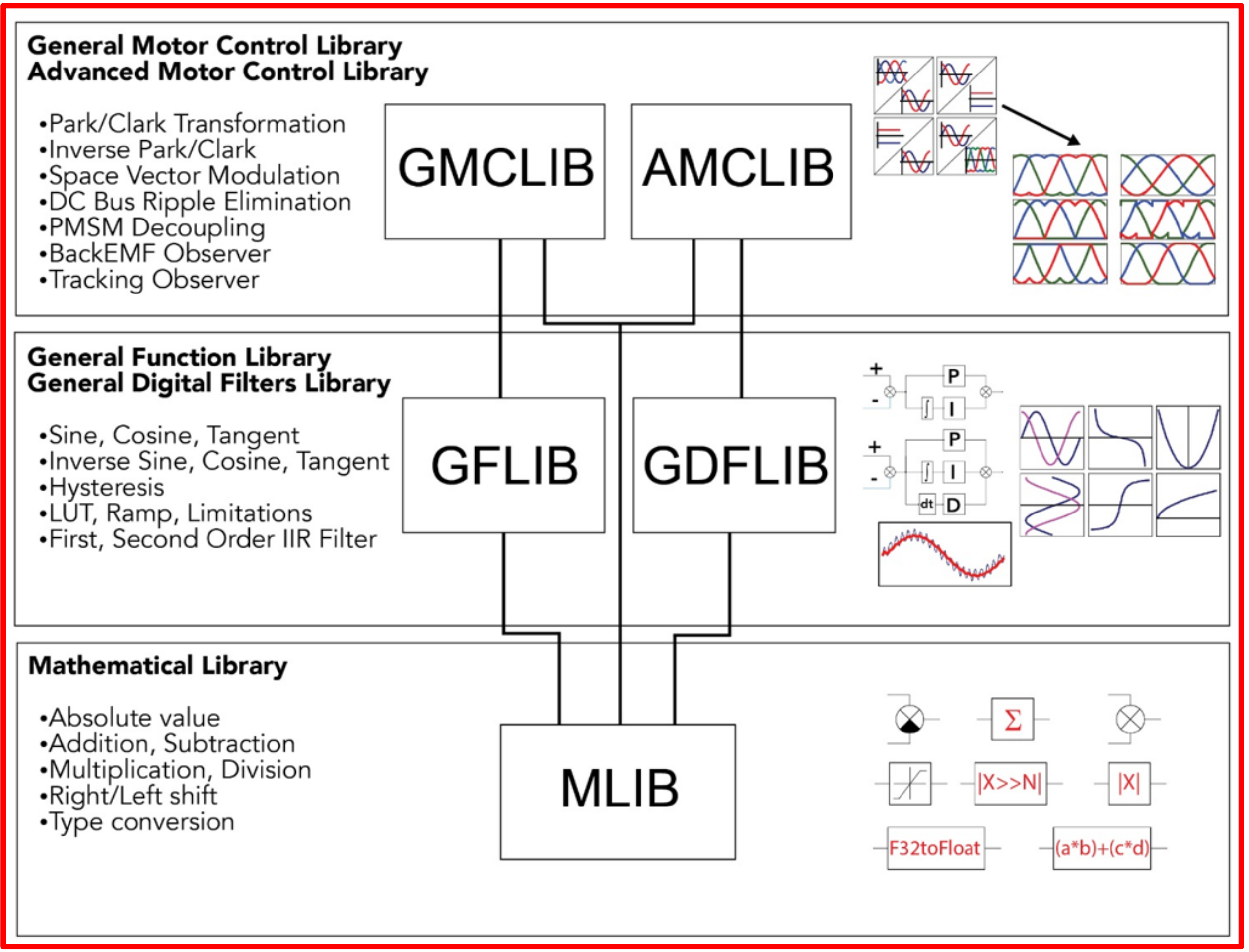


Simulink Library Browser

function

NXP Model-Based Design Toolbox for S32K MCUs/S32K14x MCUs/S32K14x Automotive Math and Motor Control

- NXP Model-Based Design Toolbox for S32K MCUs
  - External Devices
  - S32K14x MCUs
    - S32K14x Core, System, Peripherals and Utilities
      - Communication Blocks
      - Core and System Blocks
      - GPIO Blocks
      - Motor Control Blocks
      - Utility Blocks
    - S32K14x Automotive Math and Motor Control
      - AMCLIB
      - GDFLIB
      - GFLIB
      - GMCLIB
      - MLIB
    - S32K14x Example Projects
- Phased Array System Toolbox
- Report Generator
- Simulink 3D Animation



General Motor Control Library
Advanced Motor Control Library

- Park/Clark Transformation
- Inverse Park/Clark
- Space Vector Modulation
- DC Bus Ripple Elimination
- PMSM Decoupling
- BackEMF Observer
- Tracking Observer

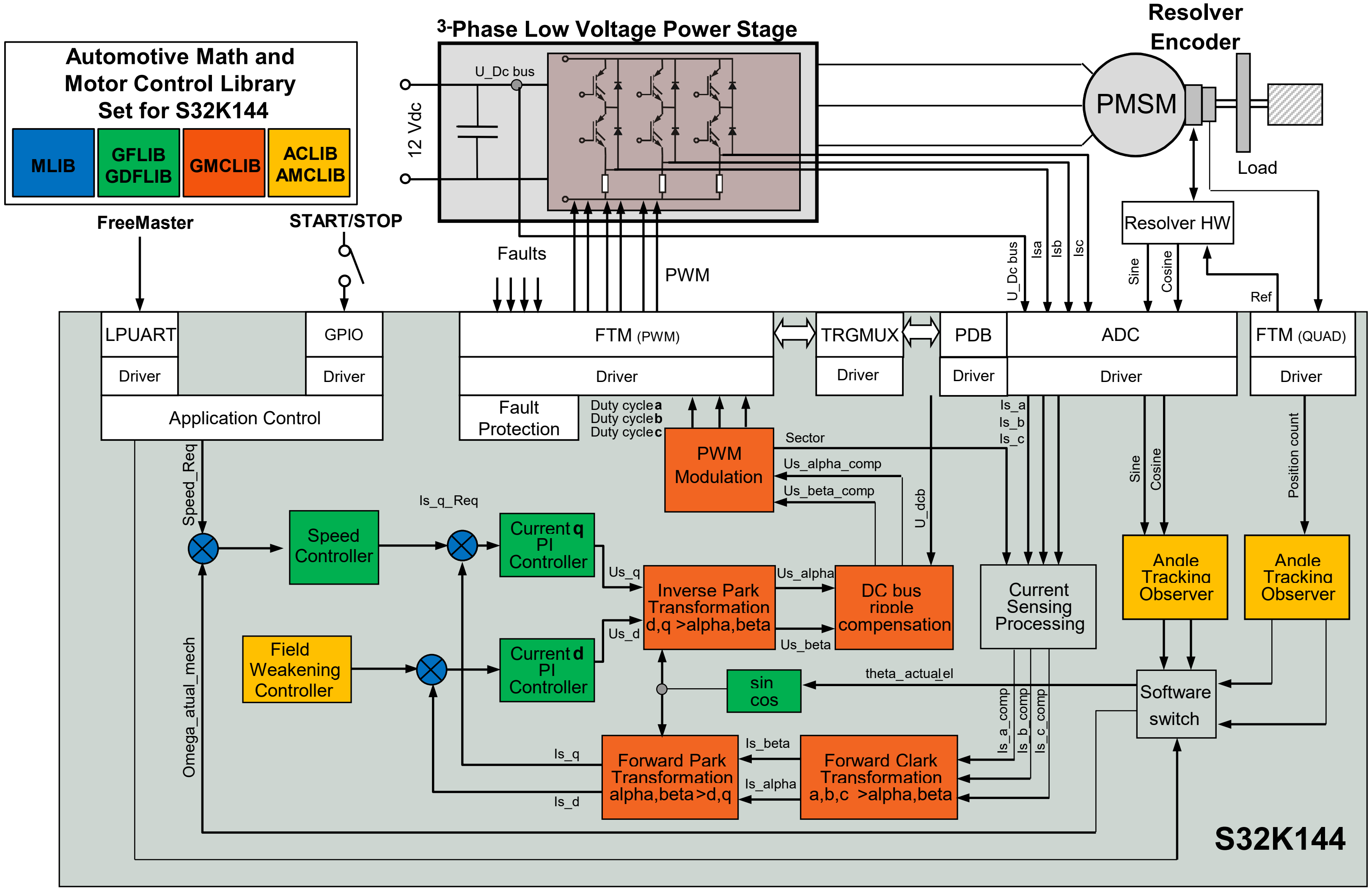
General Function Library
General Digital Filters Library

- Sine, Cosine, Tangent
- Inverse Sine, Cosine, Tangent
- Hysteresis
- LUT, Ramp, Limitations
- First, Second Order IIR Filter

Mathematical Library

- Absolute value
- Addition, Subtraction
- Multiplication, Division
- Right/Left shift
- Type conversion

Application Example for PMSM/FOC with AMMCLIB



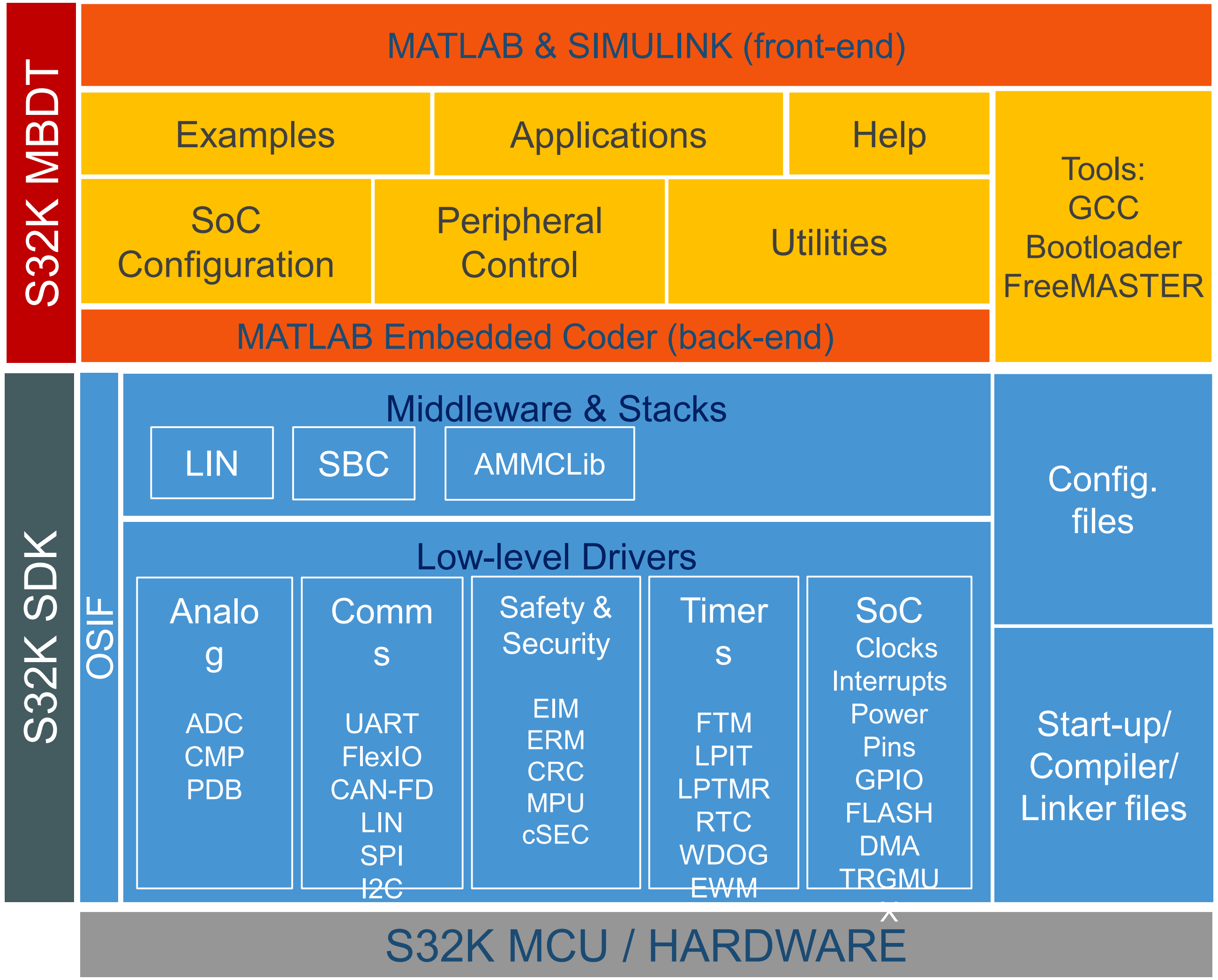
NXP S32K MBDT – “Ultimate” Abstraction Layer

S32K Model-Based Design Layer

- MATLAB/Simulink oriented
- Drag-drop programming
- Automatic C-code generation
- Easy to port
- Peripherals and Applications Examples
- Dedicated online Community

S32K SDK Enablement Layer

- Layered Software Architecture
- Documented Source Code and Examples
- Integrated with S32 DS and other IDEs
- Featuring various Middleware
- FreeRTOS integration
- Multiple toolchains supported
- Several examples and demos



S32K MBDT

MATLAB & SIMULINK (front-end)

Examples

Applications

Help

SoC Configuration

Peripheral Control

Utilities

MATLAB Embedded Coder (back-end)

Tools:
GCC
Bootloader
FreeMASTER

S32K SDK

OSIF

Middleware & Stacks

LIN

SBC

AMMCLib

Low-level Drivers

Analog
ADC
CMP
PDB

Comms
UART
FlexIO
CAN-FD
LIN
SPI
I2C

Safety & Security
EIM
ERM
CRC
MPU
cSEC

Timers
FTM
LPIT
LPTMR
RTC
WDOG
EWM

SoC
Clocks
Interrupts
Power
Pins
GPIO
FLASH
DMA
TRGMUX

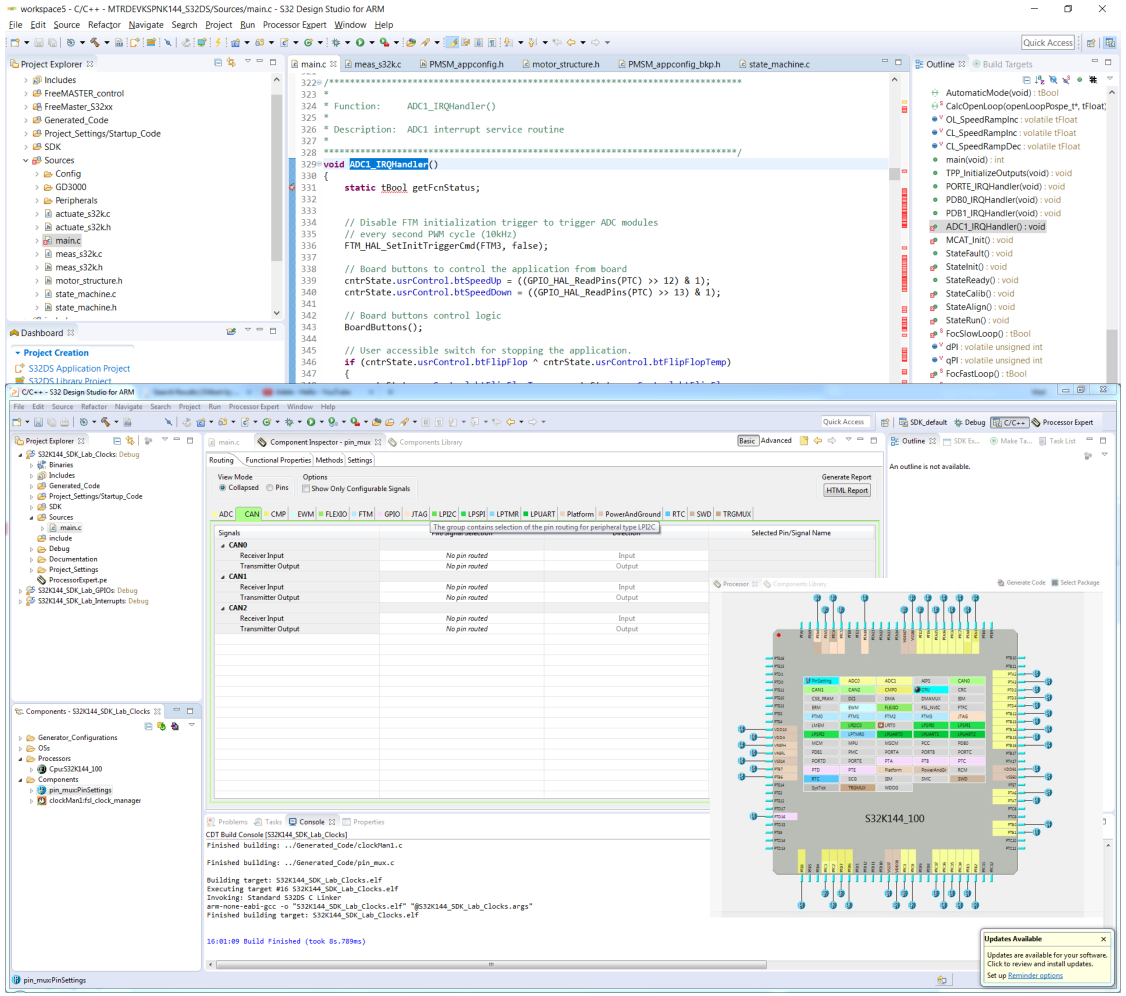
Config. files

Start-up/
Compiler/
Linker files

S32K MCU / HARDWARE

NXP S32 Design Studio – IDE

- Free of charge
- Unlimited code size
- GNU compiler & debugger integrated
- S32 SDK integrated (graphical configuration)
- Processor Expert integrated (automatic code generator)
- Not intended to compete with premium 3rd party IDEs



```c
/*****************************************************************************
*
* Function:     ADC1_IRQHandler()
*
* Description:  ADC1 interrupt service routine
*
*****************************************************************************/
void ADC1_IRQHandler()
{
    static tBool getFcnStatus;


    // Disable FTM initialization trigger to trigger ADC modules
    // every second PWM cycle (10kHz)
    FTM_HAL_SetInitTriggerCmd(FTM3, false);

    // Board buttons to control the application from board
    cntrState.usrControl.btSpeedUp = ((GPIO_HAL_ReadPins(PTC) >> 12) & 1);
    cntrState.usrControl.btSpeedDown = ((GPIO_HAL_ReadPins(PTC) >> 13) & 1);

    // Board buttons control logic
    BoardButtons();

    // User accessible switch for stopping the application.
    if (cntrState.usrControl.btFlipFlop ^ cntrState.usrControl.btFlipFlopTemp)
    {
```

CDT Build Console [S32K144_SDK_Lab_Clocks]

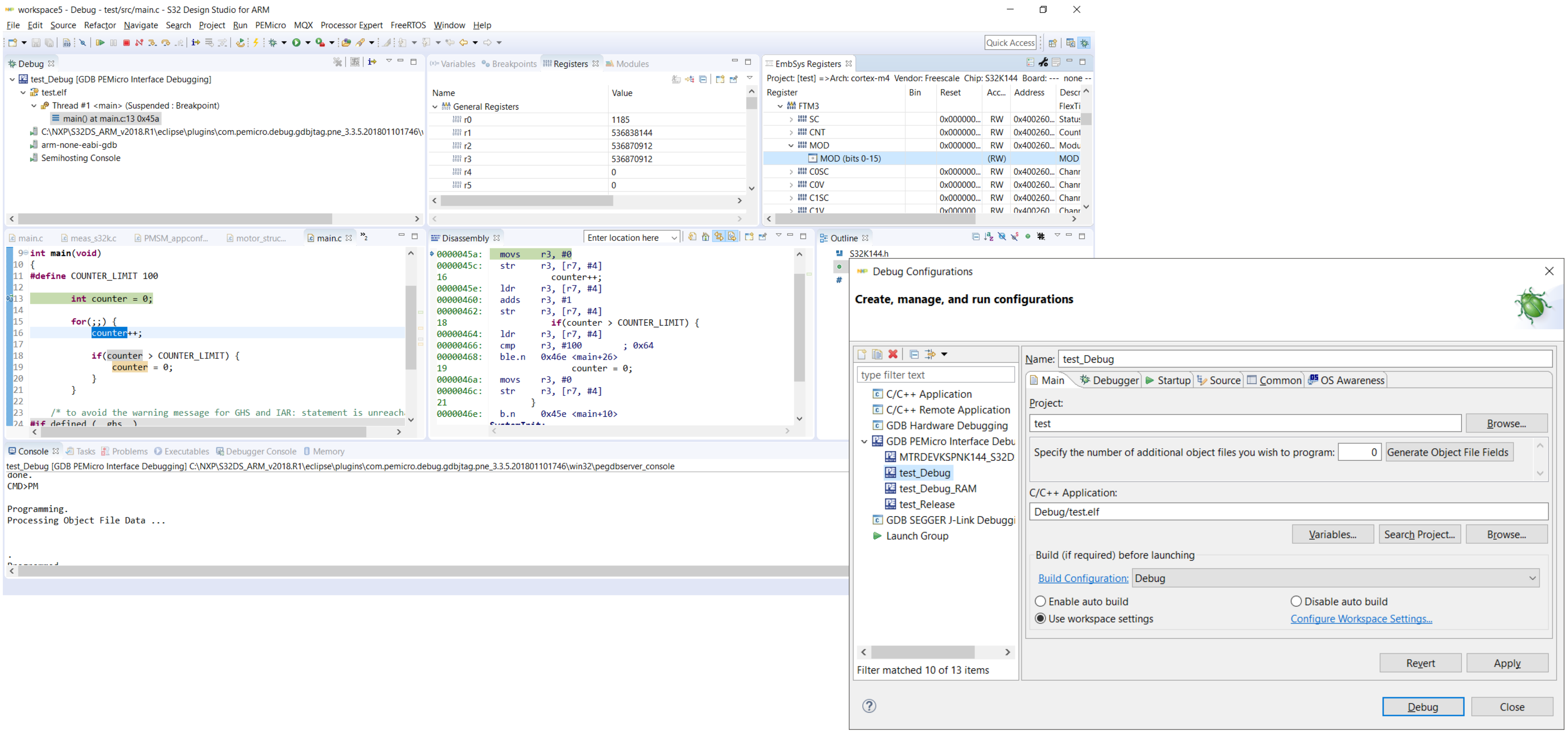
```
Finished building: ../Generated_Code/clockMan1.c

Finished building: ../Generated_Code/pin_mux.c

Building target: S32K144_SDK_Lab_Clocks.elf
Executing target #16 S32K144_SDK_Lab_Clocks.elf
Invoking: Standard S32DS C Linker
arm-none-eabi-gcc -o "S32K144_SDK_Lab_Clocks.elf" "@S32K144_SDK_Lab_Clocks.args"
Finished building target: S32K144_SDK_Lab_Clocks.elf


16:01:09 Build Finished (took 8s.789ms)
```

S32 Design Studio – Application Debugging



workspace5 - Debug - test/src/main.c - S32 Design Studio for ARM

```
test_Debug [GDB PEMicro Interface Debugging]
  test.elf
    Thread #1 <main> (Suspended : Breakpoint)
      main() at main.c:13 0x45a
  C:\NXP\S32DS_ARM_v2018.R1\eclipse\plugins\com.pemicro.debug.gdbjtag.pne_3.3.5.201801101746\
  arm-none-eabi-gdb
  Semihosting Console
```

Name	Value
General Registers	
r0	1185
r1	536838144
r2	536870912
r3	536870912
r4	0
r5	0

Project: [test] =>Arch: cortex-m4 Vendor: Freescale Chip: S32K144 Board: --- none --

Register	Bin	Reset	Acc...	Address	Descr
FTM3					FlexTi
SC		0x000000...	RW	0x400260...	Status
CNT		0x000000...	RW	0x400260...	Count
MOD		0x000000...	RW	0x400260...	Modu
MOD (bits 0-15)			(RW)		MOD
C0SC		0x000000...	RW	0x400260...	Chanr
C0V		0x000000...	RW	0x400260...	Chanr
C1SC		0x000000...	RW	0x400260...	Chanr
C1V		0x000000	RW	0x400260	Chanr

```c
int main(void)
{
#define COUNTER_LIMIT 100

	int counter = 0;

	for(;;) {
		counter++;

		if(counter > COUNTER_LIMIT) {
			counter = 0;
		}
	}

    /* to avoid the warning message for GHS and IAR: statement is unreach
#if defined (  ghs  )
```

```
0000045a:   movs     r3, #0
0000045c:   str      r3, [r7, #4]
16                 counter++;
0000045e:   ldr      r3, [r7, #4]
00000460:   adds     r3, #1
00000462:   str      r3, [r7, #4]
18                 if(counter > COUNTER_LIMIT) {
00000464:   ldr      r3, [r7, #4]
00000466:   cmp      r3, #100         ; 0x64
00000468:   ble.n    0x46e <main+26>
19                         counter = 0;
0000046a:   movs     r3, #0
0000046c:   str      r3, [r7, #4]
21                 }
0000046e:   b.n      0x45e <main+10>
```

```
test_Debug [GDB PEMicro Interface Debugging] C:\NXP\S32DS_ARM_v2018.R1\eclipse\plugins\com.pemicro.debug.gdbjtag.pne_3.3.5.201801101746\win32\pegdbserver_console
done.
CMD>PM

Programming.
Processing Object File Data ...
```

Debug Configurations

Create, manage, and run configurations

- C/C++ Application
- C/C++ Remote Application
- GDB Hardware Debugging
- GDB PEMicro Interface Debu
  - MTRDEVKSPNK144_S32D
  - test_Debug
  - test_Debug_RAM
  - test_Release
- GDB SEGGER J-Link Debuggi
- Launch Group

Filter matched 10 of 13 items

Name: test_Debug

Project: test

Specify the number of additional object files you wish to program: 0

C/C++ Application: Debug/test.elf

Build (if required) before launching

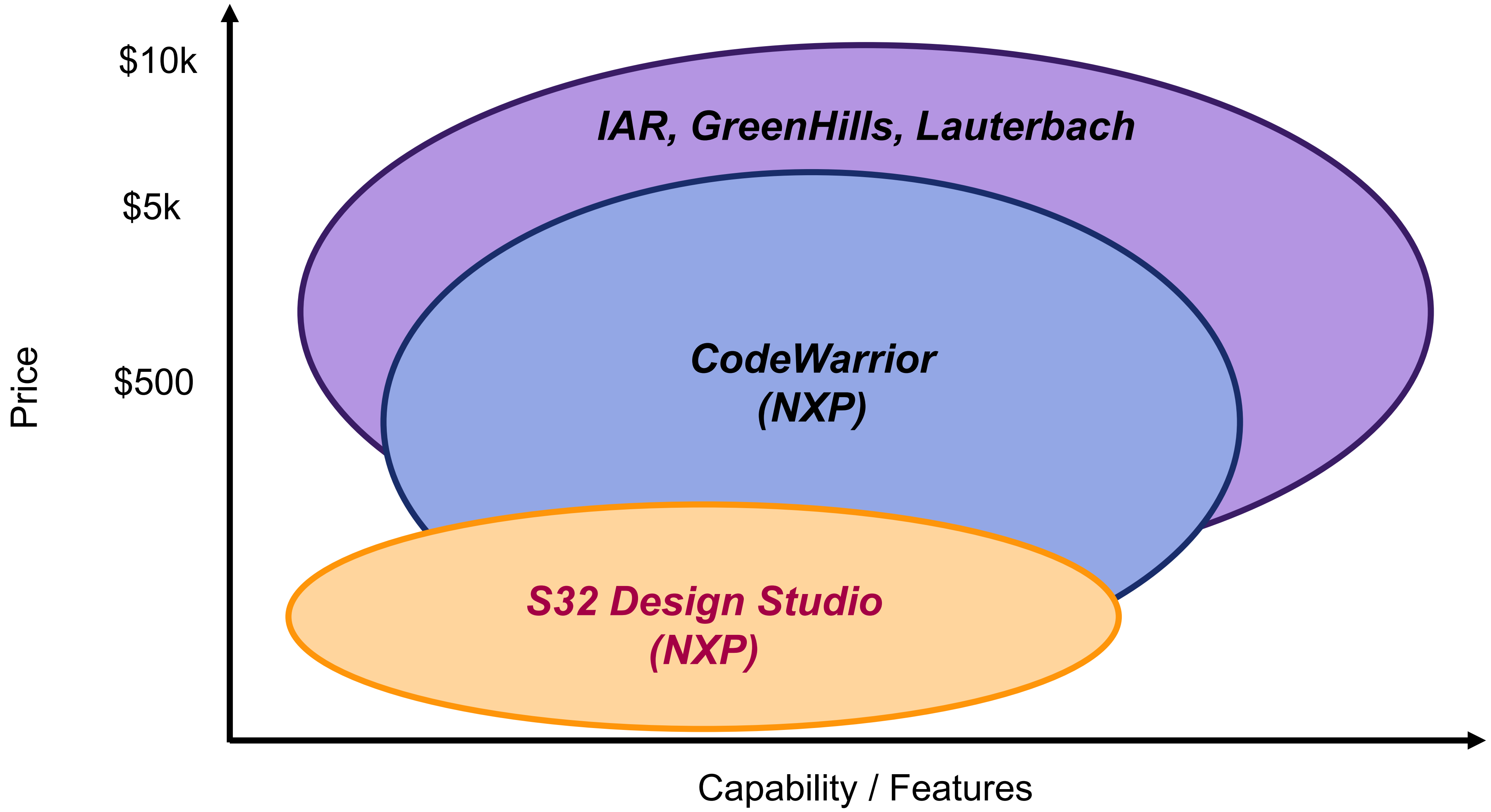
Build Configuration: Debug

○ Enable auto build ○ Disable auto build

● Use workspace settings

Configure Workspace Settings...

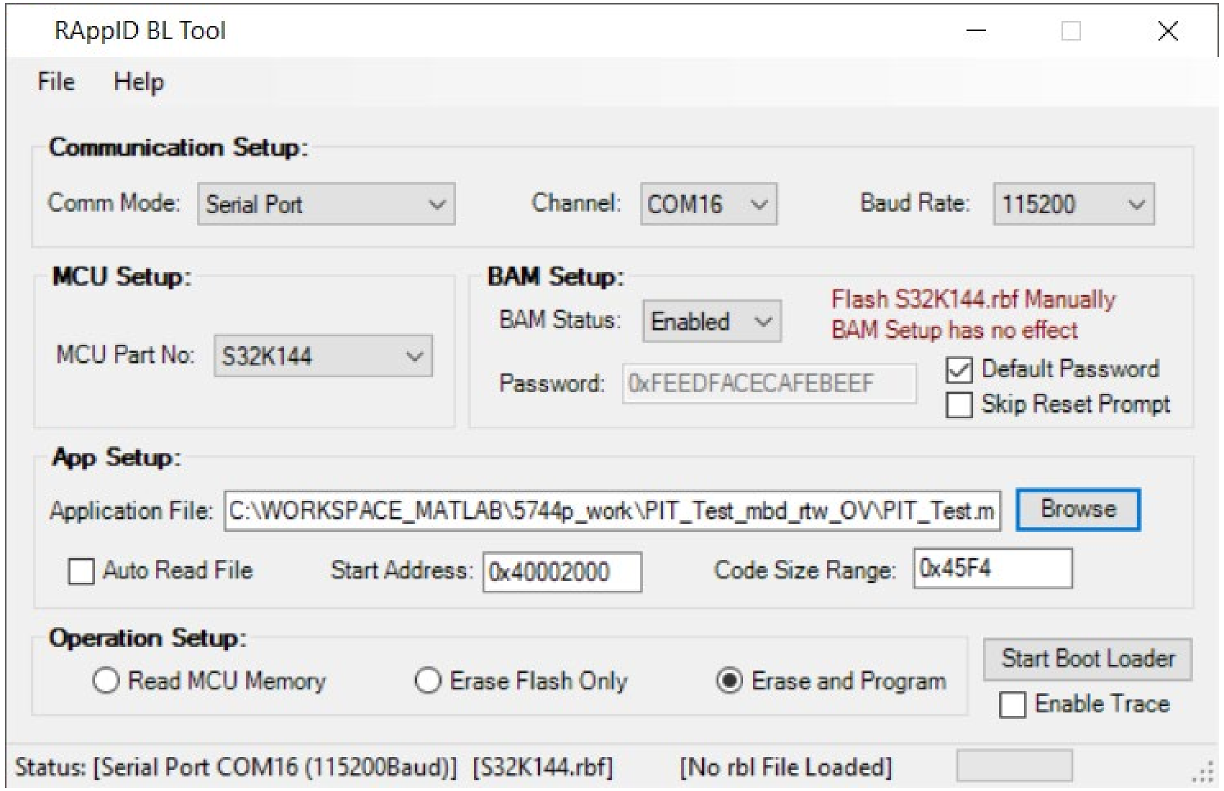
NXP & 3rd Party IDEs – Performance/Price Map



RAppID Bootloader Utility

Provides a streamlined method for programming code into FLASH or RAM on either target EVBs or custom boards. Once programming is complete, the application code automatically starts.

- Modes of Operation
  - Use as a stand-alone PC desktop GUI utility
  - Integration with different user required tools chains through a command line interface (i.e. Eclipse Plug-in, MATLAB/Simulink, …)
- MCUs Supported
  - MPC55xx
  - MPC56xx
  - MPC57xx
  - S12ZVM
  - S32Kxxx,
  - KVxx
  - 56F82xx/56F84xx.



RAppID BL Tool

Communication Setup:

Comm Mode: Serial Port
Channel: COM16
Baud Rate: 115200

MCU Setup:

MCU Part No: S32K144

BAM Setup:

BAM Status: Enabled
Password: 0xFEEDFACECAFEBEEF

Flash S32K144.rbf Manually
BAM Setup has no effect

☑ Default Password
□ Skip Reset Prompt

App Setup:

Application File: C:\WORKSPACE_MATLAB\5744p_work\PIT_Test_mbd_rtw_OV\PIT_Test.m

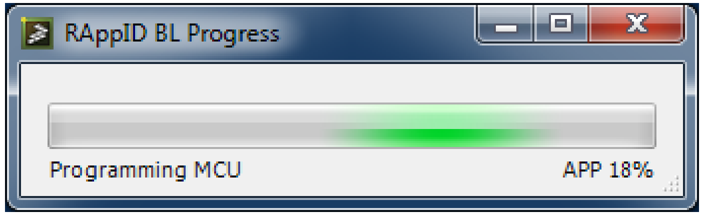
□ Auto Read File
Start Address: 0x40002000
Code Size Range: 0x45F4

Operation Setup:

○ Read MCU Memory ○ Erase Flash Only ● Erase and Program

□ Enable Trace

Status: [Serial Port COM16 (115200Baud)] [S32K144.rbf] [No rbl File Loaded]



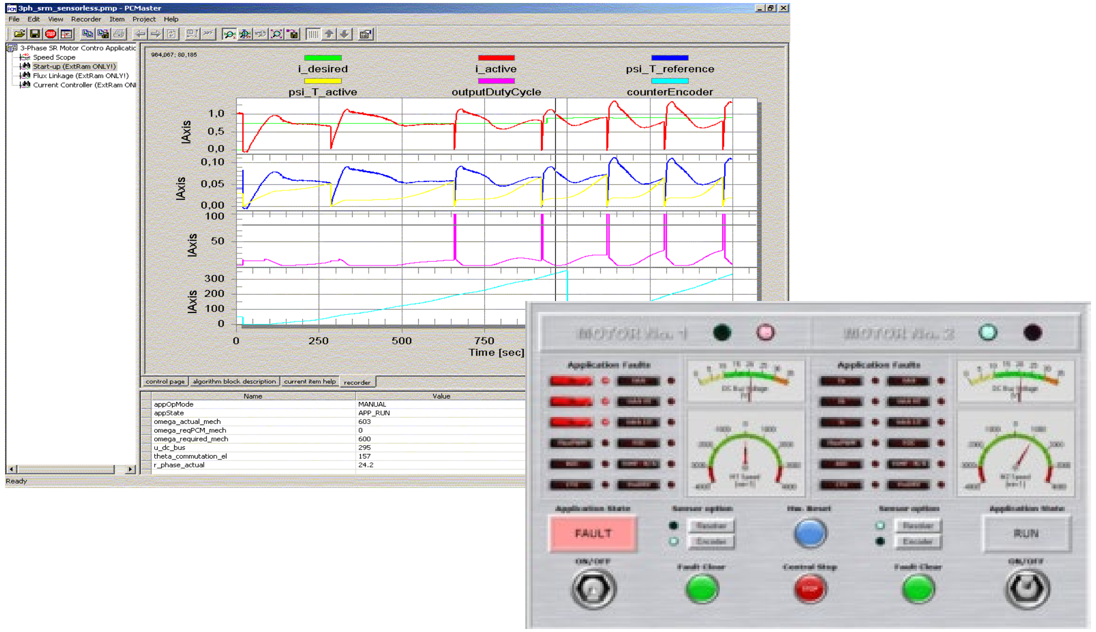
RAppID BL Progress

Programming MCU APP 18%

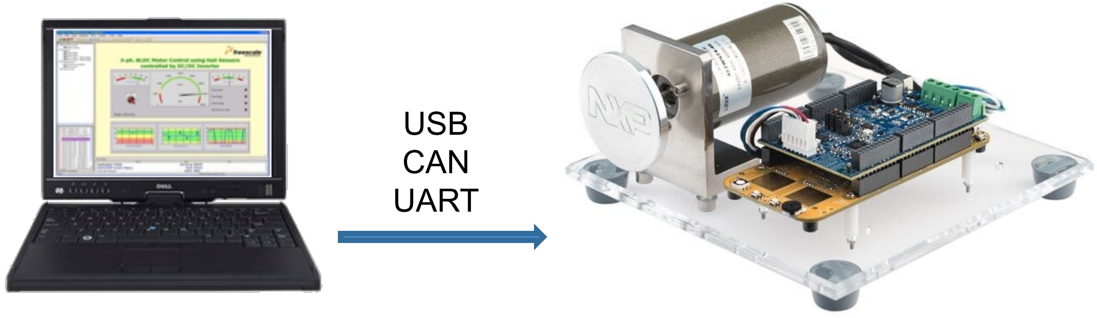
FreeMASTER — Run Time Debugging Tool

- User-friendly tool for real-time debug monitor and data visualization
  - Non-intrusive monitoring of variables on a running system
  - Display multiple variables changing over time on an oscilloscope-like display, or view the data in text form
  - Communicates with an on-target driver via USB, CAN, UART

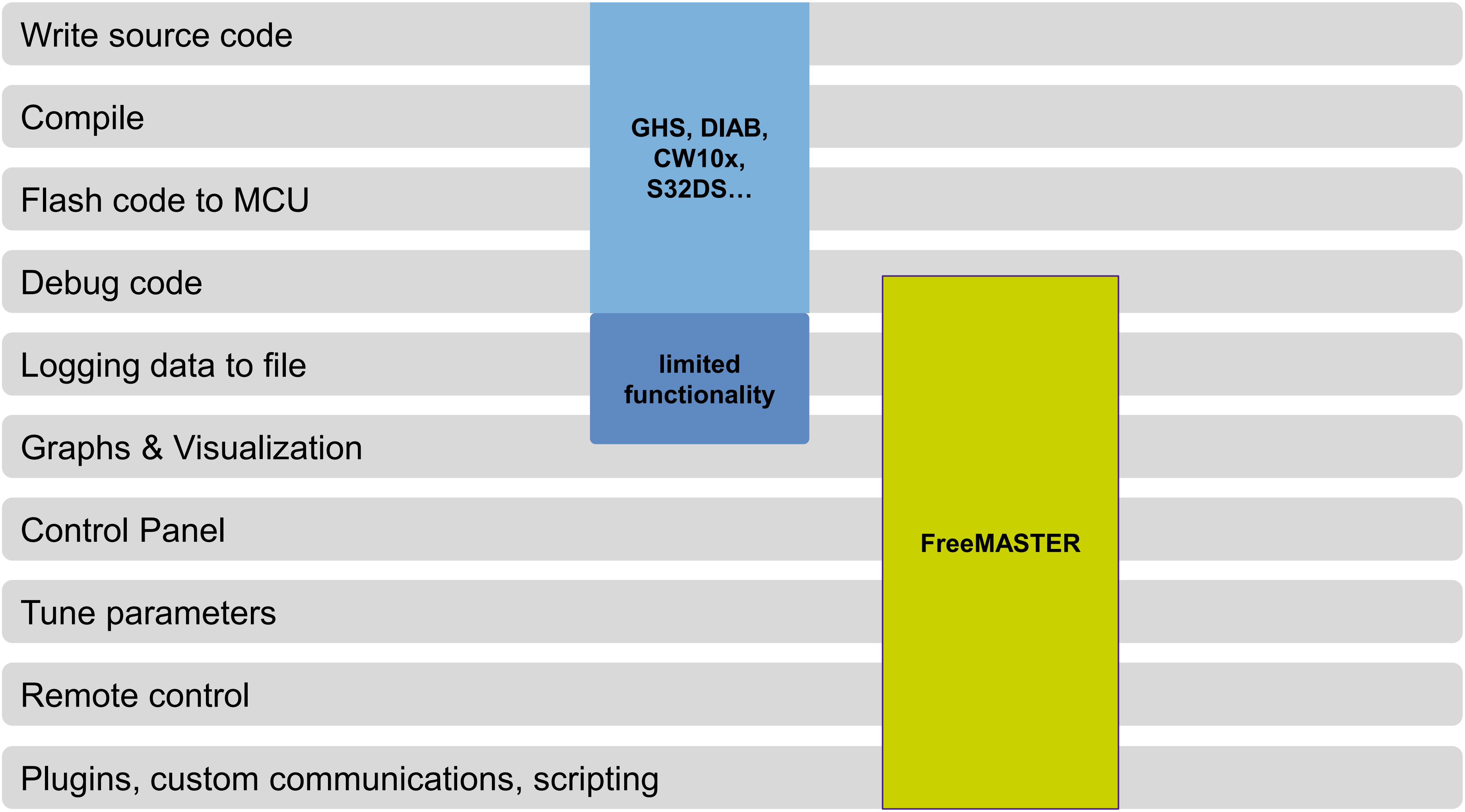
- Establish a Data Trace on Target
  - Set up buffer (up to 64 KB), sampling rate and trigger
  - Near 10-µs resolution



Name	Value
appOpMode	MANUAL
appState	APP_RUN
omega_actual_mech	603
omega_reqPCM_mech	0
omega_required_mech	600
u_dc_bus	295
theta_commutation_el	157
r_phase_actual	24.2



FreeMASTER vs. IDE & Debugger



Write source code

Compile

Flash code to MCU

Debug code

Logging data to file

Graphs & Visualization

Control Panel

Tune parameters

Remote control

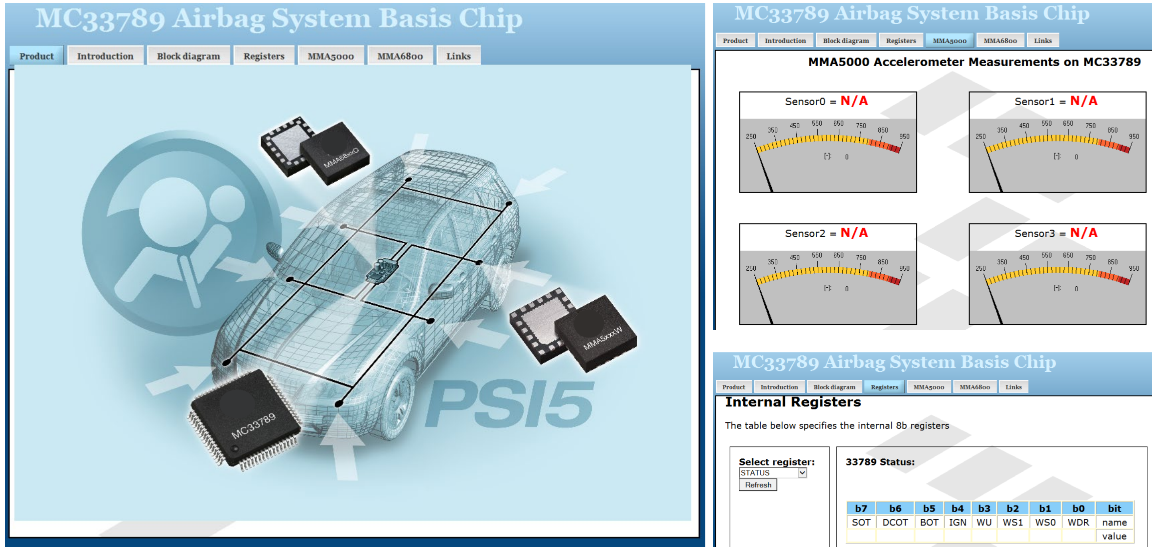
Plugins, custom communications, scripting

GHS, DIAB, CW10x, S32DS…

limited functionality

FreeMASTER

FreeMASTER – Reference Designs and Demos



MMA5000 Accelerometer Measurements on MC33789

Sensor0 = N/A

Sensor1 = N/A

Sensor2 = N/A

Sensor3 = N/A

Internal Registers

The table below specifies the internal 8b registers

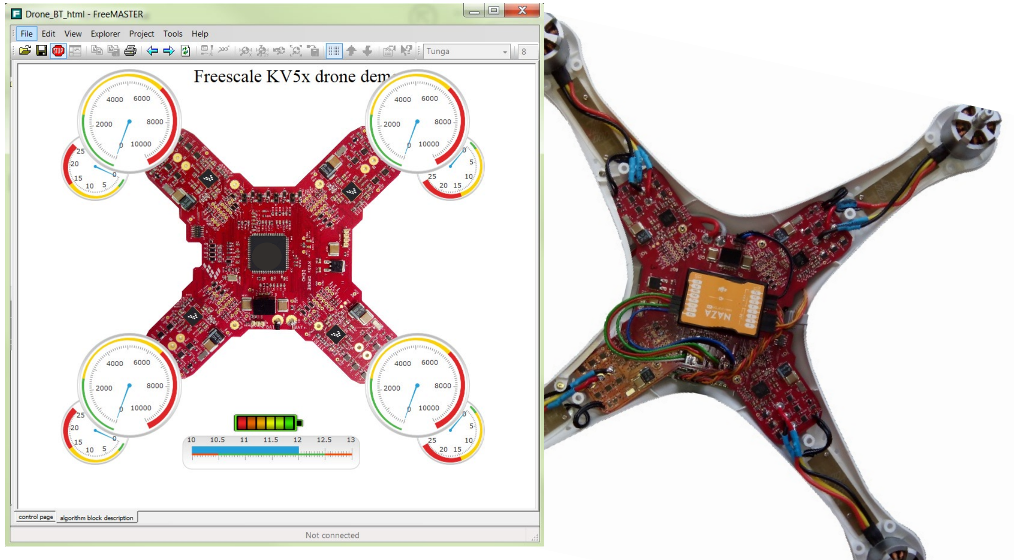
Select register: STATUS

Refresh

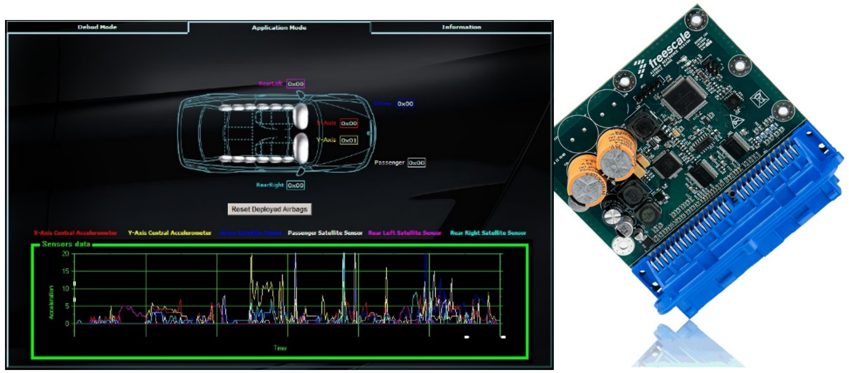
33789 Status:

b7	b6	b5	b4	b3	b2	b1	b0	bit
SOT	DCOT	BOT	IGN	WU	WS1	WS0	WDR	name
								value

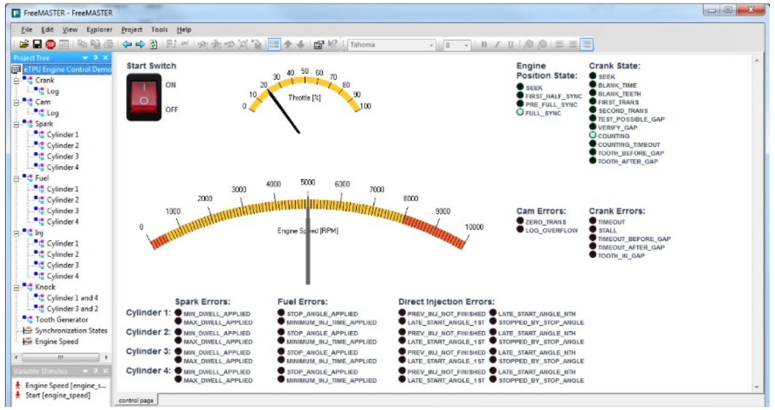
Rapid Sensors Evaluation Tool



Drone Demo



Airbag Reference Design



eTPU Engine Control

S32K Motor Control How To – Part 1

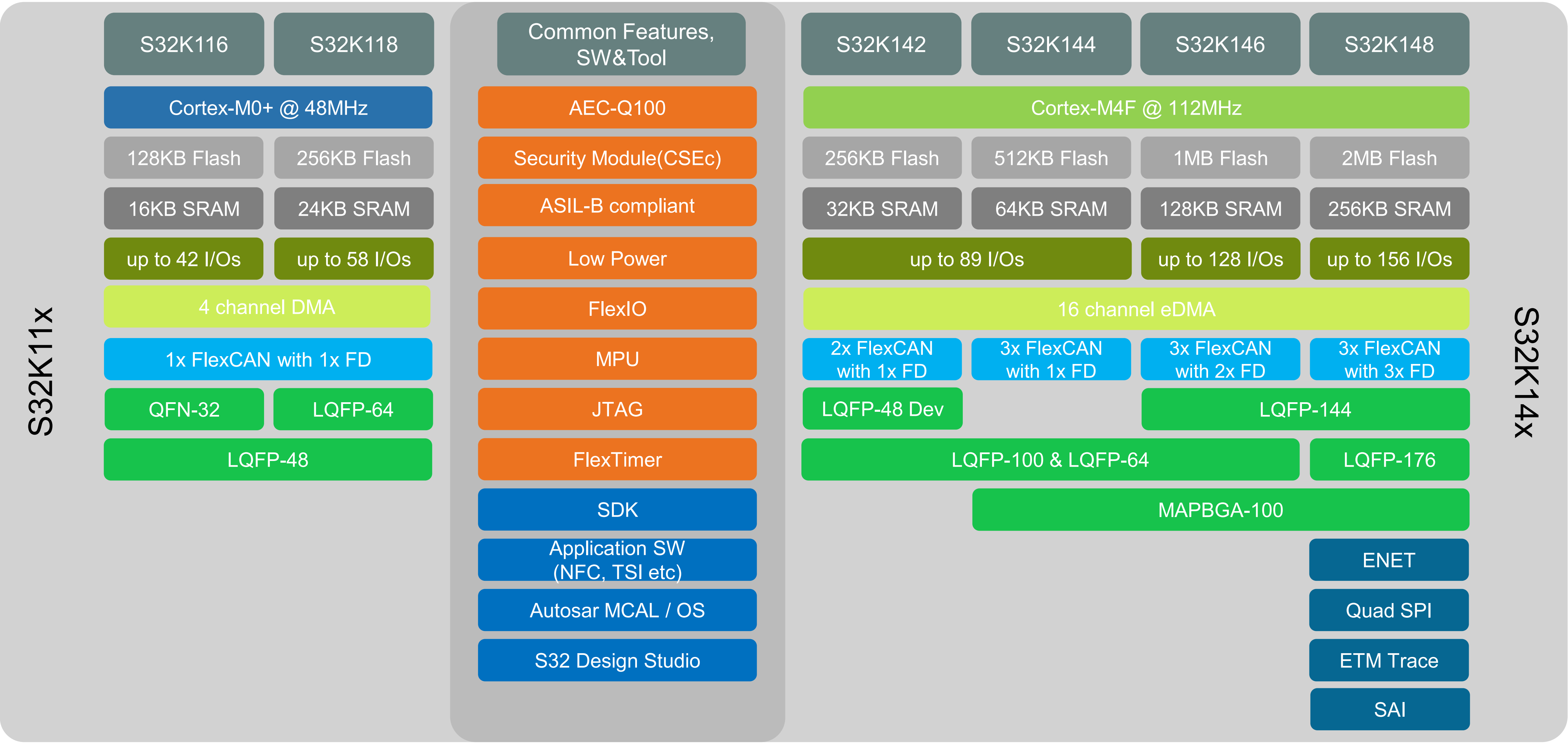
Agenda

- S32K Solutions
- S32K Motor Control Specific Modules
  - Clock
  - FTM/PWM
  - PDB
  - ADC
- FTM-PDB-ADC Synchronization

S32K Solutions

Solution, Scalability, Portfolio

S32K Product Family – S32K1**4**x and S32K1**1**x



S32K11x

S32K116	S32K118
Cortex-M0+ @ 48MHz	
128KB Flash	256KB Flash
16KB SRAM	24KB SRAM
up to 42 I/Os	up to 58 I/Os
4 channel DMA	
1x FlexCAN with 1x FD	
QFN-32	LQFP-64
LQFP-48	

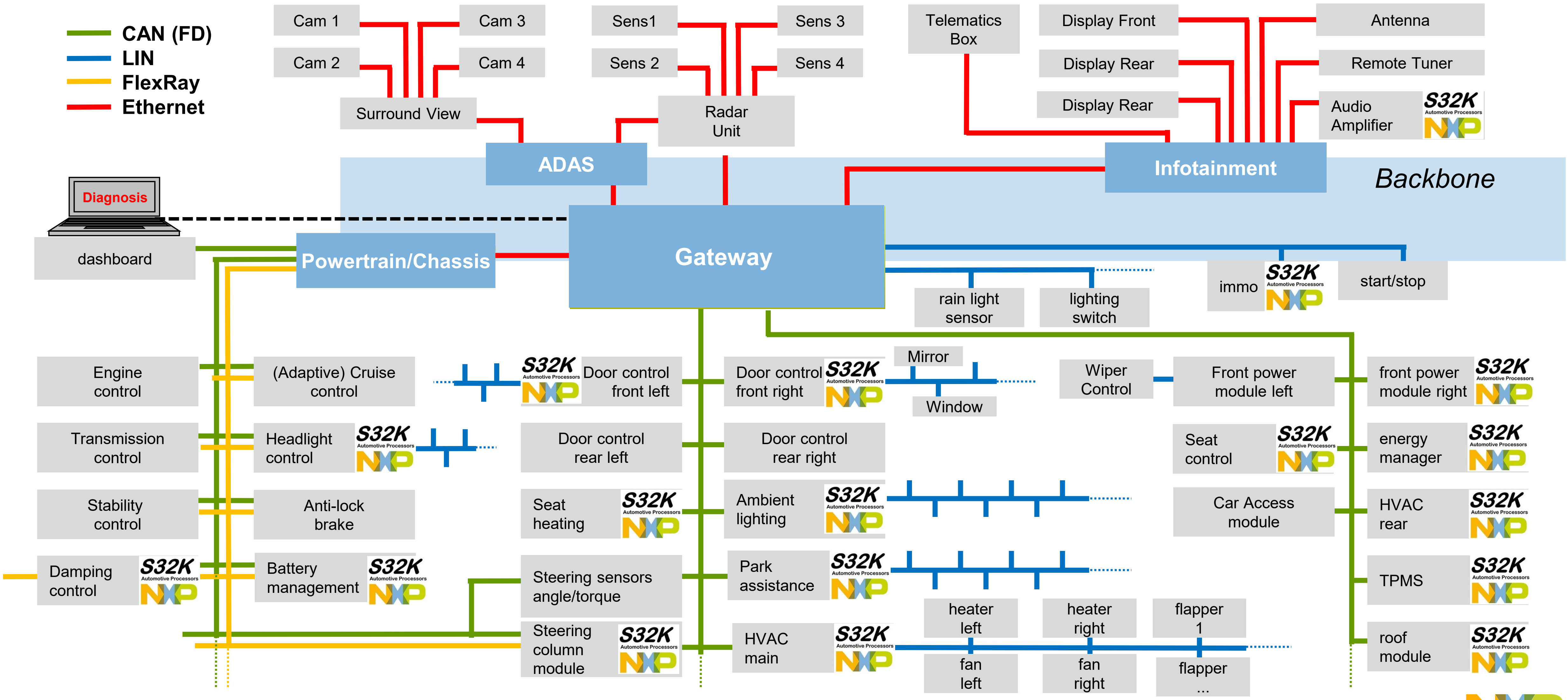
Common Features, SW&Tool

- AEC-Q100
- Security Module(CSEc)
- ASIL-B compliant
- Low Power
- FlexIO
- MPU
- JTAG
- FlexTimer
- SDK
- Application SW (NFC, TSI etc)
- Autosar MCAL / OS
- S32 Design Studio

S32K14x

S32K142	S32K144	S32K146	S32K148
Cortex-M4F @ 112MHz			
256KB Flash	512KB Flash	1MB Flash	2MB Flash
32KB SRAM	64KB SRAM	128KB SRAM	256KB SRAM
up to 89 I/Os		up to 128 I/Os	up to 156 I/Os
16 channel eDMA			
2x FlexCAN with 1x FD	3x FlexCAN with 1x FD	3x FlexCAN with 2x FD	3x FlexCAN with 3x FD
LQFP-48 Dev		LQFP-144	
LQFP-100 & LQFP-64			LQFP-176
	MAPBGA-100		
			ENET
			Quad SPI
			ETM Trace
			SAI

S32K – Solutions for Automotive Edge Nodes



S32K144 – Block Diagram

High performance

- ARM Cortex M4F up to 112MHz w FPU
- eDMA from 57xxx family

Software Friendly Architecture

- High RAM to Flash ratio
- Independent CPU and peripheral clocking
- 48MHz 1% IRC – no PLL init required in LP
- Registers maintained in all modes
- Programmable triggers for ADC → no SW delay counters or extra interrupts

Functional safety

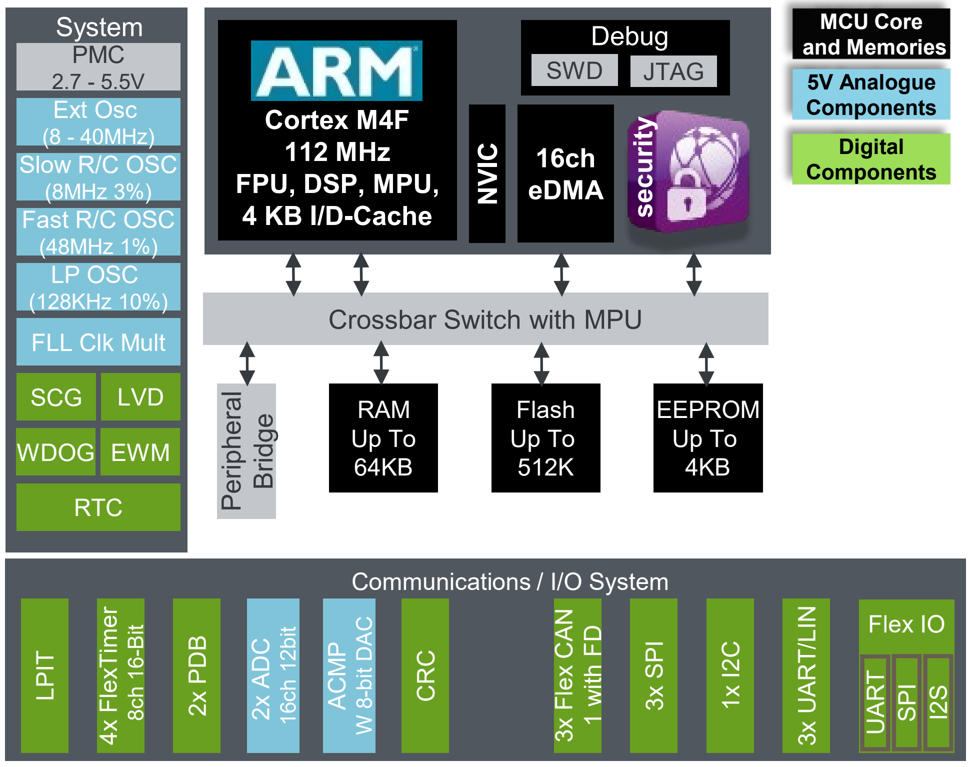
- ISO26262 support for ASIL B or higher
- Memory Protection Unit
- ECC on Flash/Dataflash and RAM
- Independent internal OSC for Watchdog
- Diversity between ADC and ACMP
- Diversity between SPI/SCI and FlexIO
- Core self test libraries
- Scalable LVD protection
- CRC

Low power

- Low leakage technology
- Multiple VLP modes and IRC combos
- Wake-up on analog thresholds

Security

- CSEc (SHE-spec)



Packages & IO

- Open-drain for 3.3 V and hi-drive pins
- Powered ESD protection
- Packages: 100 BGA, 64 LQFP, 100 LQFP

Operating Characteristics

- Voltage range: 2.7V to 5.5V
- Temperature (ambient): -40°C to +125°C

Memories

- 512KB Flash memory
- 64kB SRAM
- 4kB EEPROM

Communication Interfaces

- 3x LPUART
- 1x LPSPI
- 1x LPI2C
- 3x FlexCAN
- 1x FlexIO

(configurable for UART, SPI, I2C & I2S)

Analog Peripherals

- 2x 12bit ADC w/ up to 16 channels
- 1x CMP w/ 8bit DAC

Timers Peripherals

- 4x FTM with 8 channels
- 2x PDB with 2 channels
- Low Power Interrupt Timer (LPIT)
- Low Power Timer (LPTMR)
- Real Time Clock (RTC)

S32K – Solution for Industrial

S32K MCUs deliver quality, reliability and safety for challenging environments found in industrial, automation, communications, transportation, medical and A&D applications.

Oil Rig Sensor

Signaling System

Fire Alarm

Construction/Harvesting machine engine management and motor control

Air Conditioning System

Fluid Pump

Circular Pump for Heating and Cooling Water Circuit

Alternator, Generator

Medical Pump

High Pressure Pump

Gate Closing

Robot Arm movement

Machinery Positioning

Central Locking System

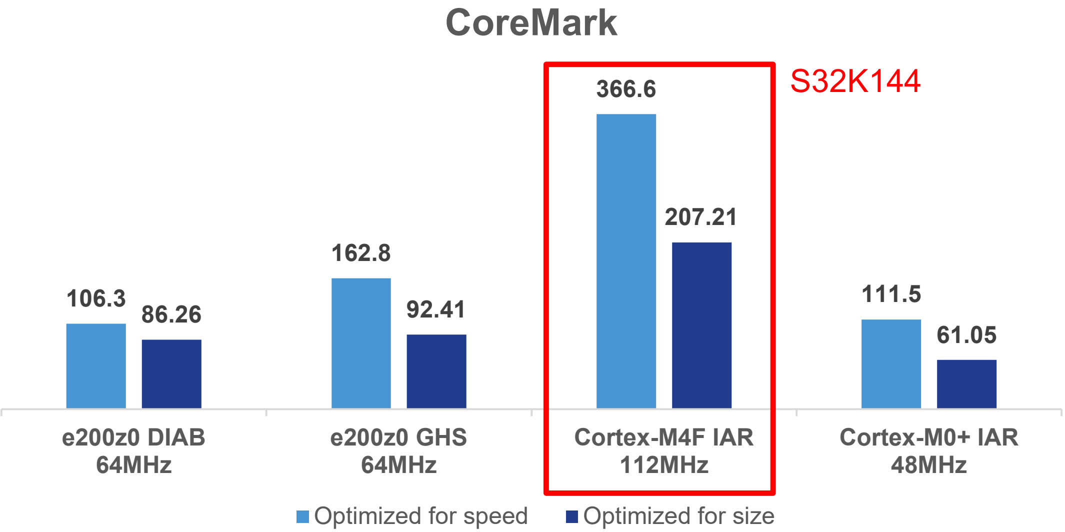
Lighting Control

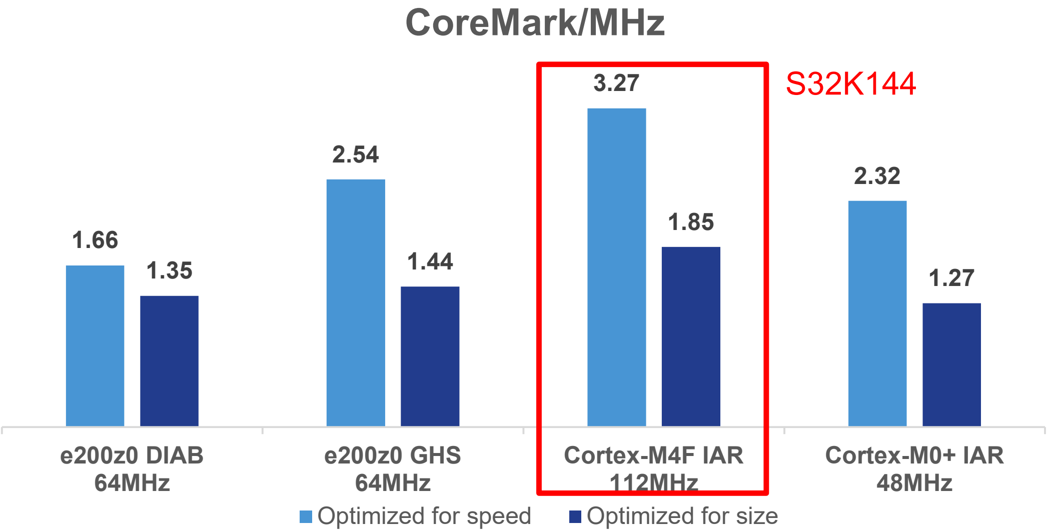
Elevator

Solar Inverter

Avionics

S32K – Performance



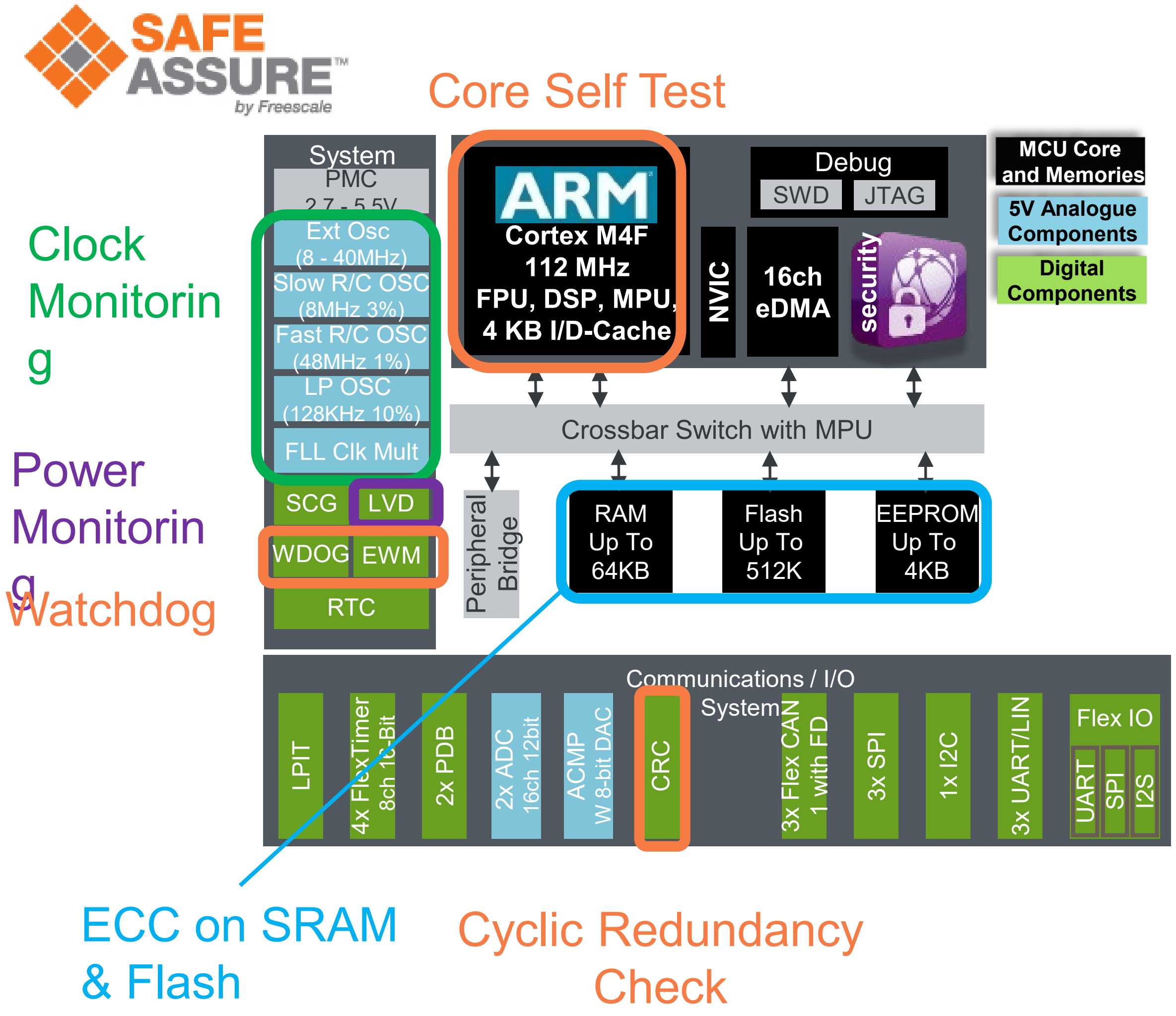


S32K – Low Power Performance

MCU/Core Type	Ta (C)	VLSP (uA)	VLPR (mA)	Stop 1 (mA)	Run (mA)
S32K116/M0	25 (typ)	26	1.9	7	tbd
S32K118/M0	25 (typ)	26	1.9	7	tbd
S32K142/M4	25 (typ)	29	1.9	10	tbd
S32K144/M4	25 (typ)	29.8	1.25	7	39.6
	105 (typ)	256.3	1.48	7.8	40.5
	125 (max)	1492	2.71	12.2	46
S32K146/M4	25 (typ)	40	5	15	tbd
S32K148/M4	25 (typ)	40	5	15	tbd

S32K – ASIL-B Functional Safety

- Safety Hardware
  - Power supplies
  - Clocks generation
  - Core platform (core, DMA, cache …)
  - Busses - XBAR
  - Memories – NVM, SRAM
- Safety Process
  - ISO 26262 development process
- Safety Support
  - FMEDA
  - Safety manual
  - Technical support
- Safety Software
  - S32K core self-test SW,



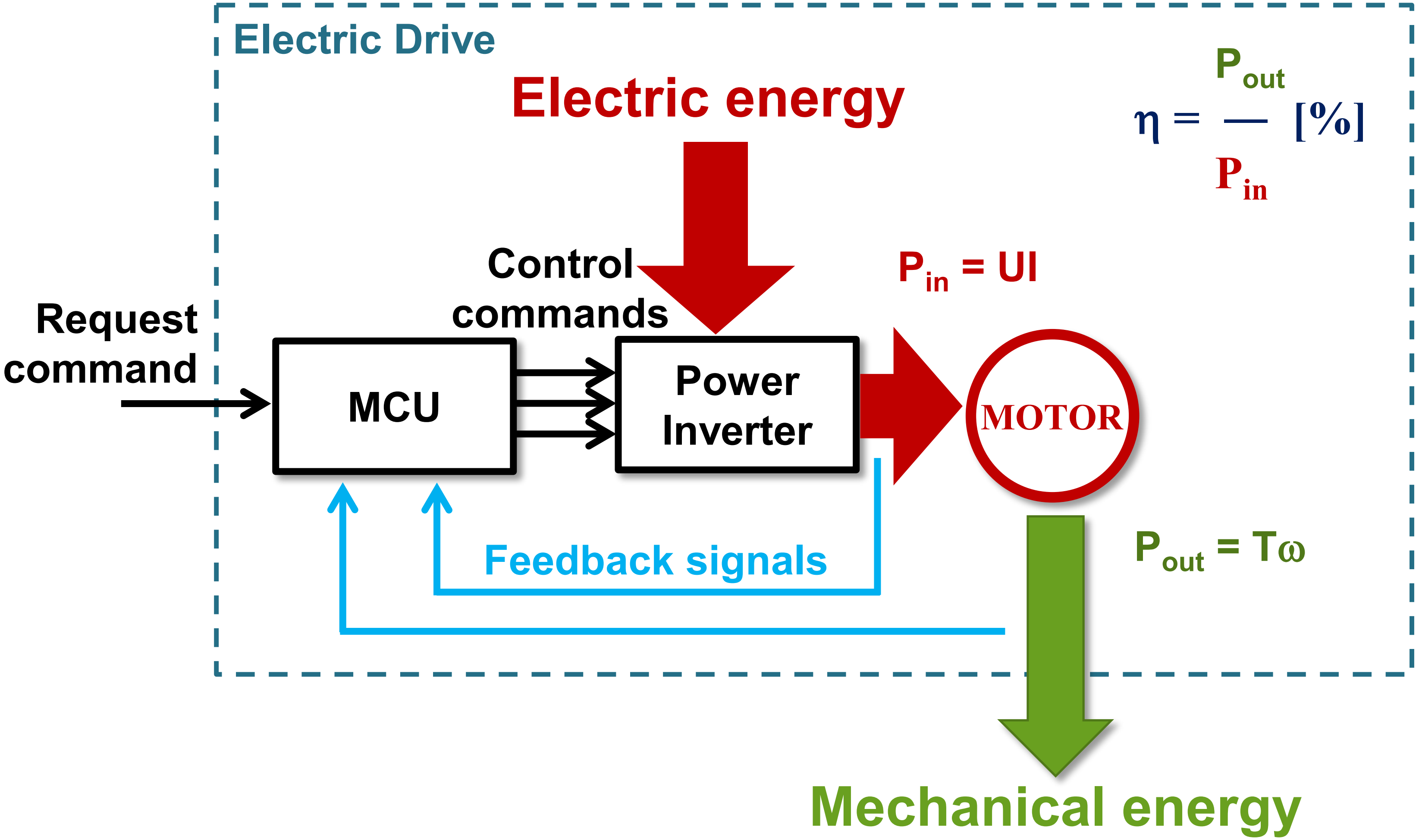
S32K Functional Safety SW

- Cortex-M Core Self-Test Library: Structural Core Self-Test Library (SCST) is a safety measure against permanent faults in the cores
- Developed for detecting hardware permanent faults in a core by means of executing machine op-codes with fixed set of operands and comparing their execution results
- This library is considered as Safety Element out of Context and was developed according to **ASIL B**
- SCST library provides tests to achieve the claimed diagnostic coverage (analytically estimated)

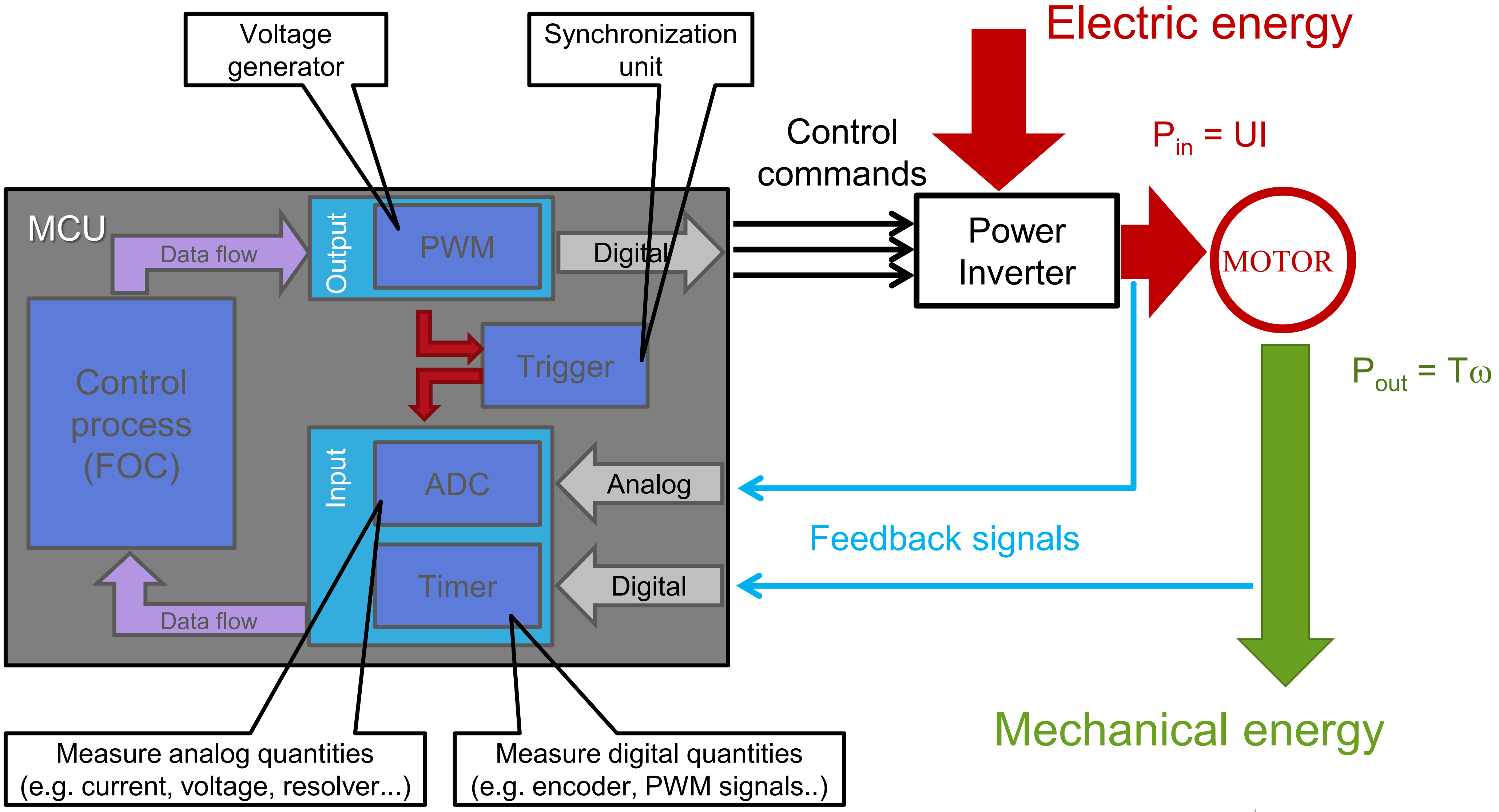
S32K Motor Control Specific Modules

FTM, PDB, ADC, GPIO

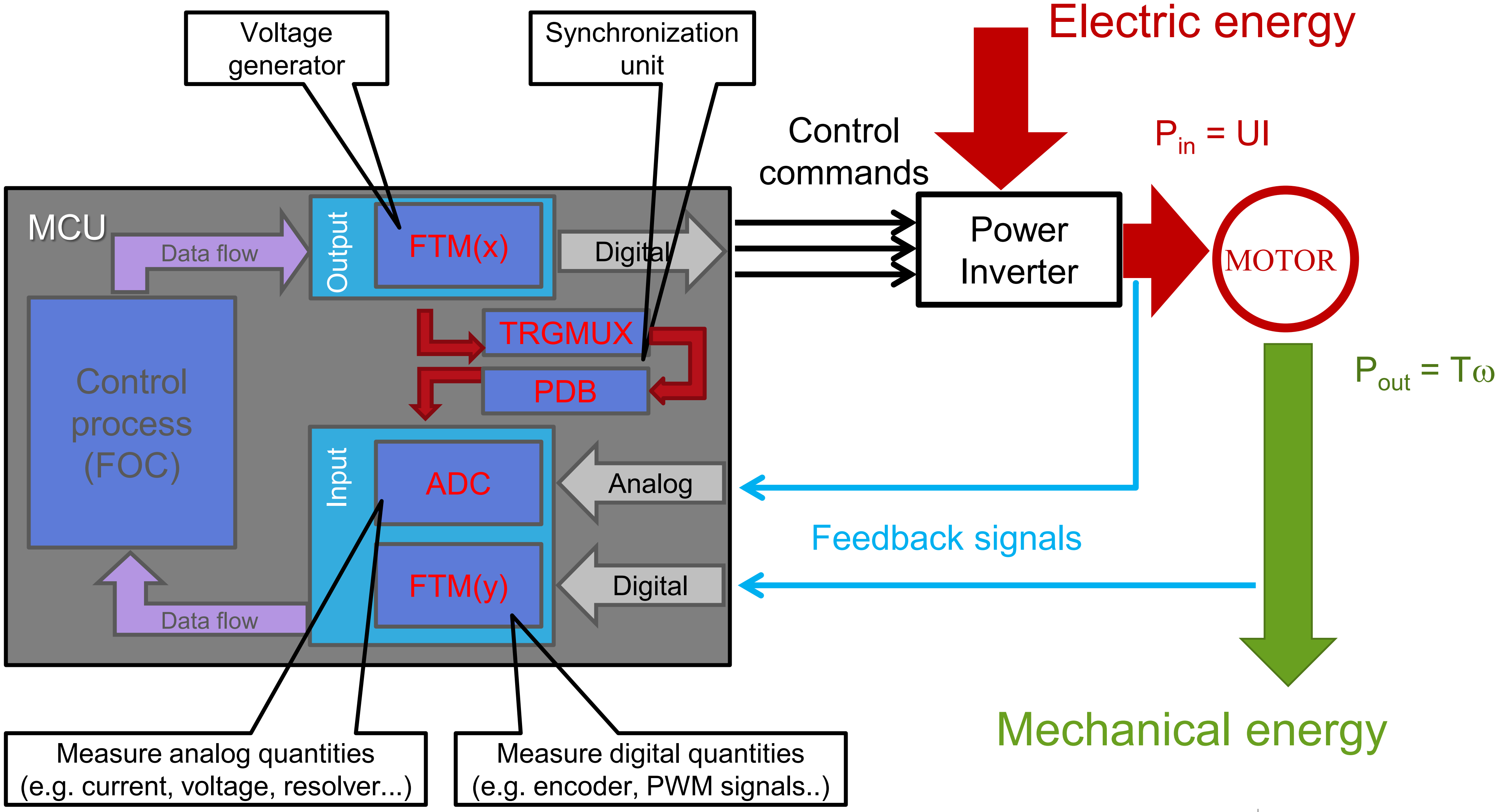
Motor Control Drive Concept



Motor Control Loop Implementation – Generic



Motor Control Loop Implementation on S32K



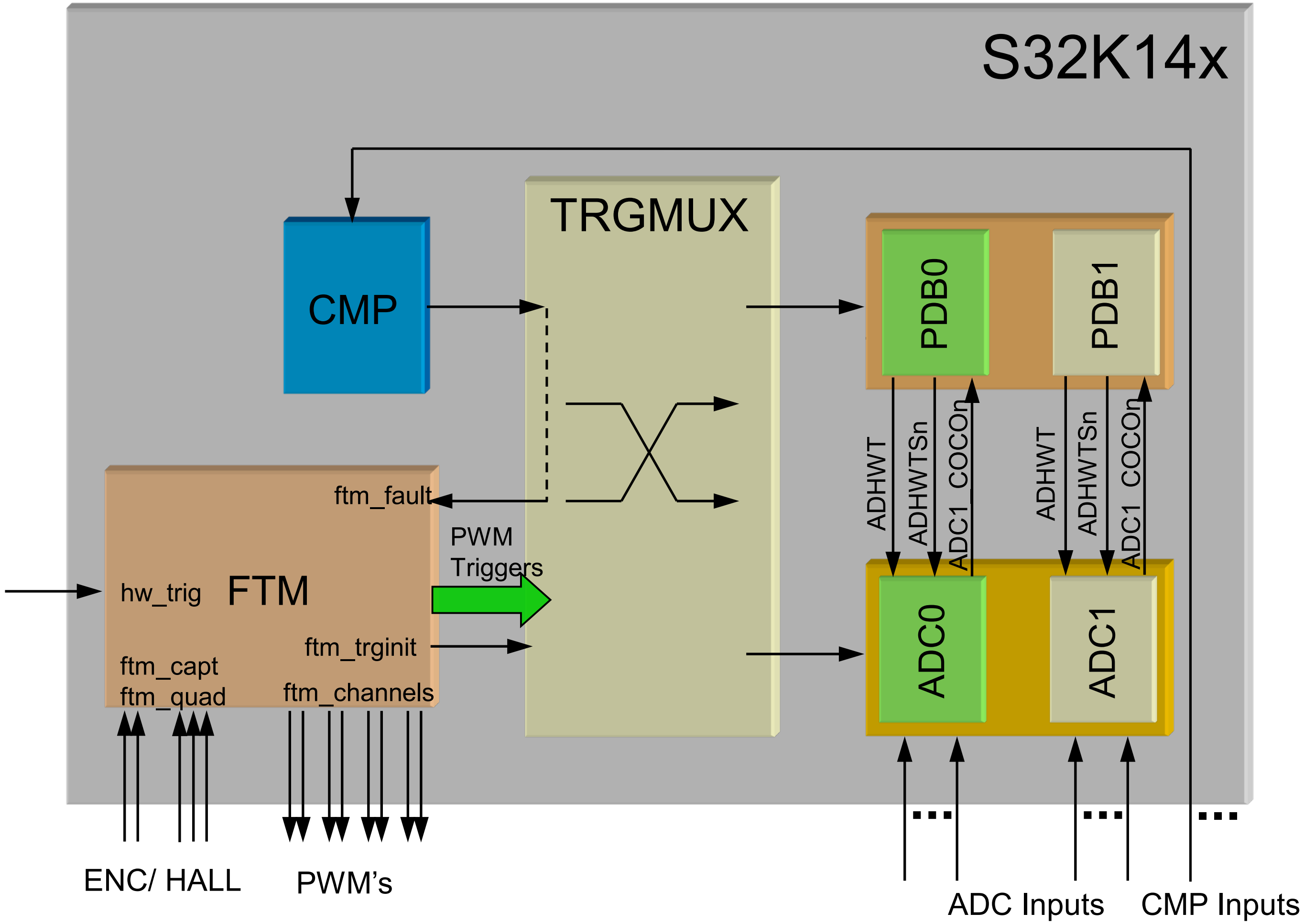
S32K144 Motor Control Specific Modules

CMP

- In MC as a fault protection unit
- Up to 8 channels some shared with ADC channels
- 8bit internal DAC

FTM0/1/2/3

- Various PWM modes
- Sync of double buffered registers
- Double buffered registers with various sync schemes
- Fault control
- SW Control & Masking
- Triggers generator for PDB or directly for ADC



TRGMUX

- Each peripheral has 32-bit trigger control register
- Each control register support up to 4 triggers
- Each trigger can be selected from up to 64 inputs.

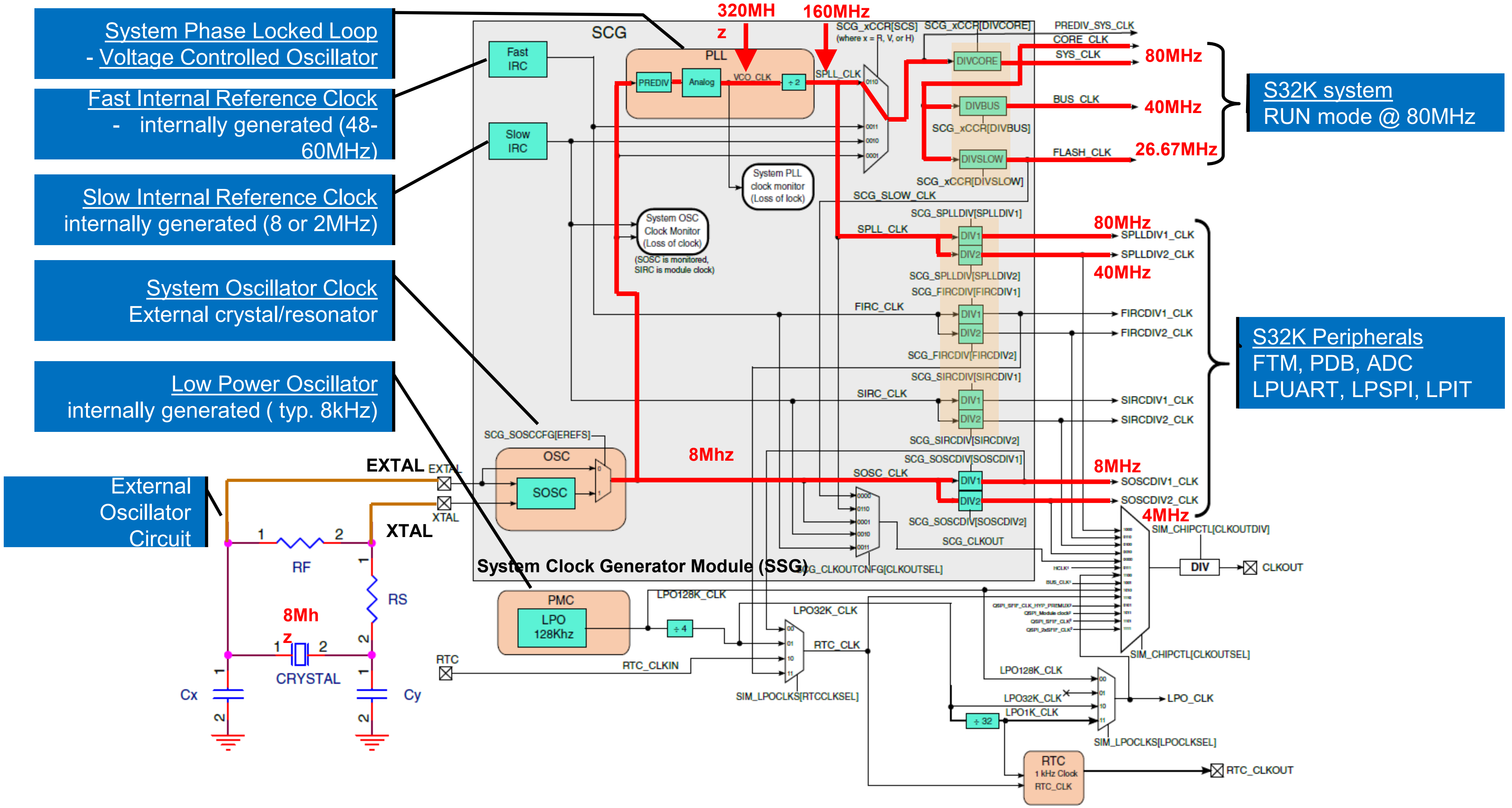
PDB0/1

- 16bit delay and triggering unit for ADC
- Two channels and each channel has 8 pre-triggers
- Back-to-back operation
- DMA support

ADC0/1

- 12bit resolution
- Up to 16 channels some of them interleaved & shared with analog CMP
- HW average function

S32K144 Clocking Diagram



System Phase Locked Loop
- Voltage Controlled Oscillator

Fast Internal Reference Clock
- internally generated (48-60MHz)

Slow Internal Reference Clock
internally generated (8 or 2MHz)

System Oscillator Clock
External crystal/resonator

Low Power Oscillator
internally generated (typ. 8kHz)

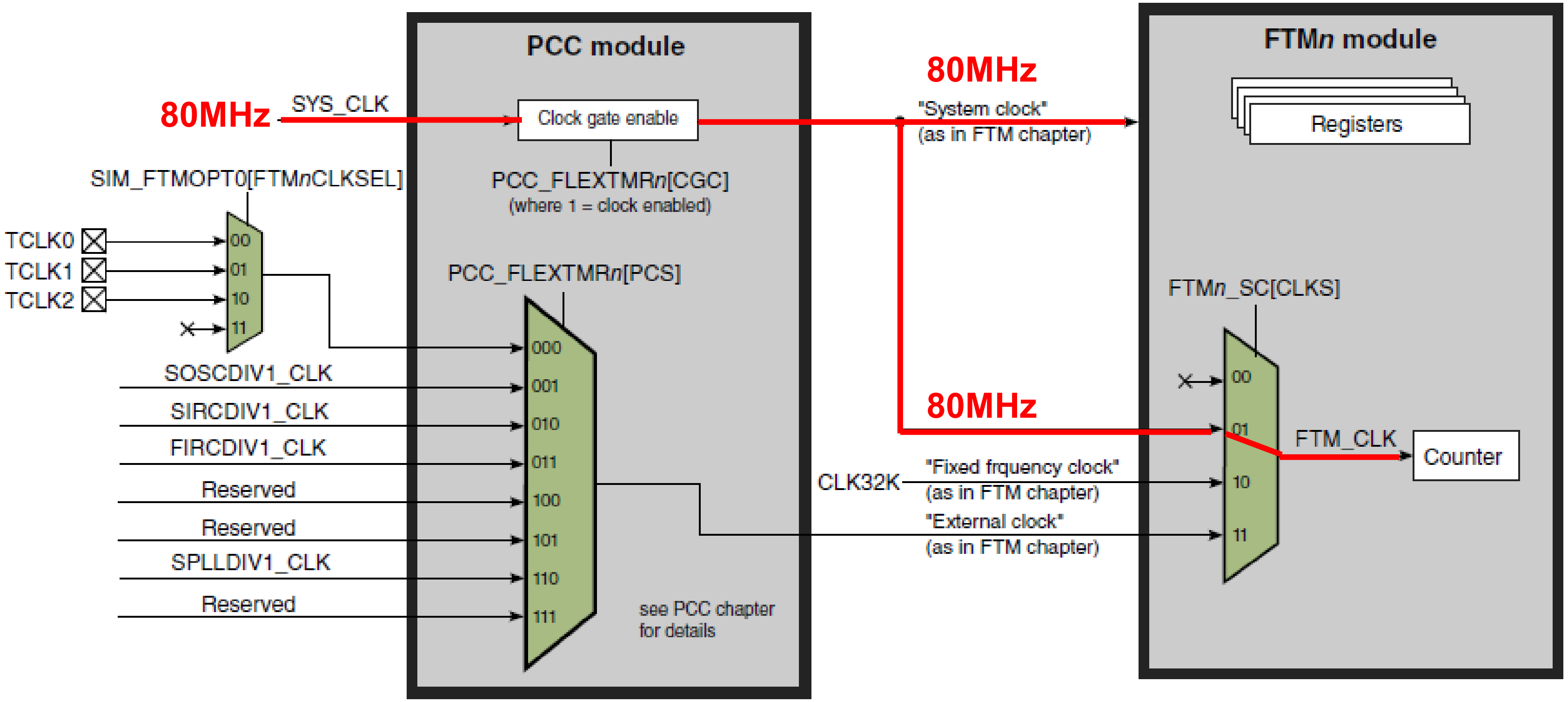
External Oscillator Circuit

S32K system
RUN mode @ 80MHz

S32K Peripherals
FTM, PDB, ADC
LPUART, LPSPI, LPIT

S32K144 Clock Source of FlexTimer

- Clock selection in Peripheral Clock Controller (PCC) – PCC_FTMn register
- FTM module needs two clock sources:
  - Module clock
  - Clock for FTM counter
- Source clock for FTM counter:
  - External clock source
  - Divided OSC
  - Divided FIRC (48MHz-60MHz)
  - Divided SIRC (8 or 2Mhz)
  - Divided PLL
  - **System clock (80MHz) to get the maximum PWM resolution**

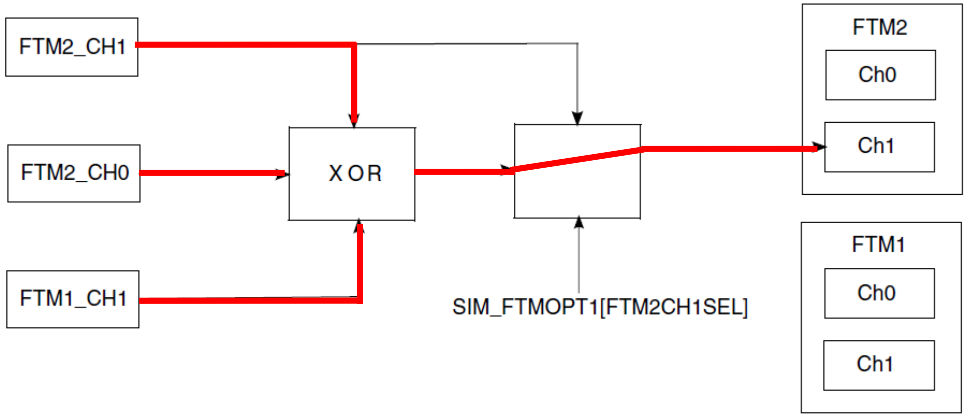


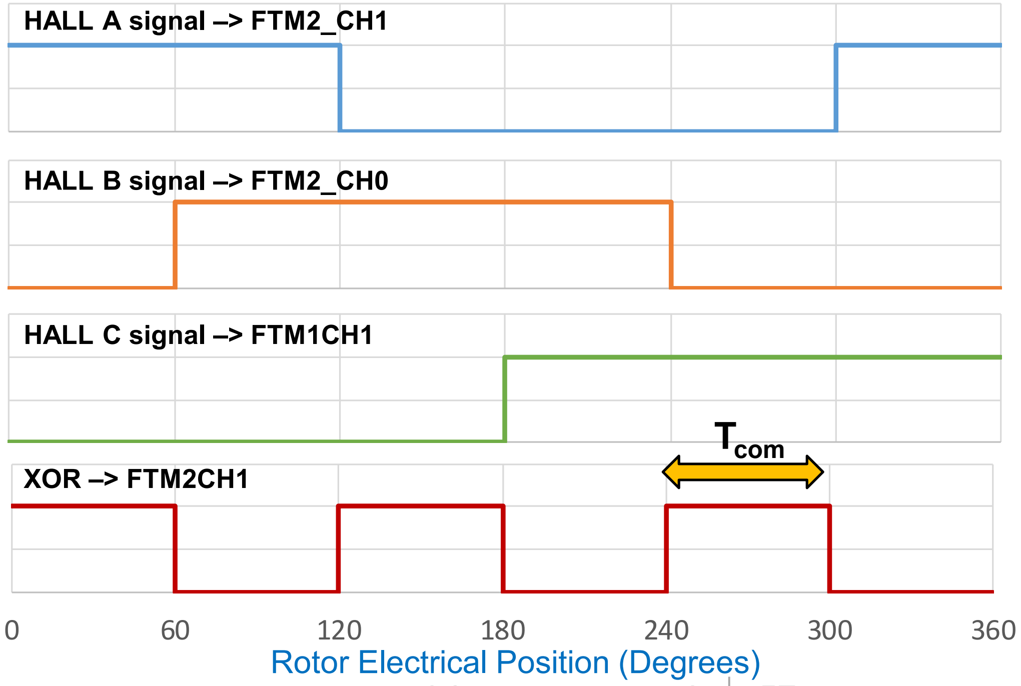
S32K144 FlexTimer Features

- **4 x FTM, with 8 channels** (inputs/outputs)
- FTM source clock is selectable with prescaler divide-by 1, 2, 4, 8, 16, 32, 64, or 128
- FTM has a 16-bit counter
- The **counting** can be **up** or **up-down**
- Each channel can be configured for input capture, output compare, or PWM generation
- New **combined mode to generate a PWM signal** (with independent control of both edges of PWM signal)
- **Complementary outputs, include the deadtime insertion**
- **Software control masking of PWM outputs**
- Up to **4 fault inputs** for global fault control
- The polarity of each channel is configurable
- The generation of an interrupt per channel input capture/compare, counter overflow, at fault condition
- **Synchronized loading of write buffered FTM registers**
- **Write protection for critical registers**
- Dual edge capture for pulse and period width measurement

S32K144 FlexTimer – Input Capture Mode

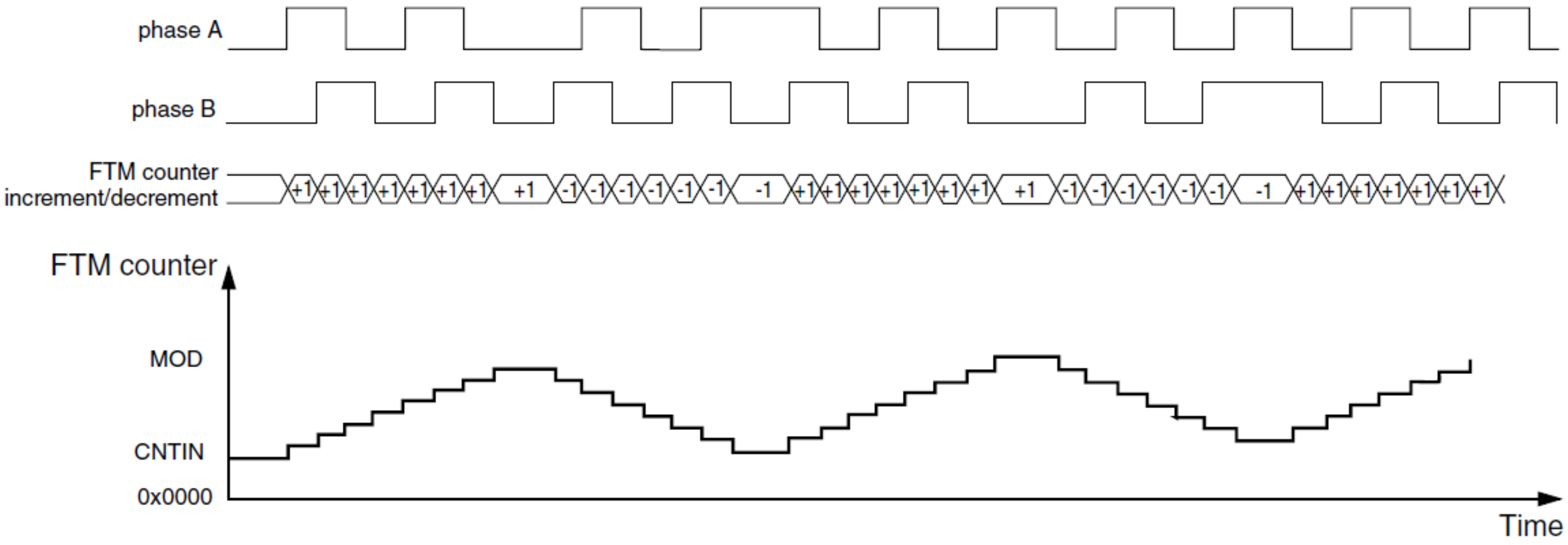
- **Single or Dual Edge Capture** mode for pulse or period width measurement
- Configurable capture events on rising, falling or both edges
- **Input filter** for some channel inputs
- When selected edge occurs, current FTM counter value is captured into the CnV register and interrupt is generated, if enabled
- **HALL sensor support** for position detection and speed calculation of an electric motor
- **FTM2_CH1** can be configures as normal input or the **XOR** of FTM2_CH0, FTM2_CH1 and FTM1_CH1 that will be applied to **FTM2_CH1**
- **Rotor speed** is calculated based on the captured time every rising/falling edge
- $\omega_{mech} = \frac{360°}{pT_{com}}$
- **Rotor position** is determined according to the HALL logic captured by **GPIO input**





S32K144 FlexTimer – Quadrature Decoder Mode

- Dedicated feature to process Phase A and Phase B signals generated by rotary or linear sensors to determine rotor/slider position and speed
- Phase A and Phase B input signals control FTM counter increment and decrement
- Phase A and Phase B input signals can be additionally filtered
- Two sub-modes available in quadrature mode:
  - Count and Direction Encoding mode
  - Phase A and Phase B Encoding mode
- 1024 pulses rotary encoder sensor MOD value is set to 4x1024=4096 and CNTIN is set to 0



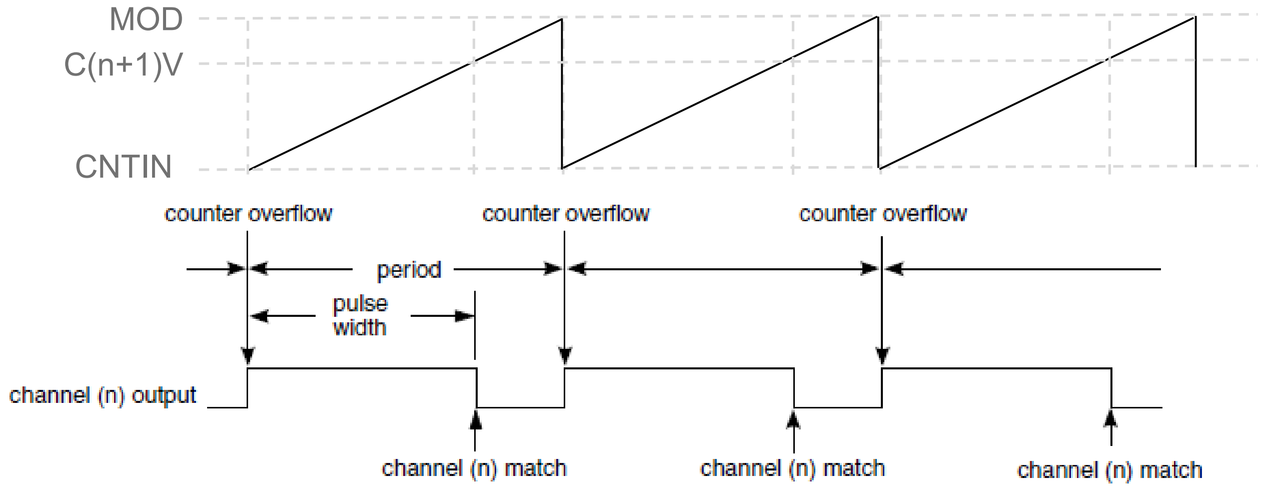
PWM Modes

- **Up counting** mode
  - For Edge-Align PWM
  - ELSnB:ELSnA bits define pulse polarity
  - **PWM period** = MOD - CNTIN
  - **Duty cycle** = CnV – CNTIN

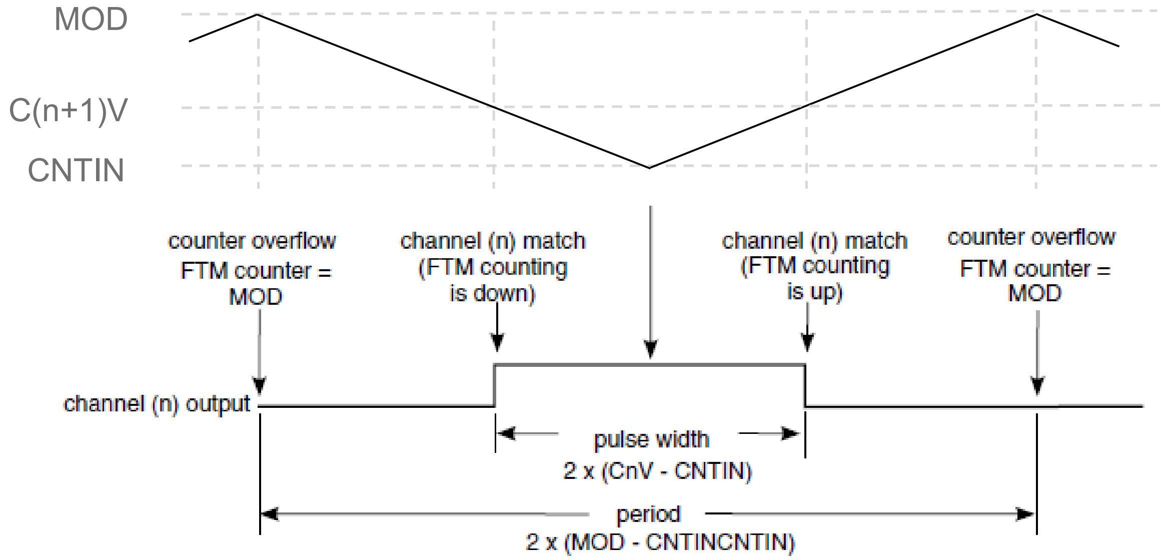
- **Up-down counting** mode
  - For Center-Align PWM
  - ELSnB:ELSnA bits define pulse polarity
  - **PWM period** = 2x (MOD-CNTIN)
  - **Duty cycle** = 2x (CnV – CNTIN)

- **In up counting mode** even (n) and odd channels (n+1) can work as a pair in **combine mode** & **complementary mode**

Up counting mode - Edge-Aligned PWM
ELSnB:ELSnA = 1:0



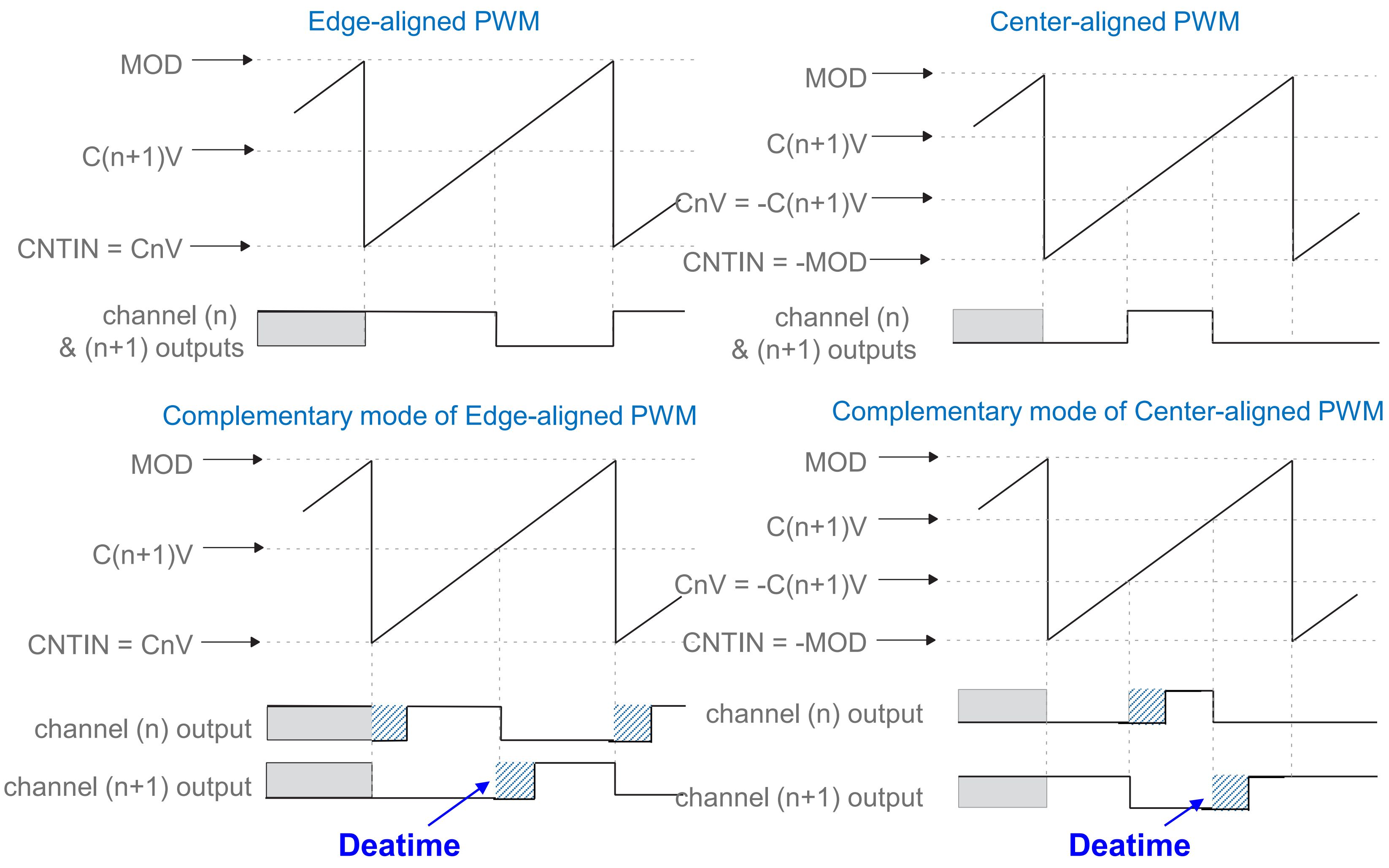
Up-down counting mode - Center-Aligned PWM
ELSnB:ELSnA = 1:0



MOD
C(n+1)V
CNTIN
counter overflow FTM counter = MOD
channel (n) match (FTM counting is down)
channel (n) match (FTM counting is up)
counter overflow FTM counter = MOD
channel (n) output
pulse width
2 x (CnV - CNTIN)
period
2 x (MOD - CNTINCNTIN)

PWM Modes (Cont'd)

- In **Combine mode** two FTM channels are combined together to define one PWM signal
  - An **even channel (n)** defines rising edge of PWM signal
  - An **adjacent odd channel (n+1)** defines falling edge of PWM signal
  - PWM period = MOD - CNTIN
  - Duty cycle = C(n+1)V - C(n)V
- **Complementary mode**
  - Channel (n+1) output is the inverse of the channel (n)
- **Deadtime insertion**
  - Clock source for deadtime delay is the FTM input clock divided by deadtime prescaler value DTPS (1, 4, 16)



Edge-aligned PWM

Center-aligned PWM

Complementary mode of Edge-aligned PWM

Complementary mode of Center-aligned PWM

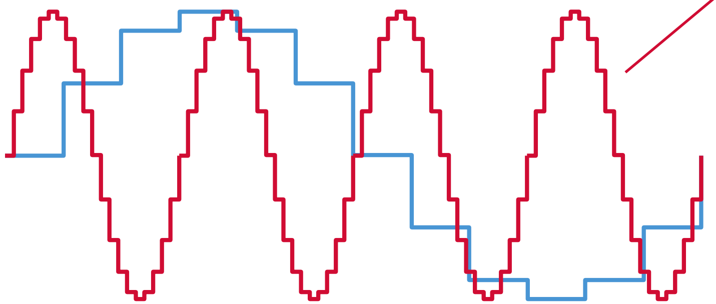
Advanced Peripherals Reduce Motor/Generator Size

- High Resolution PWM (Pulse Width Modulator) Peripheral is key to design powerful electric motors at reduced cost:
  - Motor/Generator to be small and powerful (not to carry unnecessary mass/space).
    shall be designed to work at high electric frequency (~1 kHz – high number of poles).
  - Generation of high electric frequency 3-phase system requires high resolution PWM, both in time and amplitude
- FTM ~ 12bit resolution at 80MHz input clock frequency

Air gap size × Sum of windings wire’s cross section

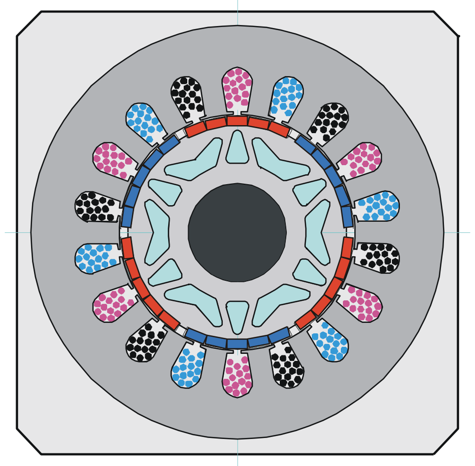
$$P \approx \frac{1}{\pi} \cdot \omega_e \cdot B_\delta \cdot A_{\delta_\Sigma} \cdot A_{Cu} \cdot \sigma_{Cu}$$

Sine-wave generation



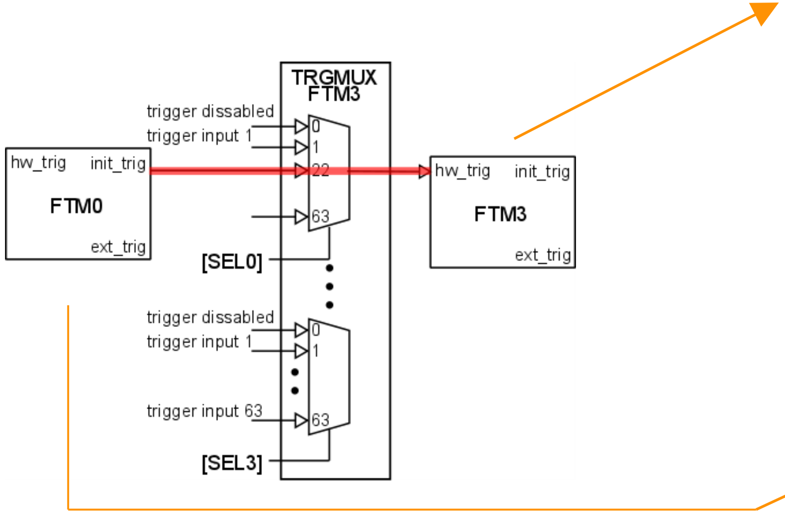
Low-resolution PWM

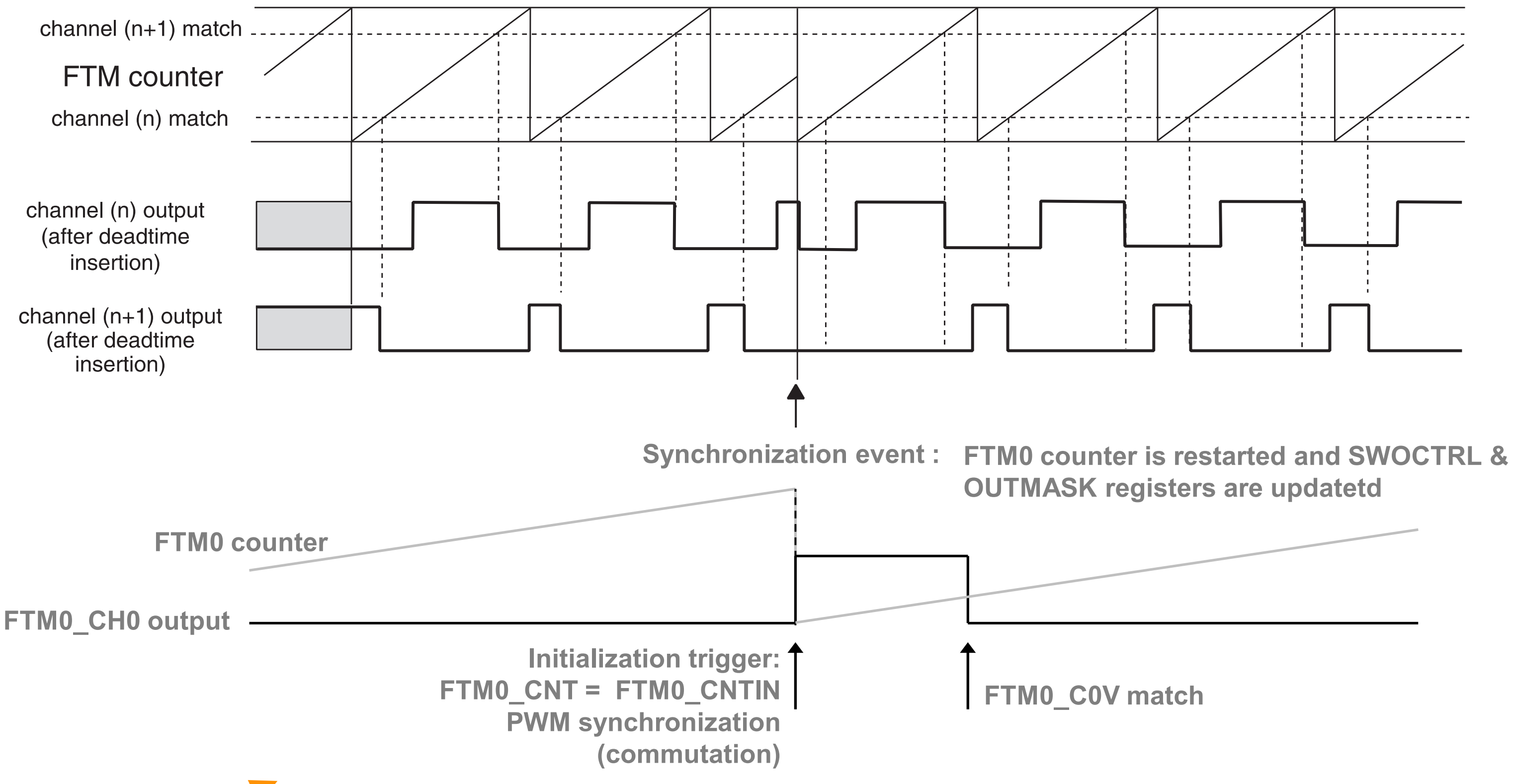
High-resolution PWM



S32K144 FlexTimer – PWM Synchronization

- PWM synchronization provides an opportunity to force FTM counter to the CNTIN value and opportunity to update MOD, CNTIN, CnV, OUTMASK, INVCTRL, SWOCTRL registers with their buffered value
- The commutation event timing depends on the rotor position and it is asynchronous to PWM (may occur anytime during the PWM cycle)
- FTM3 can be synchronized by FTM0 init trigger





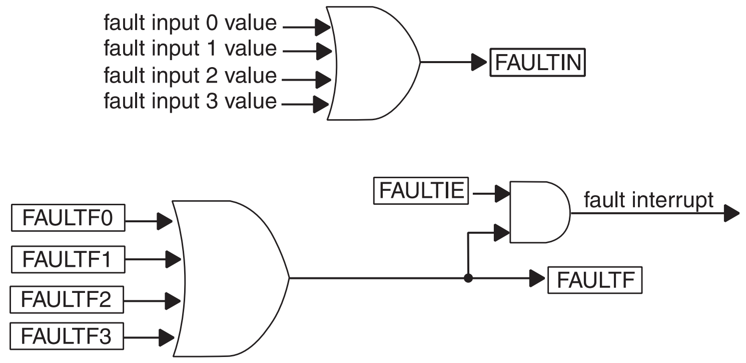
S32K144 FlexTimer – Fault Control

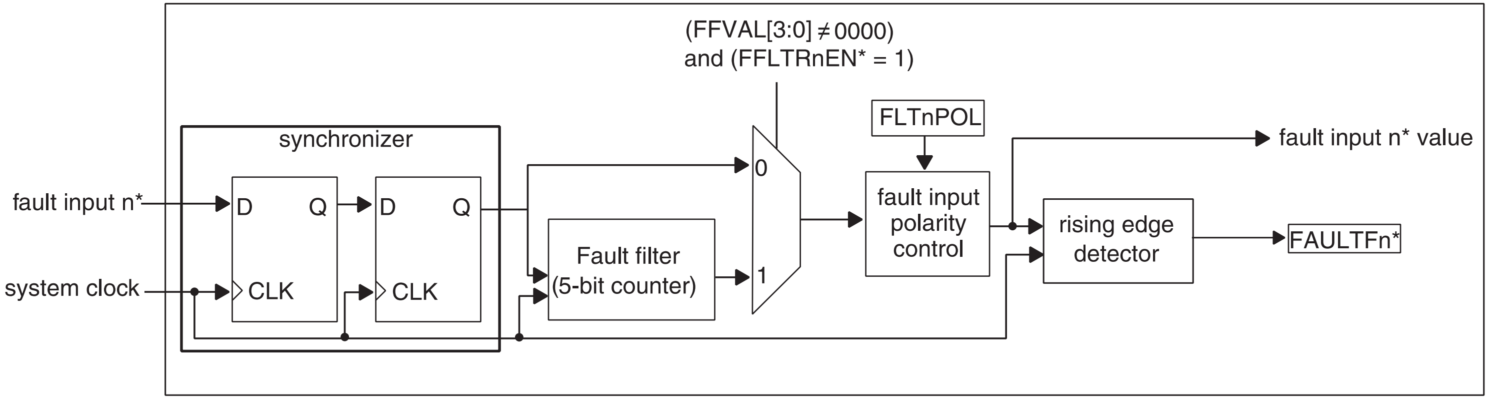
- To protect system in critical moments:

  over-current, under-voltage, over-temperature...
- 4 fault inputs OR-ed into single fault signal
- Fault signal disables all PWM outputs
- The polarity of a fault signal is user configurable
- All fault inputs have configurable dig. input filter
- Automatic or Manual fault clearing:

  - AFC – at new PWM cycle

  - MFC – at new PWM cycle, FAULTF has to be cleared

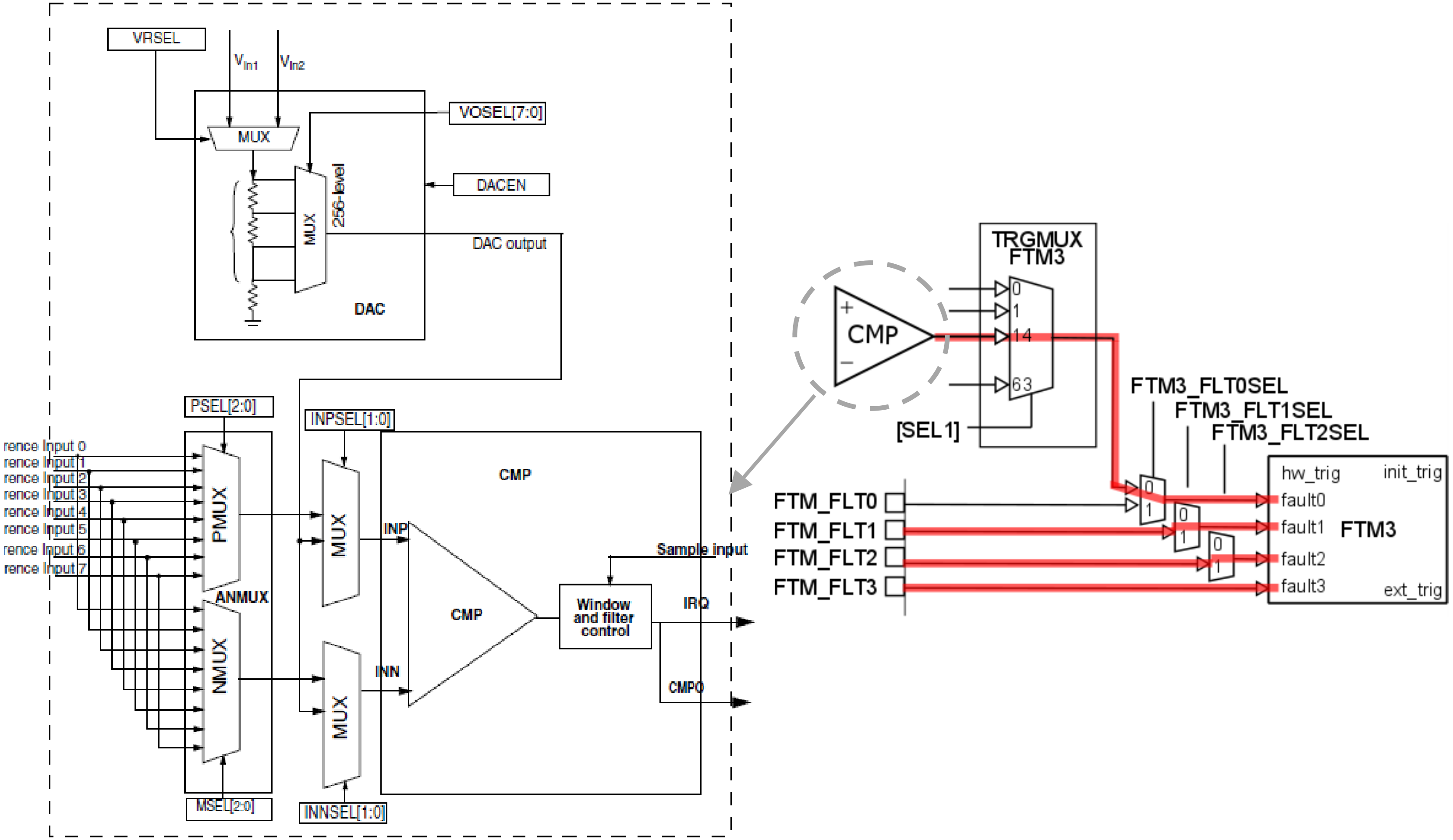




* where n = 3, 2, 1, 0

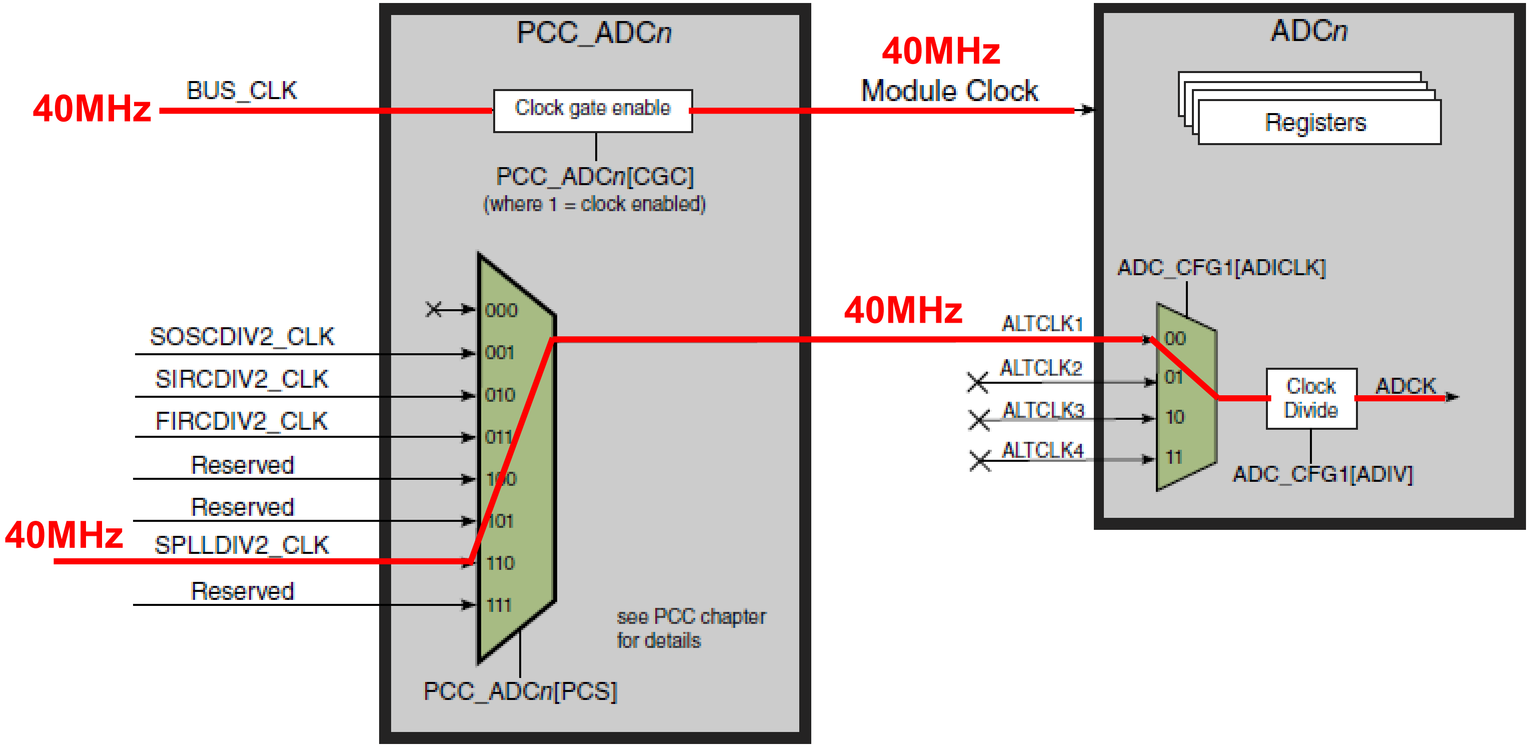
S32K144 Analog Comparator (CMP)

- CMP output can be used as a fault input signal of FTM module
- Up to 8 independently selectable channels for positive and negative comparator inputs
- Some channels are shared with ADC input pins
- Internal 8-bit DAC with range of reference voltage:
  - 3.9mV from 0 to 1V or
  - 19mV over full 5V range
- Configurable hysteresis Control
- CMP output :
  - Sampled
  - Digitally filtered
  - Windowed (via PDB or LPIT)



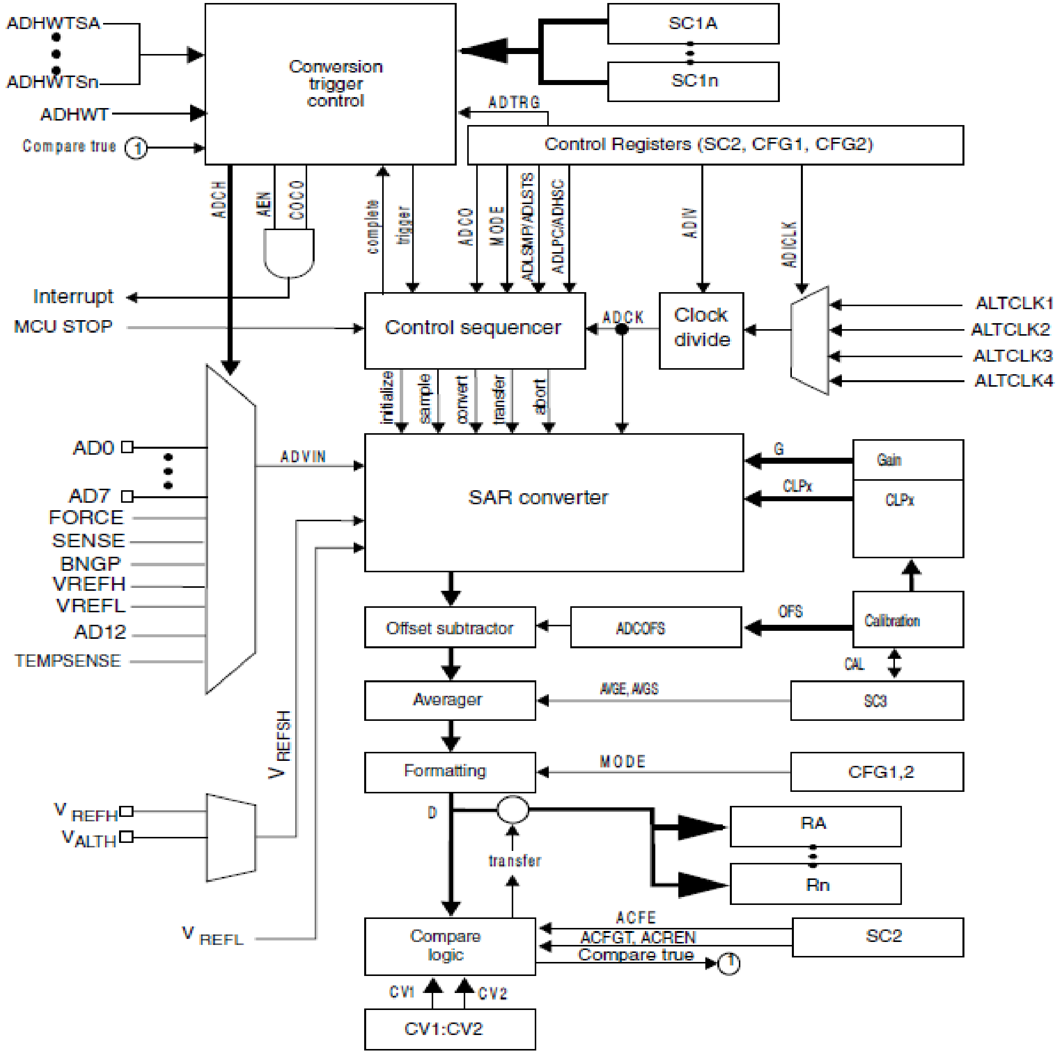
S32K144 Clock Source for ADC

- Clock selection in Peripheral Clock Controller (PCC) – PCC_ADCn register
- ADC module needs two clock sources:
  - Module clock
  - Asynchronous clock
- Asynchronous clock sources:
  - Divided SOSC (SOSCDIV2_CLK)
  - Divided FIRC (48MHz-60MHz)
  - Divided SIRC (SIRCDIV2_CLK)
  - **Divided SPLL (SPLLDIV2_CLK = 40MHz)**
- For short sample and maximum frequency 40 MHz T_{conv} = 1.25us



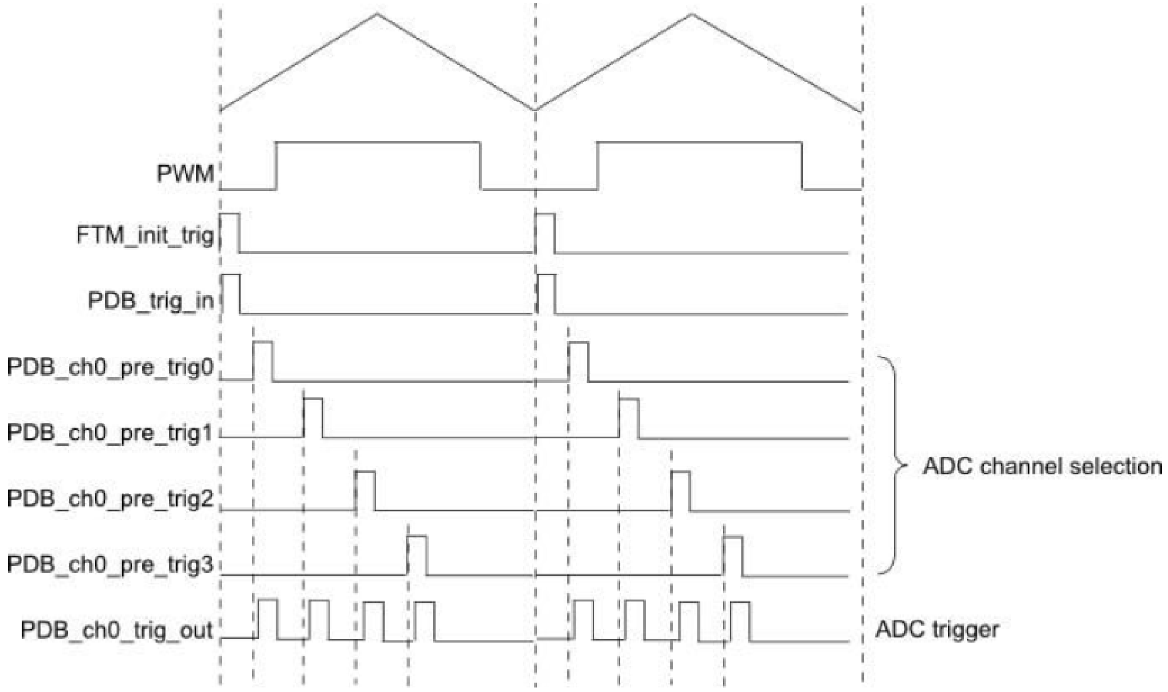
S32K144 ADC Block Diagram

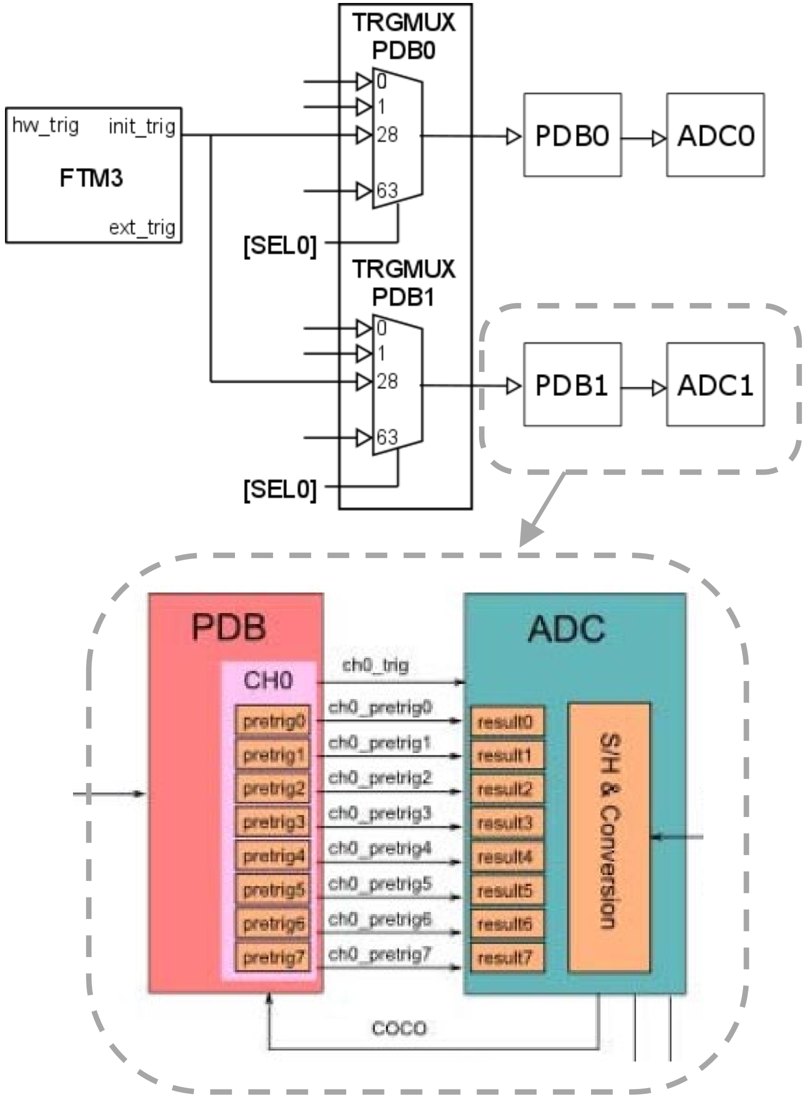
- Up to 16 single-ended external analog inputs some of them can be hardware interleaved:
  - ADC0_SE4 and ADC1_SE14
  - ADC0_SE5 and ADC1_SE15
  - ADC1_SE8 and ADC0_SE8
  - ADC1_SE9 and ADC0_SE9
- Input clock selectable from up to four sources (f_{ADCLK} = 50 MHz)
- Self-Calibration mode
- Single or continuous conversion
- Hardware average function
- Automatic compare with interrupt for less-than, greater-than or equal-to, within range, or out-of-range, programmable value
- Programmable sample time and conversion speed/power (For short sample and 56 MHz frequency T_{conv} = 0.89us)
- DMA support



S32K144 ADC Triggering by PDB

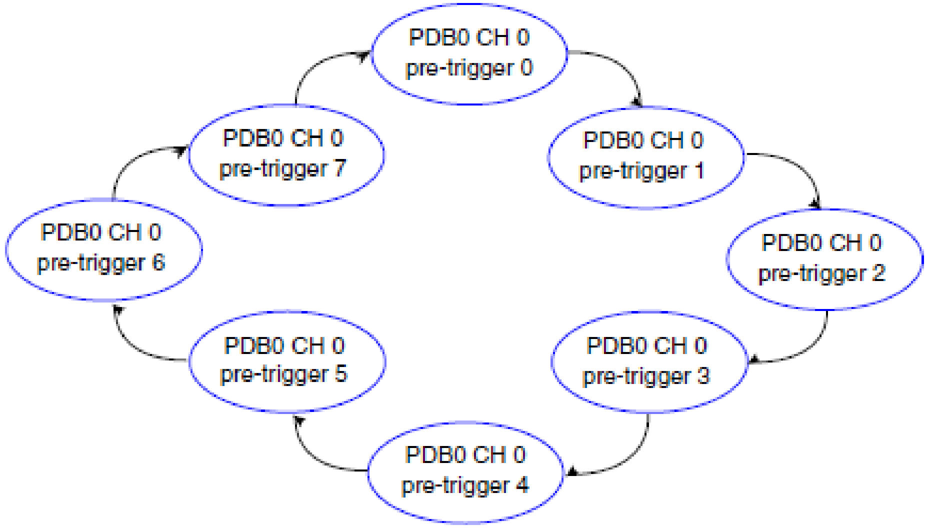
- ADC can be triggered by SW or HW
- Several hardware trigger sources: LPIT, CMP... routed through TRGMUX
- One ADC and one PDB work as one pair : PDB0-ADC0, PDB1-ADC1
- PDB0 pre-triggers will connect directly to ADC0 ADHWTS ports to control the channels
- The ADC0 coco signals are directly feedbacked to PDB0 to deactivate the PDB lock state



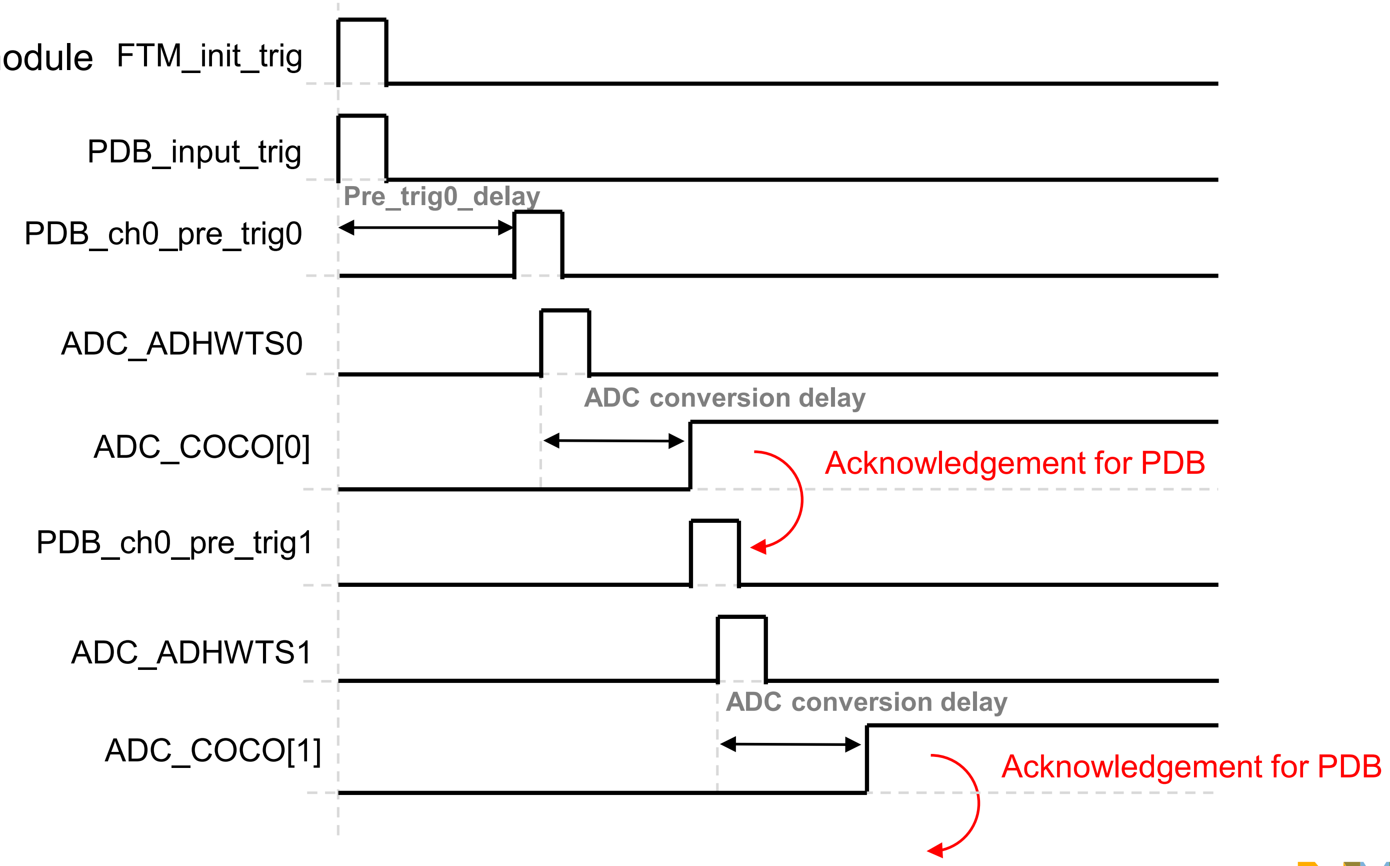


S32K144 ADC Triggering by PDB (Back-to-back)

- In **back-to-back** operation is enabled by CH*n*C1[BB[***m***]] bit and each ***m*** pre-trigger delay is ignored
- PDB acknowledgment = COCO[n]
- All acknowledgments are implemented in each PDB module as a ring

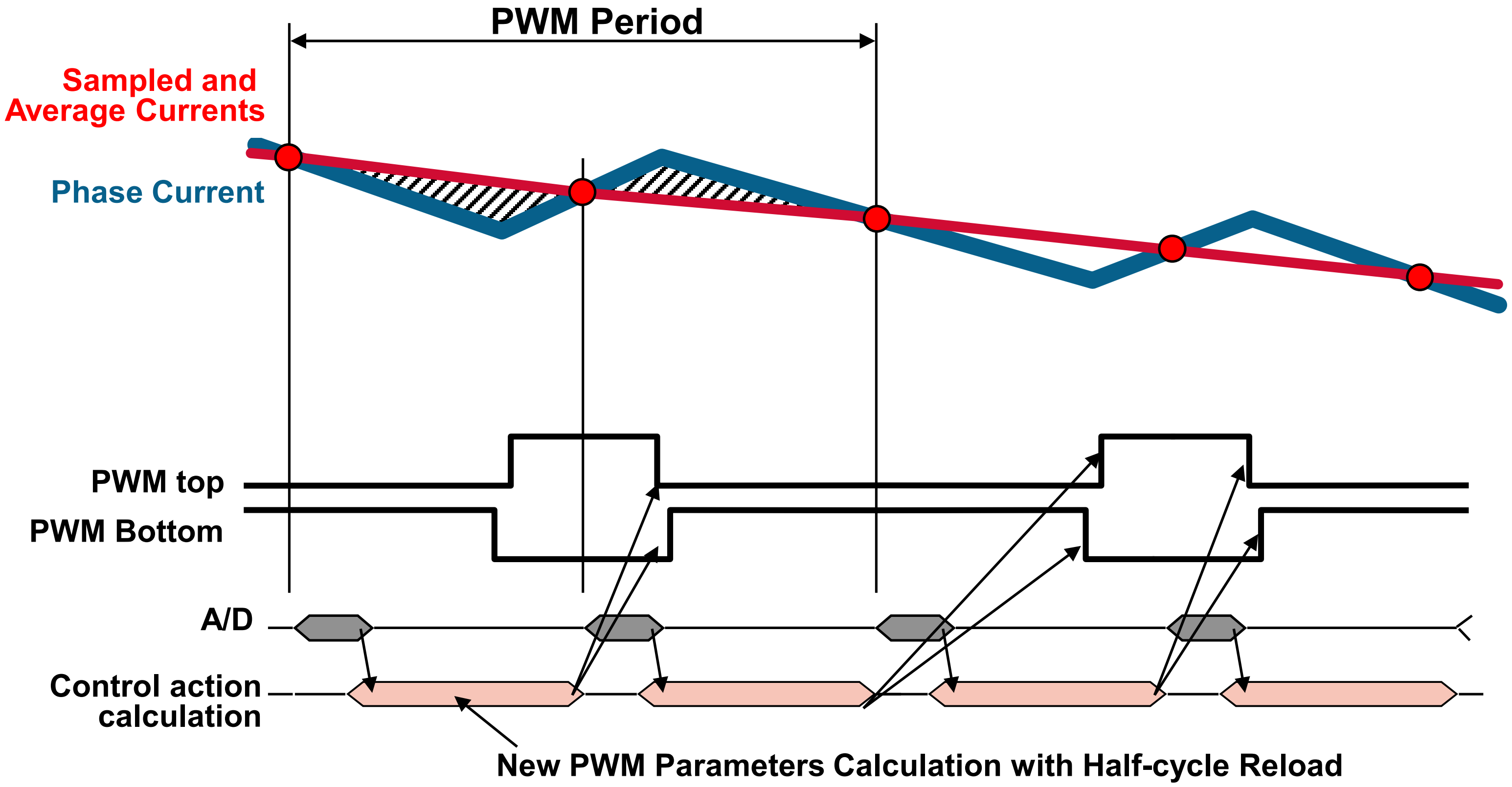


Back-to-Back Operation



S32K144 PDB Benefits

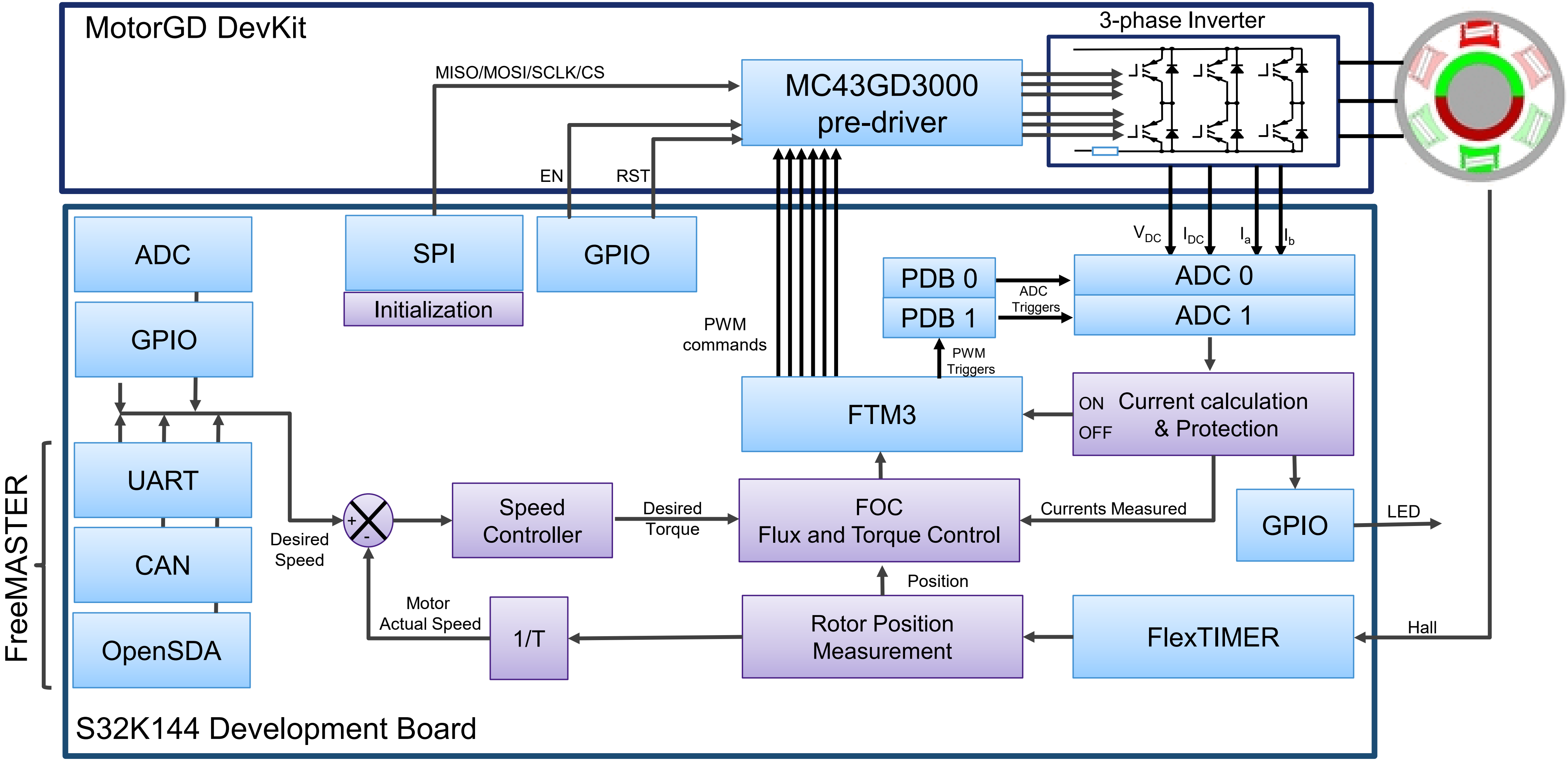
- ADC Sampling helps to filtering the measured current – anti-aliasing
- Noise free ADC sampling when the power switch is not acting


PWM Period
Sampled and Average Currents
Phase Current
PWM top
PWM Bottom
A/D
Control action calculation
New PWM Parameters Calculation with Half-cycle Reload

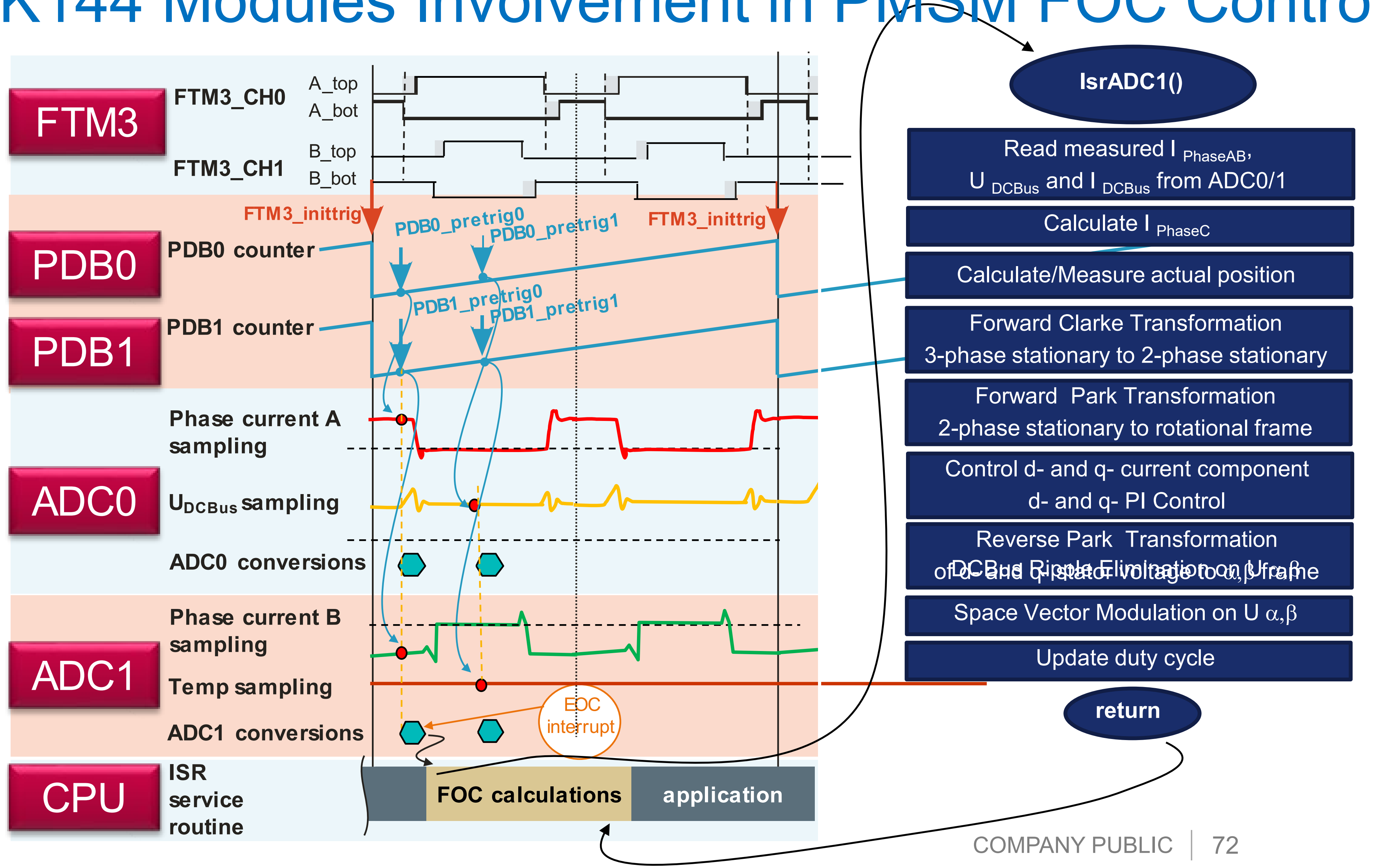
Practical FOC Implementation with NXP S32K Evaluation Boards

FTM-PDB-ADC Synchronization

PMSM FOC Block Diagram



S32K144 Modules Involvement in PMSM FOC Control Loop



IsrADC1()

Read measured I $_{PhaseAB}$, U $_{DCBus}$ and I $_{DCBus}$ from ADC0/1

Calculate I $_{PhaseC}$

Calculate/Measure actual position

Forward Clarke Transformation
3-phase stationary to 2-phase stationary

Forward Park Transformation
2-phase stationary to rotational frame

Control d- and q- current component
d- and q- PI Control

Reverse Park Transformation
of d- and q- stator voltage to α,β frame

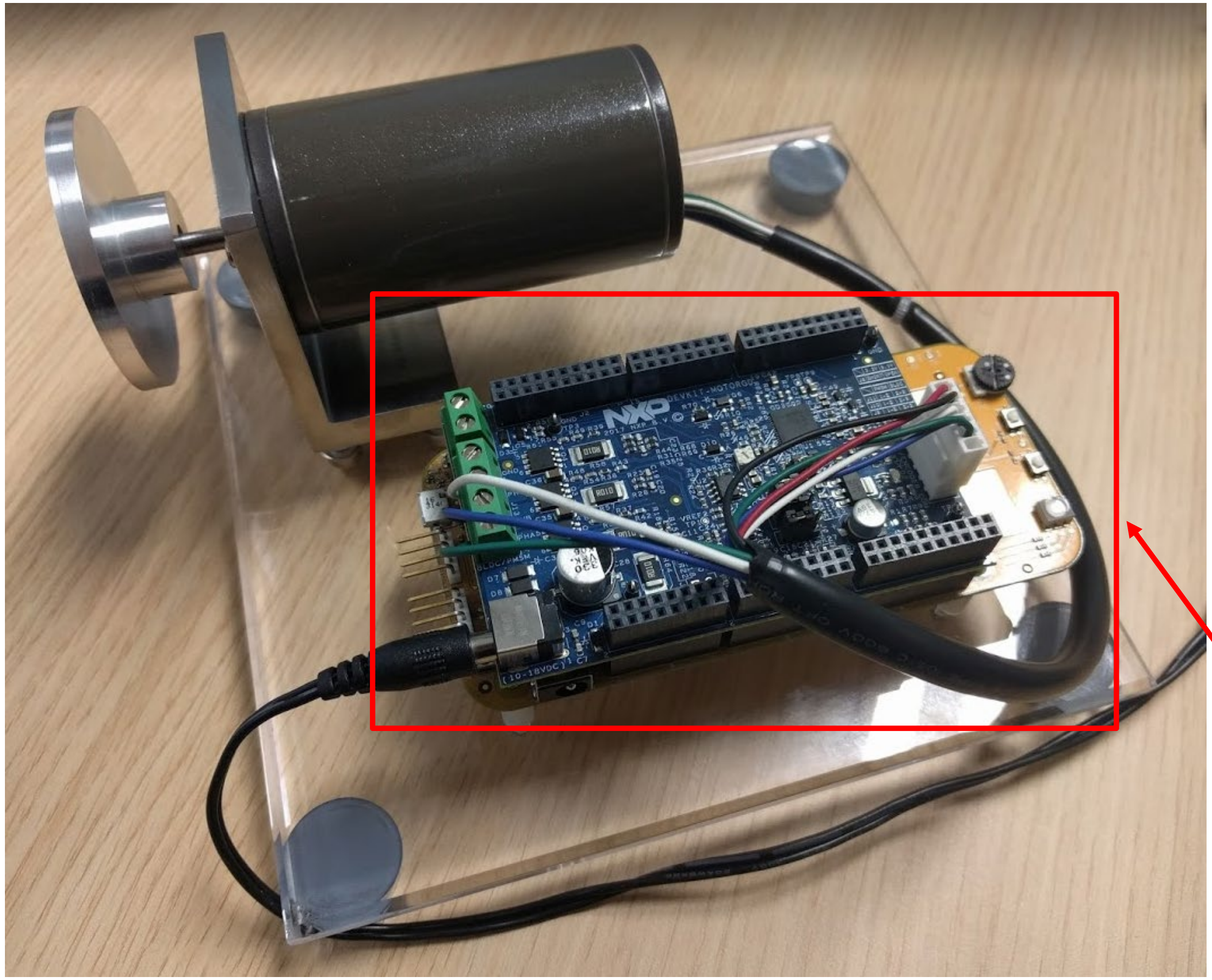
DCBus Ripple Elimination on Uα,β

Space Vector Modulation on U α,β

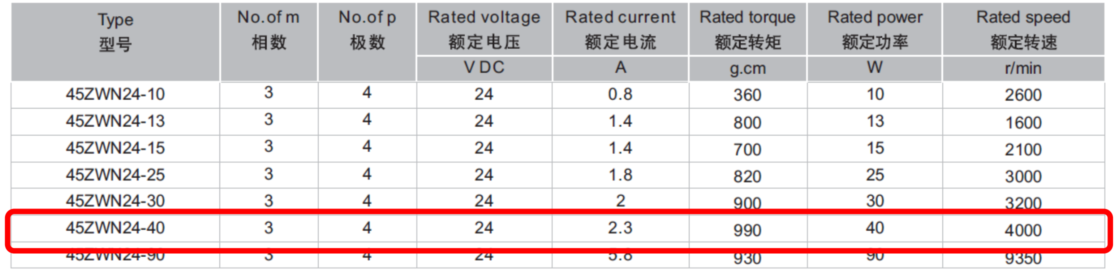
Update duty cycle

return

MTRDEVKSPNK144 – Motor Control Development Kit



LINIX Motor 45ZWN24-40 – Data Sheet

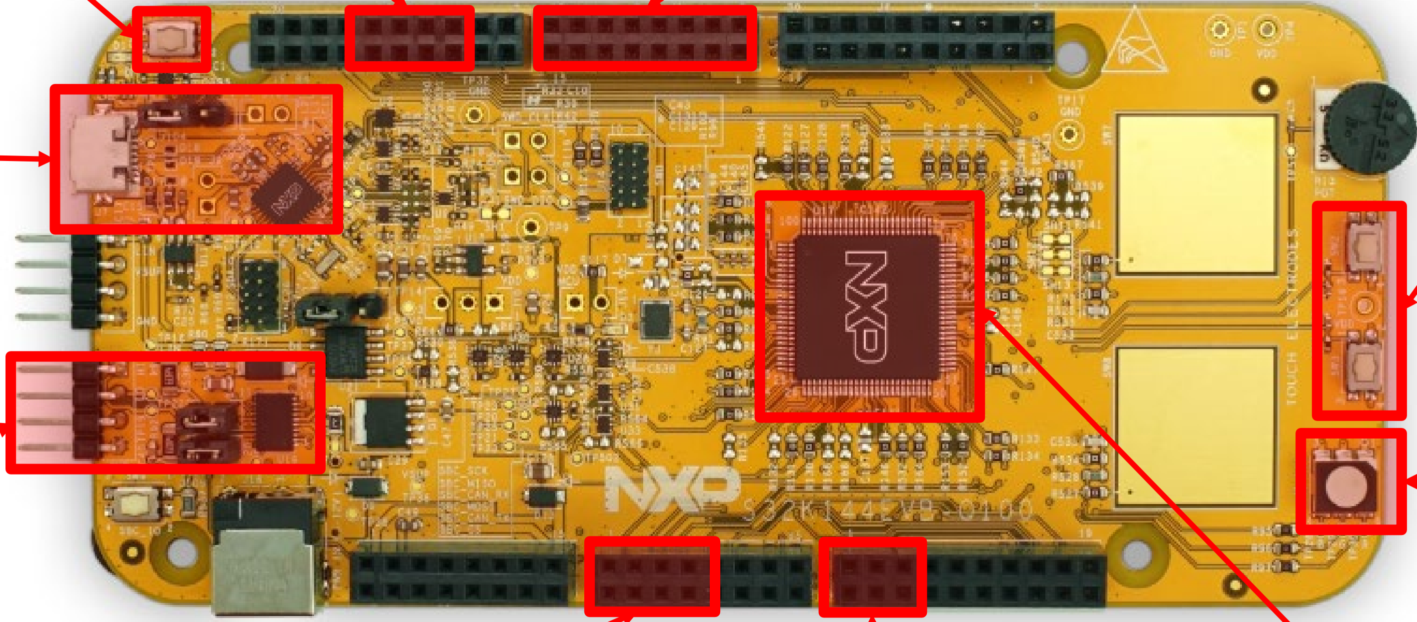


Type 型号	No.of m 相数	No.of p 极数	Rated voltage 额定电压	Rated current 额定电流	Rated torque 额定转矩	Rated power 额定功率	Rated speed 额定转速
			V DC	A	g.cm	W	r/min
45ZWN24-10	3	4	24	0.8	360	10	2600
45ZWN24-13	3	4	24	1.4	800	13	1600
45ZWN24-15	3	4	24	1.4	700	15	2100
45ZWN24-25	3	4	24	1.8	820	25	3000
45ZWN24-30	3	4	24	2	900	30	3200
45ZWN24-40	3	4	24	2.3	990	40	4000
45ZWN24-90	3	4	24	5.8	930	90	9350

Control Tower: DEVKIT MOTORGD on top of S32K144EVB

*All the parts can be bought and assembled separately
*Workshop is designed to be generic and valid for any other combination of S32K14x MCU and power inverter and 3 phase motors

S32K144EVB – Components Used During Workshop



RESET

SPI Signals for pre-driver configuration

PWM Signals

OpenSDA Interface
- Debug interface
- ELF loader
- Communication

GPIO – digital speed command
- SW1 to accelerate
- SW2 to decelerate

LEDs to signal various states

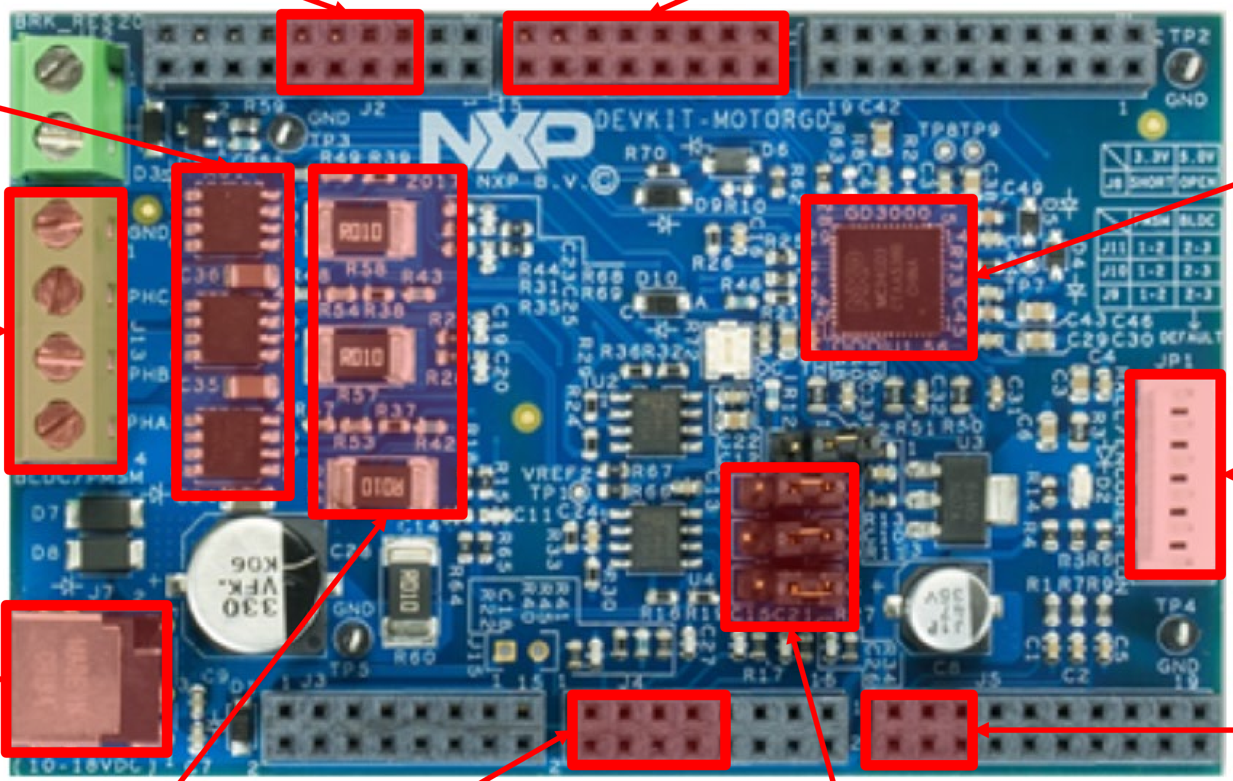
CAN
Optional for data visualization via FreeMASTER

System Measurements

Motor Position Sensor Signals

S32K144 SoC for controlling the system

DEVKIT MOTORGD – Components Used During Workshop



SPI Signals for pre-driver configuration

PWM and Inverter enable signals

3-phase invertor with BUK7K52-60E

Pre-driver MC34GD3000

Motor phases A,B,C

Motor Position Sensors

Power Supply 10-18Vdc

HALL Signals

3-phase current shunts

System measurements

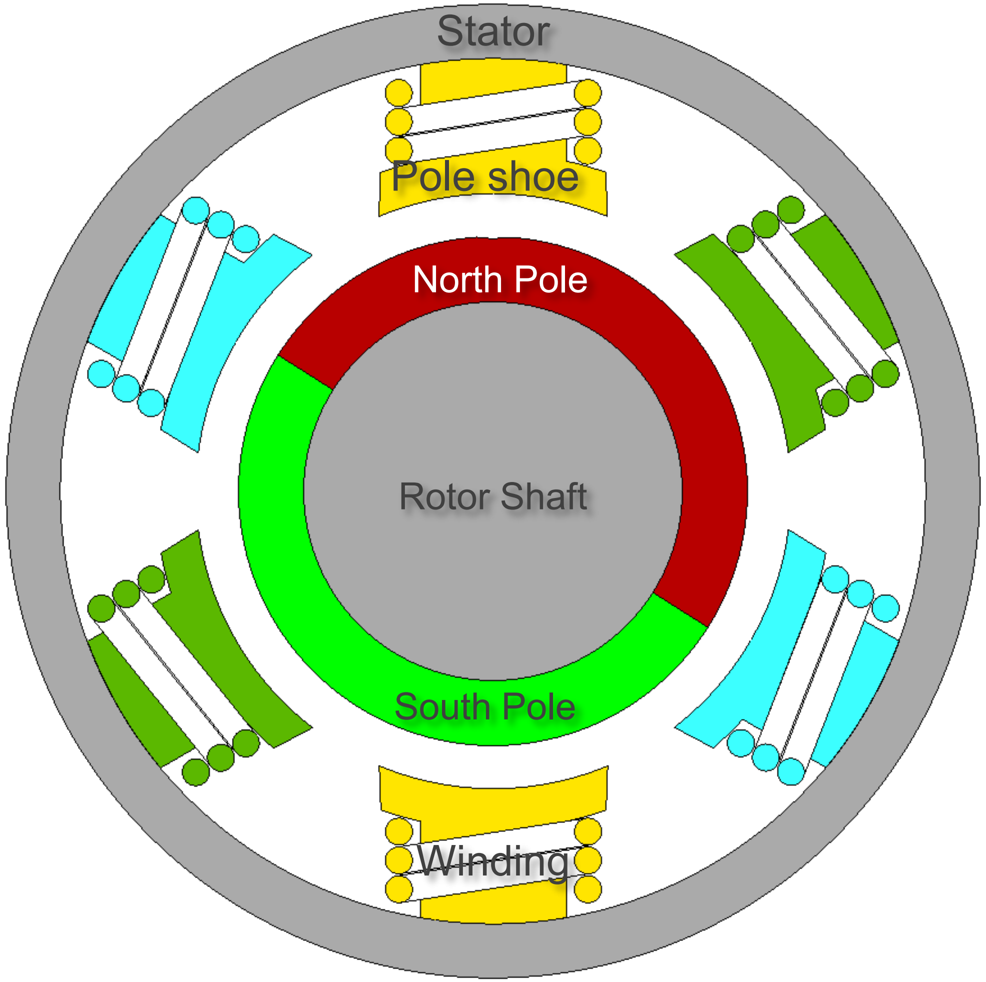
PMSM configuration position 1-2

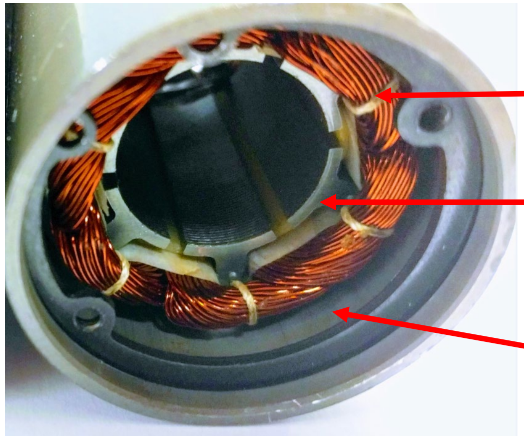
S32K Motor Control How To – Part 2

Agenda

- S32K Solutions
- S32K Motor Control Specific Modules
  - PMSM FOC

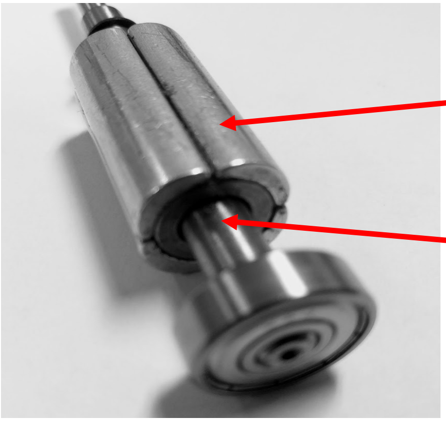
PMSM – Construction Aspects





Stator parts:

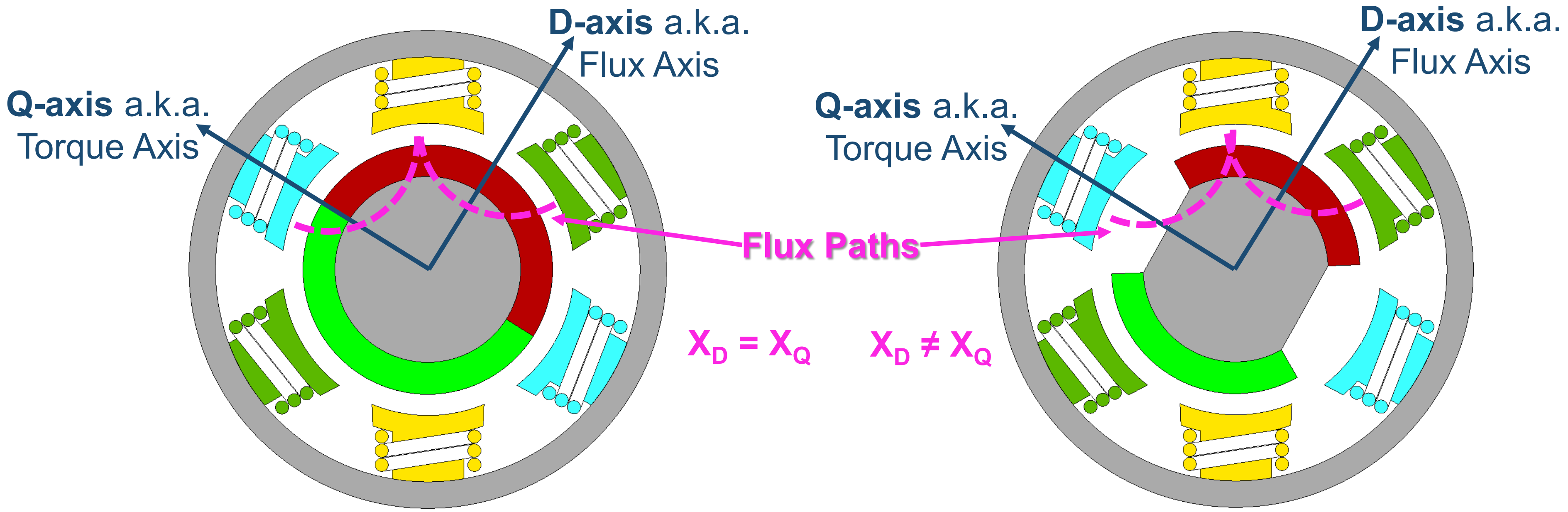
- windings/coils form a 3 phase Y/Δ system energized with AC power
- pole shoe provides a low reluctance path for the magnetic flux
- frame/yoke forms a protective covering and support



Rotor parts:

- permanent magnets mounted on the surface (SPM) or interior (IPM) produces a constant magnetic flux
- shaft that transmits the mechanical torque

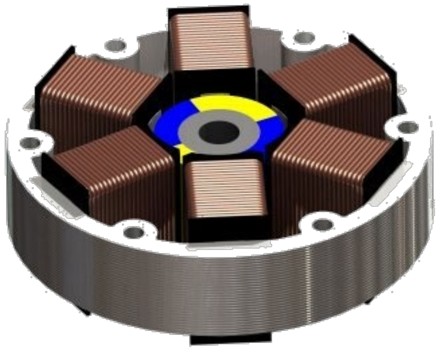
PMSM – Cylindrical vs. Salient Pole Rotor Types

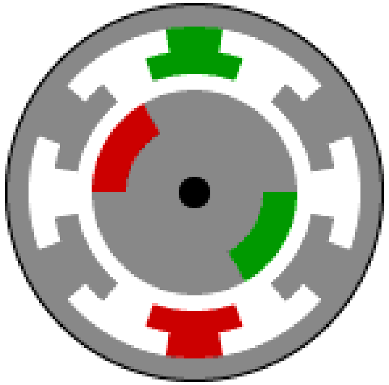


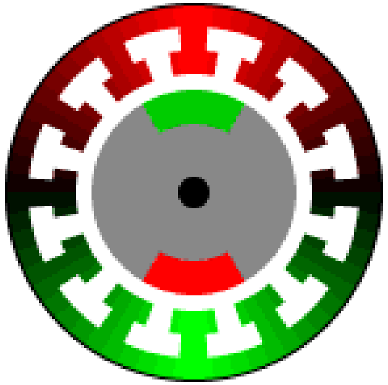
Cylindrical rotor with 4 pole pairs

Salient pole rotor with 2 pole pairs

BLDC vs. PMSM

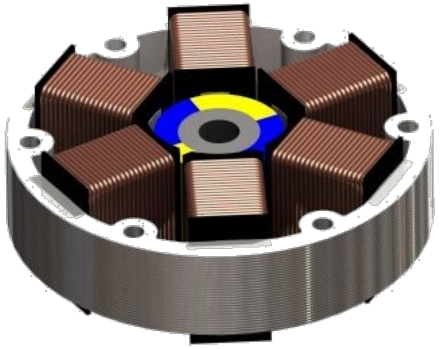






BLDC motor		**PMSM motor**
3-phase machine with PM on the rotor	=	3-phase machine with PM on the rotor
Rotor position sensing required for rotor flux position	=	Rotor position sensing required for rotor flux position
High torque per frame size	=	High torque per frame size
Synchronous operation	=	Synchronous operation
Trapezoidal Back-EMF	≠	**Sinusoidal Back-EMF**
Six-step commutation control	≠	**Field-oriented control**
High torque ripple	≠	**Low torque ripple**

BLDC vs. PMSM in Automotive Industry

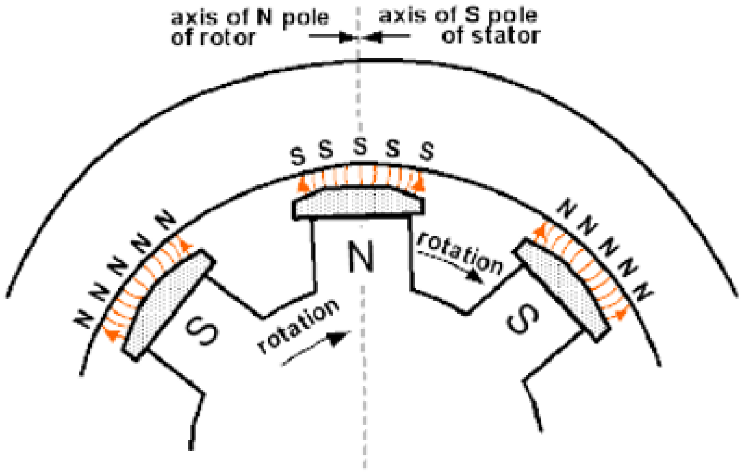


versus

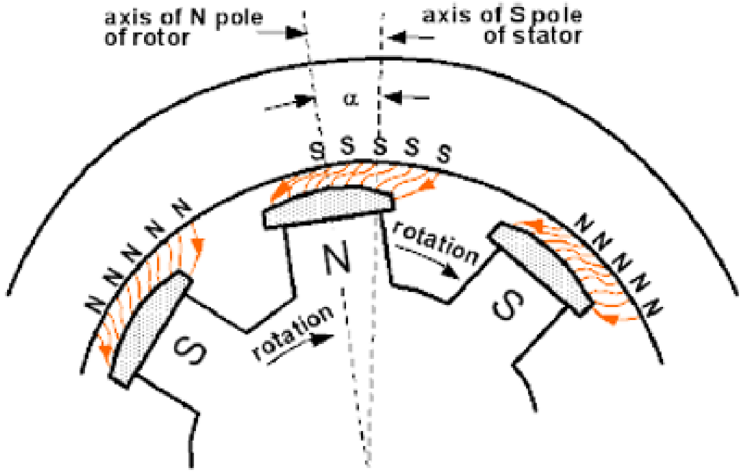
BLDC		PMSM
HIGHER	Level of torque ripple	**LOWER**
HIGHER	Vibration and noise as a consequence of the torque ripple	**LOWER**
LOWER	Electromagnetic compatibility (EMC)	**HIGHER**
LOWER	Control structure complexity level	HIGHER
SHORTER	Execution time of the control approach	LONGER
SIMPLE	Sensorless control	MORE COMPLEX
HIGHER	Heating	**LOWER**
LOWER	Price	HIGHER

PMSM – How is Torque Produced

- Attraction between stator and rotor magnetic poles cause the motor to spin
- Opposite polarity between rotor and stator facing poles create a strong magnetic field
- The mutual attraction between rotor and stator poles, locks the rotor in synchronism with rotating stator magnetic field
- Under no-load condition the rotor and stator pole axes are identical, while under load the axes of rotor poles will lag behind the stator pole axes but still turns at synchronous speeds



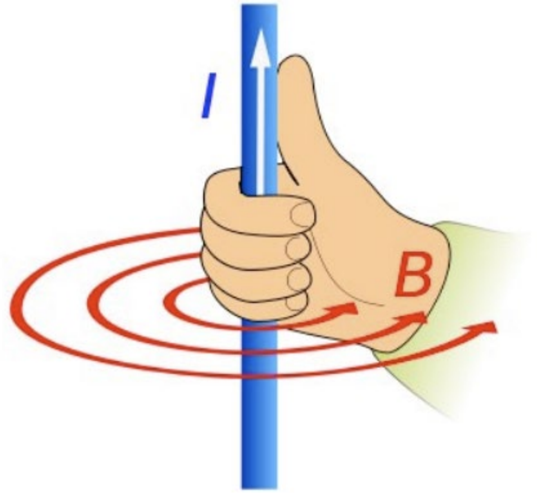
Torque under no-load



Torque under load

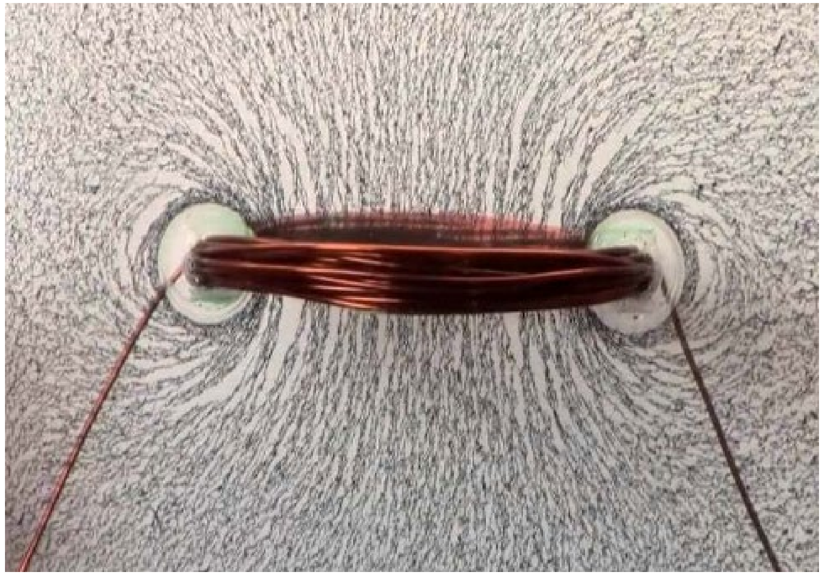
PMSM – Magnetic Field Produced By Stator Windings

Magnetic field **B** around a conductor, produced by current **I**

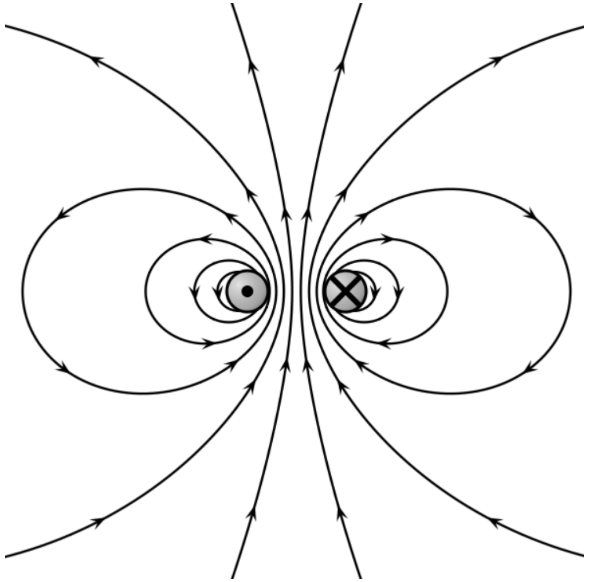


Source: physics.stackexchange.com

Magnetic field produced by a coil/winding

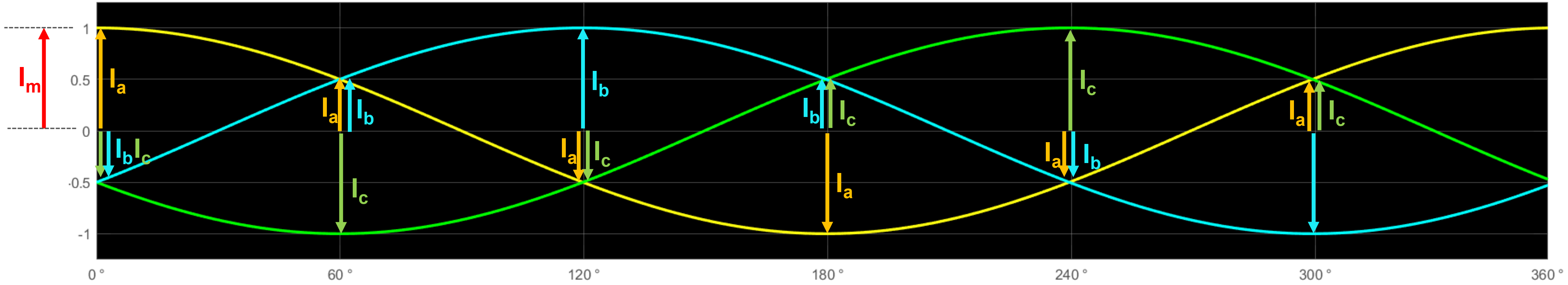


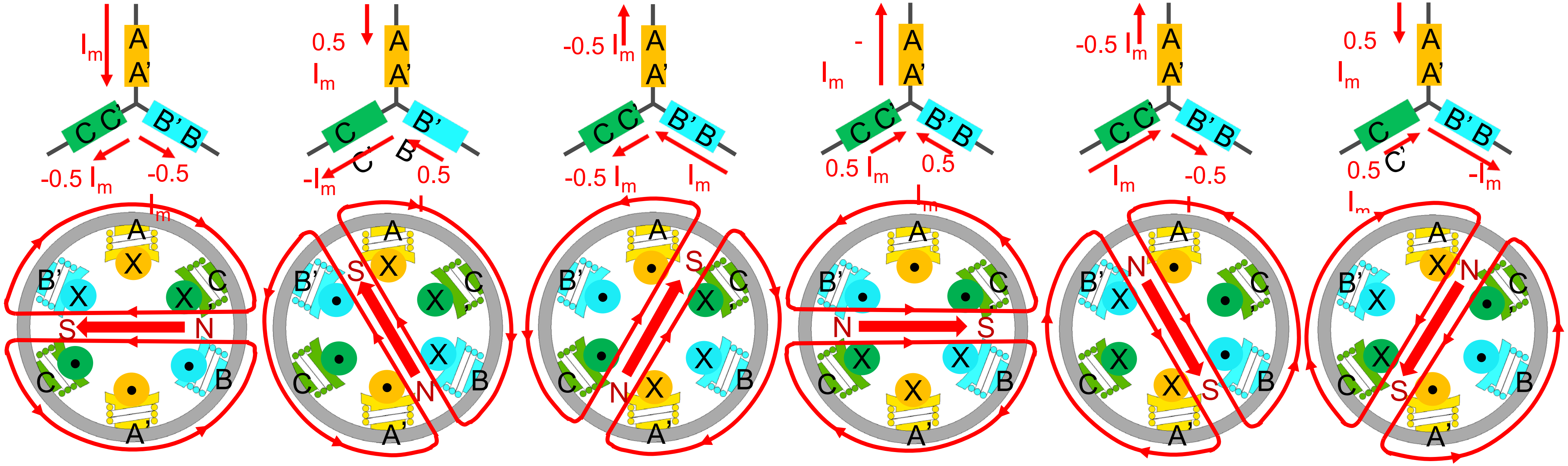
Source: http://www.funscience.in



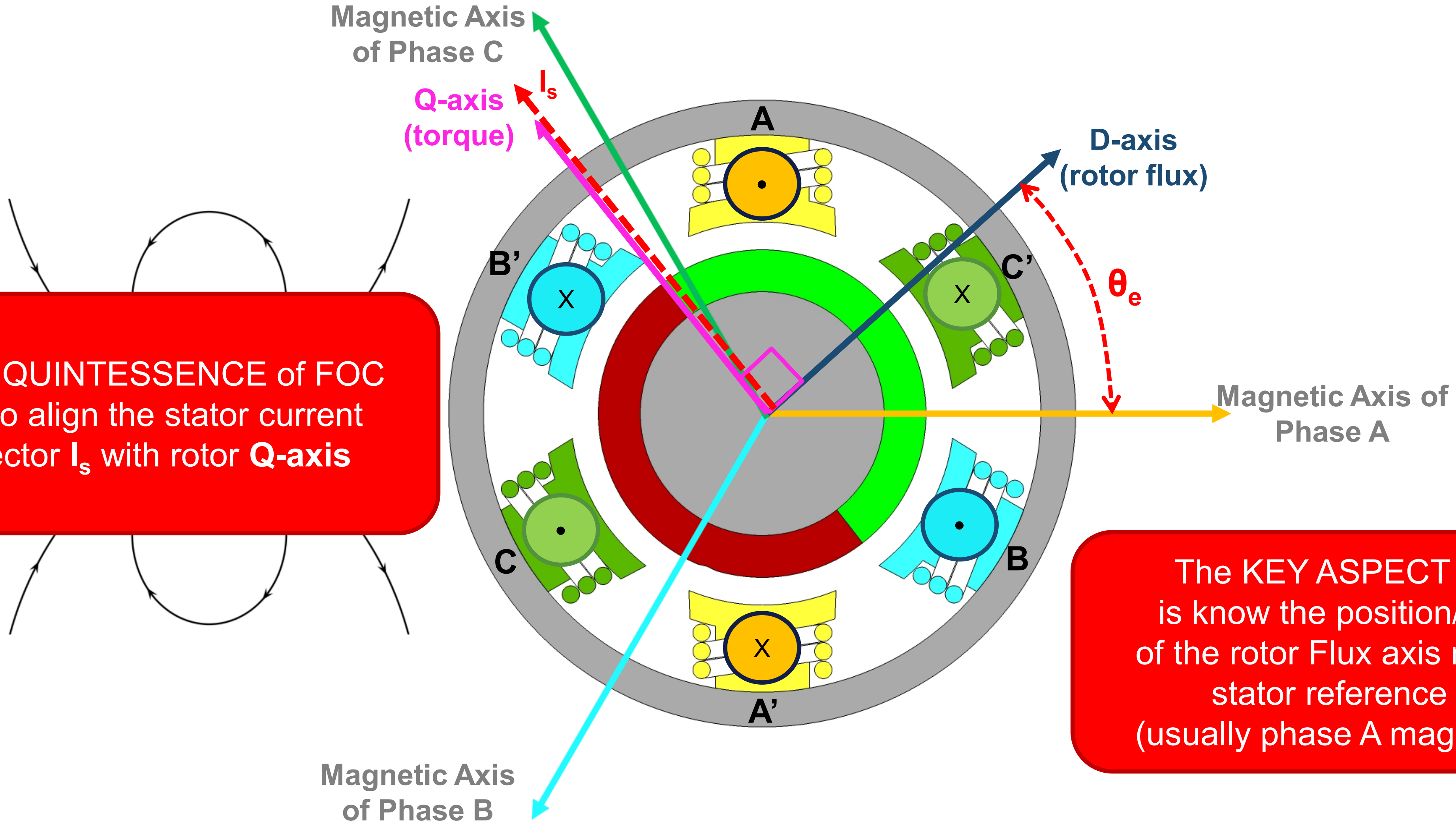
PMSM – How Stator Rotating Magnetic Field is Produced

$i_a(t)$, $i_b(t)$ and $i_c(t)$ 3-phase AC balanced currents (a positive current is considered the one that enters the motor coil)





PMSM – Field Oriented Control (FOC)



The QUINTESSENCE of FOC is to align the stator current vector **I**$_s$ with rotor **Q-axis**

The KEY ASPECT of FOC is know the position/angle $\boldsymbol{\theta_e}$ of the rotor Flux axis relative to a stator reference point (usually phase A magnetic axis)

PMSM – Why Should We Use FOC?

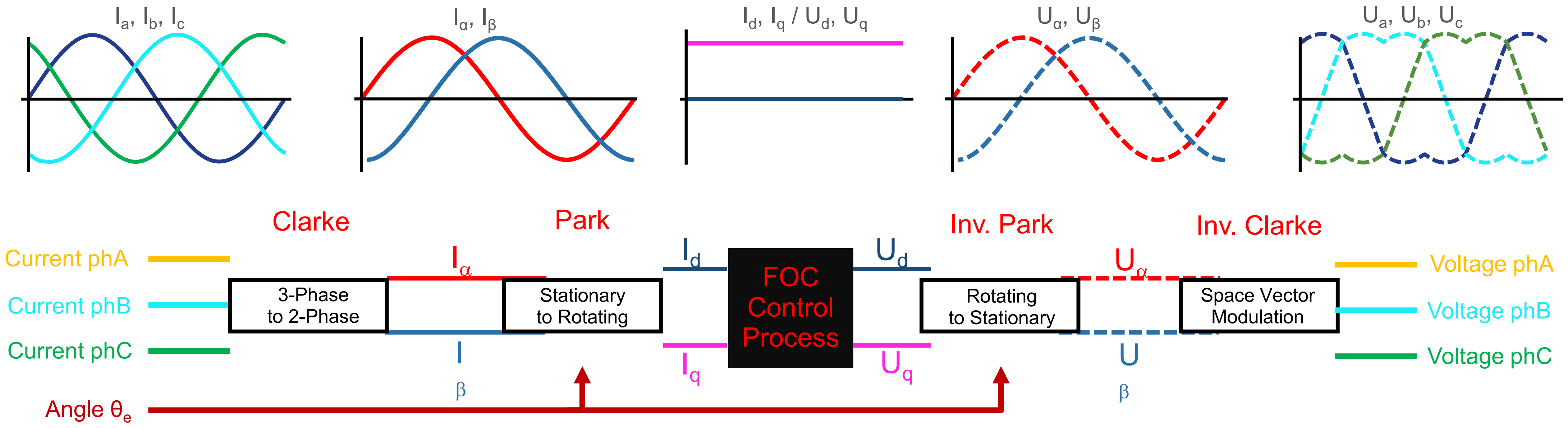
- For a PMSM motor the 3 sinusoidal phase currents i_a, i_b and i_c have to be controlled to create a stator flux vector which is perpendicular to the rotor flux vector. (we will see later why 90 deg)

- Controllig the three sinusoidal currents independantly would be a very complex mathematical task since you have to track and regulate a sinusoidal reference.

- **FOC simplifies the math by transforming the 3 phase stationary system (abc) to a (dq) rotating synchronous with the rotor system.**

- **FOC decomposes the stator currents into two main components that can be controlled separately**:
  - magnetic field-generation (D-axis)
  - torque generation (Q-axis)

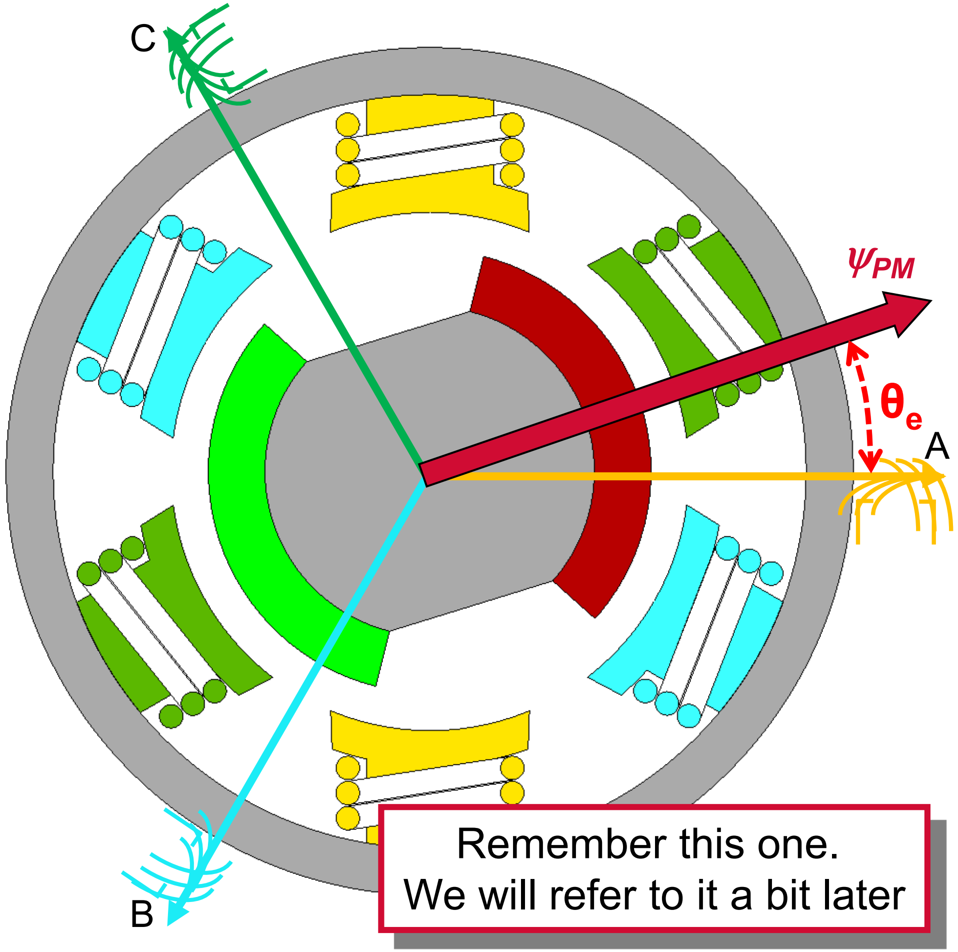
PMSM – FOC Transformations from Stationary to Rotating Frame



3-Phase System	2-Phase System	3-Phase System
AC CURRENT Feedback Quantities	DC quantities	AC VOLTAGE Command Quantities
Stationary Reference Frame	Rotating Reference Frame	Stationary Reference Frame

FOC – 3 Phase (ABC) PMSM Model

Assumptions: sinusoidal 3-phase distributed winding and neglecting effect of magnetic saturation and leakage inductances



Remember this one.
We will refer to it a bit later

Time variation – how to get rid of it

Stator voltage equations

$$\begin{bmatrix} u_A \\ u_B \\ u_C \end{bmatrix} = R \begin{bmatrix} i_A \\ i_B \\ i_C \end{bmatrix} + \frac{d}{dt} \begin{bmatrix} \psi_A \\ \psi_B \\ \psi_C \end{bmatrix}$$

Clarke Transformation
3 phase -> 2 phase

Stator linkage flux

$$\begin{bmatrix} \psi_A \\ \psi_B \\ \psi_C \end{bmatrix} = \begin{bmatrix} L_{aa} & L_{ab} & L_{ac} \\ L_{ba} & L_{bb} & L_{bc} \\ L_{ca} & L_{cb} & L_{cc} \end{bmatrix} \begin{bmatrix} i_A \\ i_B \\ i_C \end{bmatrix} + \psi_{PM} \begin{bmatrix} \cos(\theta_e) \\ \cos\left(\theta_e - \frac{2}{3}\pi\right) \\ \cos\left(\theta_e + \frac{2}{3}\pi\right) \end{bmatrix}$$

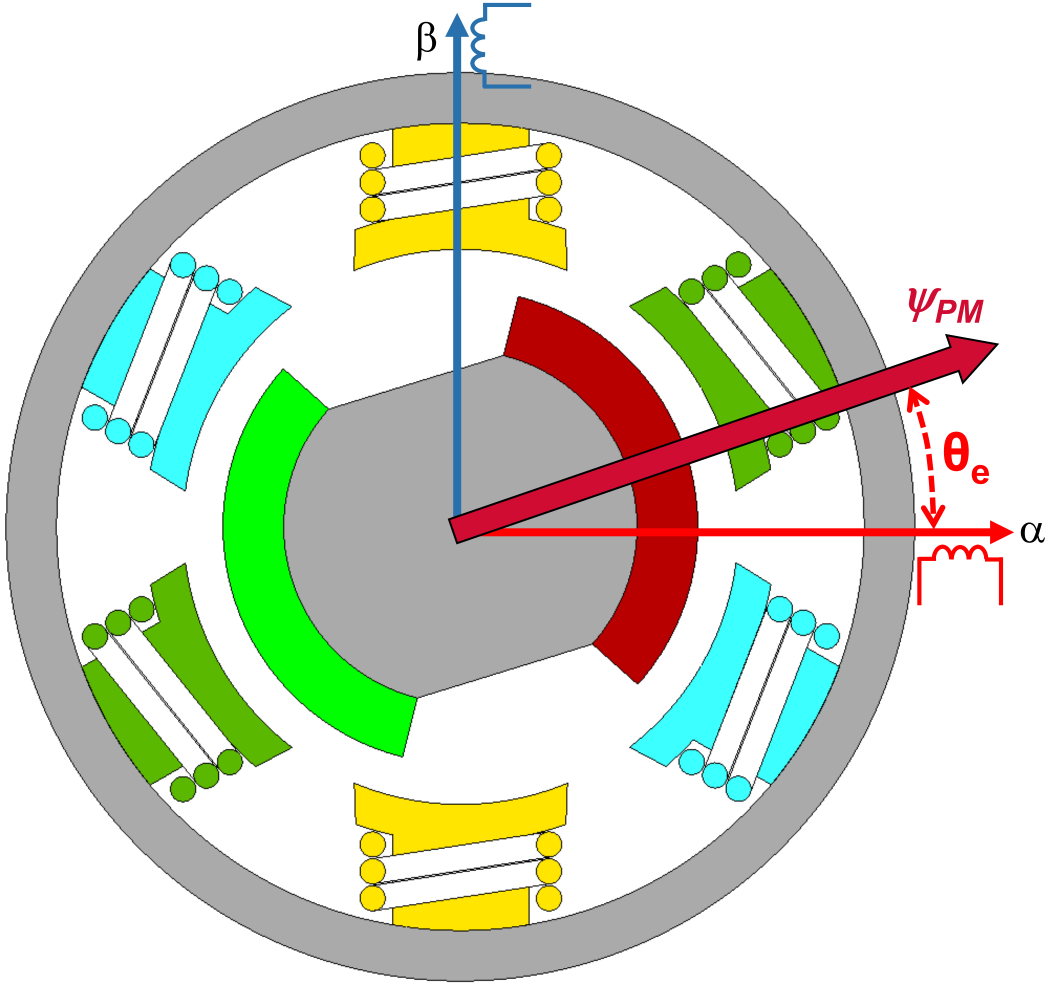
Internal motor torque

$$T_i = \frac{P_i}{\omega_m} = \frac{p_p}{\omega_e}(u_{iA} i_A + u_{iB} i_B + u_{iC} i_C)$$

$$T_i = p_p(-\Psi_{PM} i_A \sin(\theta_e) - \Psi_{PM} i_B \sin(\theta_e - \tfrac{2}{3}\pi) - \Psi_{PM} i_C \sin(\theta_e + \tfrac{2}{3}\pi))$$

FOC – 2 Phase (αβ) PMSM Model

Motor equations are getting simpler but still contains time variations (AC – component)



Stator voltage equations

$$\boxed{\begin{bmatrix} u_\alpha \\ u_\beta \end{bmatrix}} = R_S \boxed{\begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix}} + \frac{d}{dt} \boxed{\begin{bmatrix} \psi_\alpha \\ \psi_\beta \end{bmatrix}}$$

Park Transformation
AC -> DC

Stator linkage flux

$$\begin{bmatrix} \Psi_{S\alpha} \\ \Psi_{S\beta} \end{bmatrix} = \begin{bmatrix} L_S & 0 \\ 0 & L_S \end{bmatrix} \cdot \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} + \Psi_{PM}\Big|_{i_{Sd}=0} \cdot \begin{bmatrix} \cos\theta_e \\ \sin\theta_e \end{bmatrix}$$

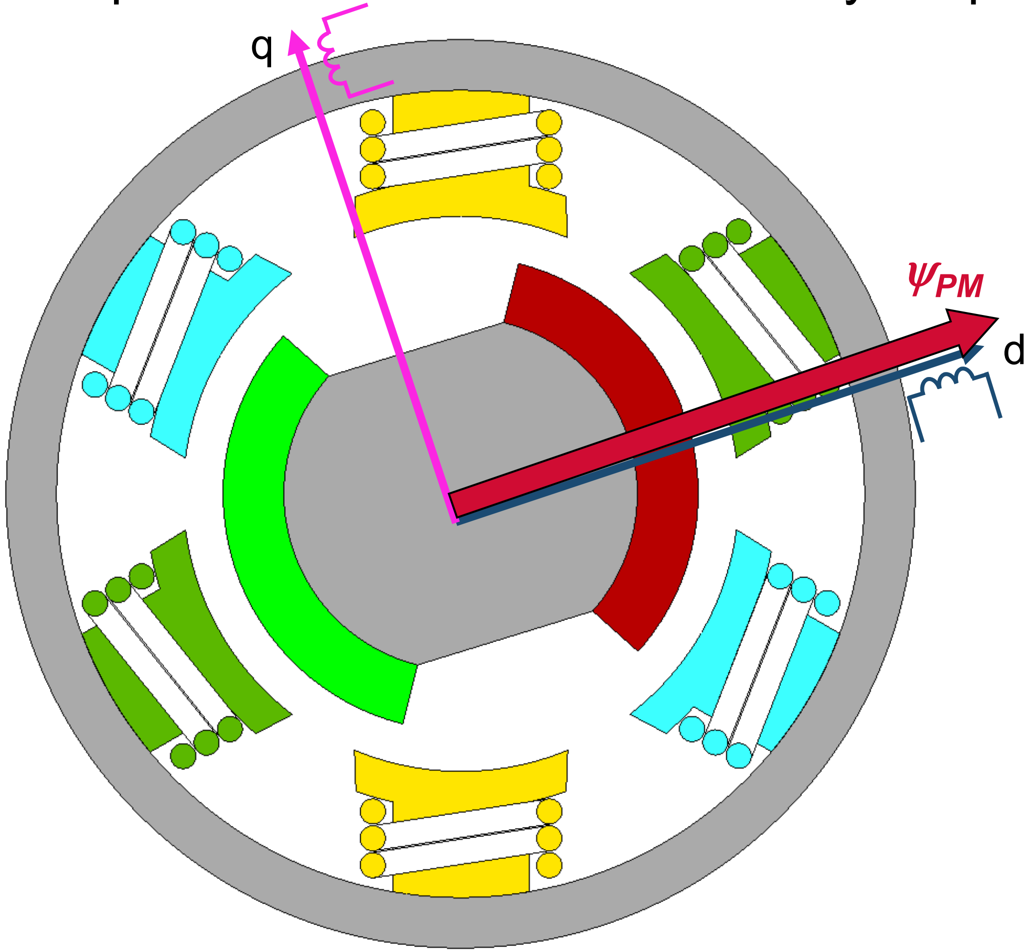
Internal motor torque

$$T_i = \frac{3}{2}\frac{p_p}{\omega_e}(u_{i\alpha} i_\alpha + u_{i\beta} i_\beta) = \frac{3}{2} p_p (\Psi_\alpha i_\beta - \Psi_\beta i_\alpha)$$

FOC – (*dq*) Synchronous Frame PMSM Model

PMSM model in (dq) synchronous frame aligned with the rotor

All equations can now be resolved by simple linear algebra



Stator voltage equations

$$\begin{bmatrix} u_d \\ u_q \end{bmatrix} = R_s \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} s & \omega_e \\ -\omega_e & s \end{bmatrix} \begin{bmatrix} \psi_d \\ \psi_q \end{bmatrix}$$

Stator linkage flux

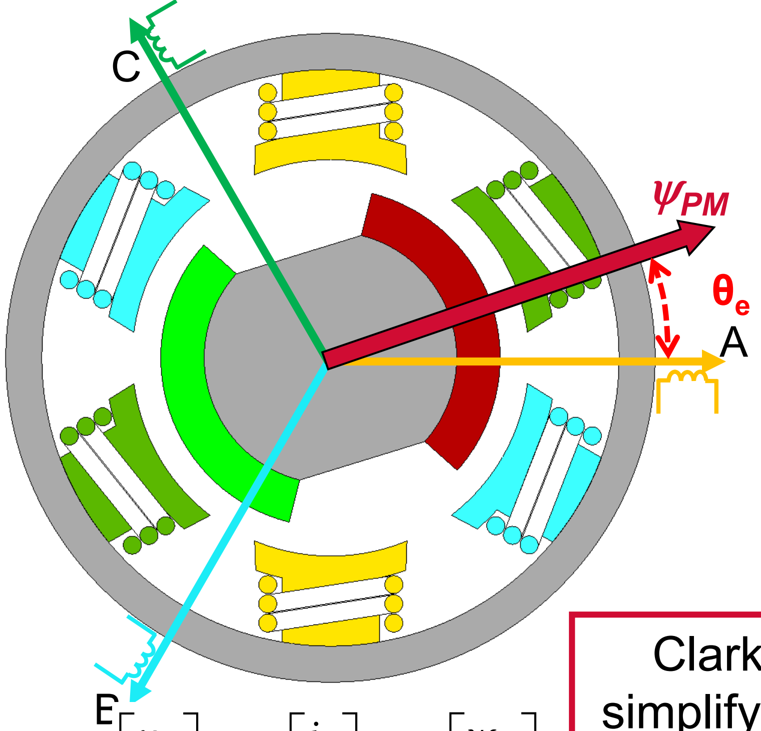
$$\begin{bmatrix} \psi_d \\ \psi_q \end{bmatrix} = \begin{bmatrix} L_d & 0 \\ 0 & L_q \end{bmatrix} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \psi_{PM} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Internal motor torque

$$T_i = \frac{3}{2} \frac{p_p}{\omega_e} (u_{id} i_d + u_{iq} i_q) = \frac{3}{2} p_p (\Psi_d i_q - \Psi_q i_d) = \frac{3}{2} p_p \cdot \Psi_{PM} i_q$$

FOC – Transformations in Nutshell

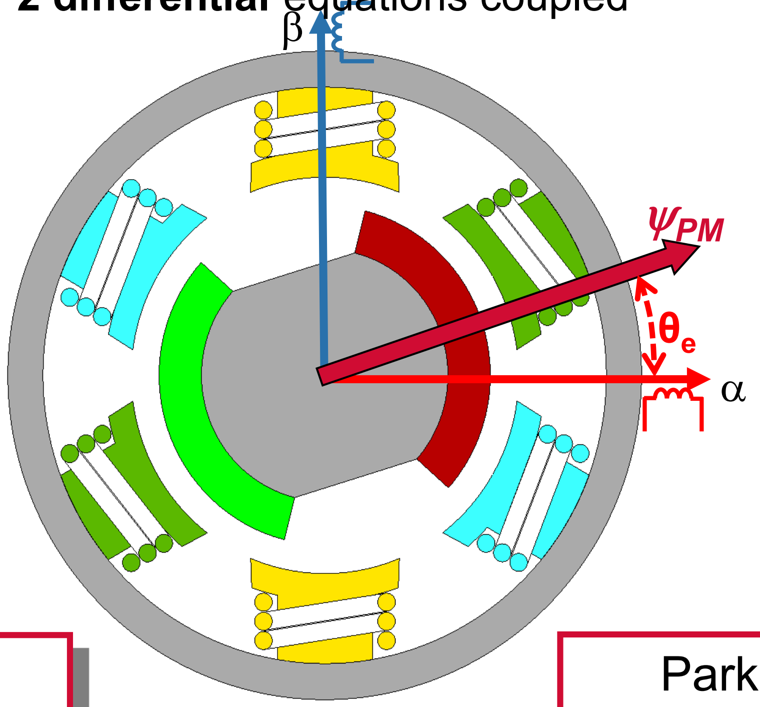
(abc) **stationary** reference frame
3 **AC** quantities (voltage/current/flux)
3 differential equations coupled



$$\begin{bmatrix} u_A \\ u_B \\ u_C \end{bmatrix} = R_s \begin{bmatrix} i_A \\ i_B \\ i_C \end{bmatrix} + \frac{d}{dt} \begin{bmatrix} \psi_A \\ \psi_B \\ \psi_C \end{bmatrix}$$

Clarke
simplify eq.

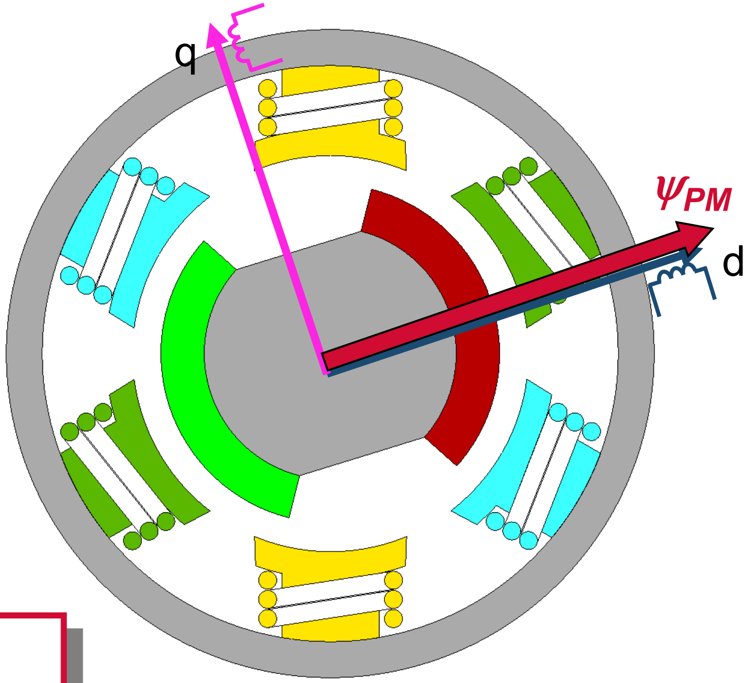
(αβ) **stationary** reference frame
3 **AC** quantities (voltage/current/flux)
2 differential equations coupled



$$\begin{bmatrix} u_\alpha \\ u_\beta \end{bmatrix} = R_s \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} + \frac{d}{dt} \begin{bmatrix} \psi_\alpha \\ \psi_\beta \end{bmatrix}$$

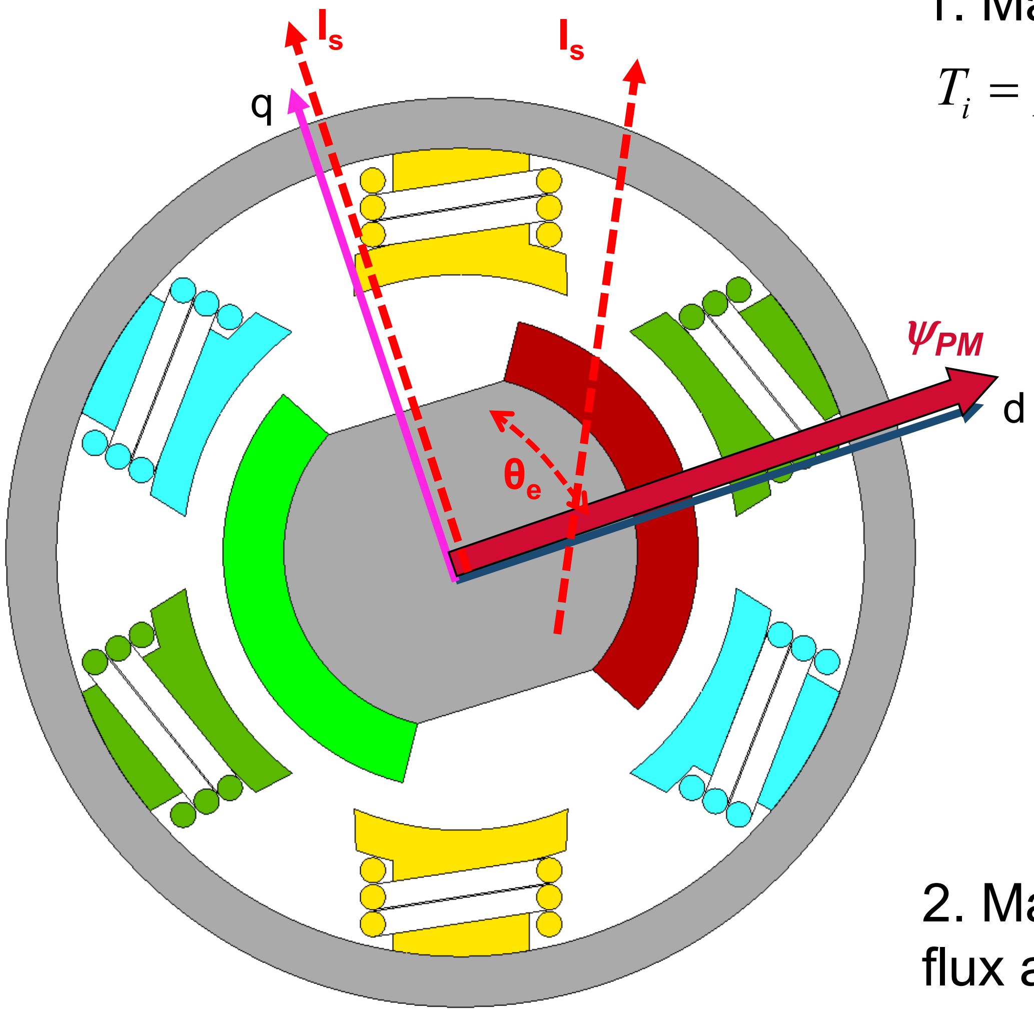
Park
linearize eq.

(dq) **rotating** reference frame
3 **DC** quantities (voltage/current/flux)
2 linear equations cross-coupled



$$\begin{bmatrix} u_d \\ u_q \end{bmatrix} = R_s \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} s & \omega_e \\ -\omega_e & s \end{bmatrix} \begin{bmatrix} \psi_d \\ \psi_q \end{bmatrix}$$

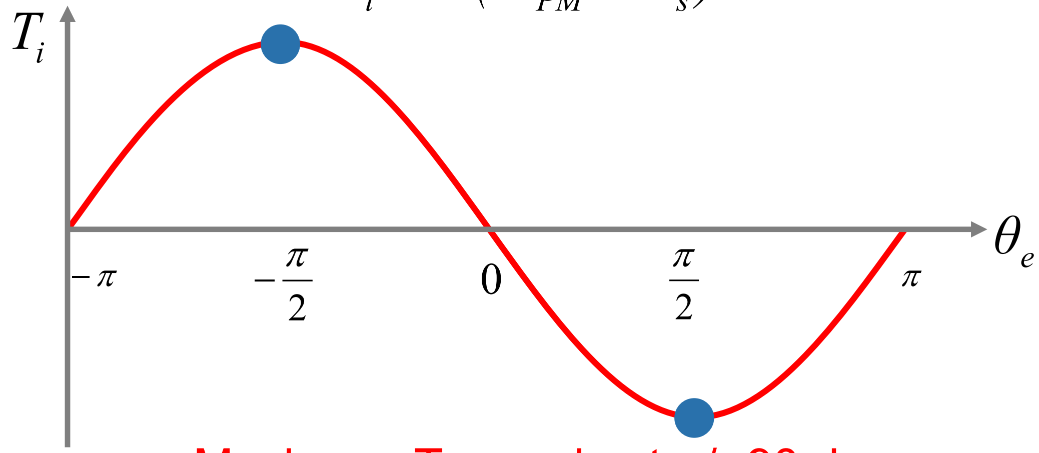
PMSM – Why 90deg Between I_s and Rotor Flux Vectors?



1. Maximum Torque per Amps criteria

$$T_i = p_p(-\Psi_{PM} i_A \sin(\theta_e) - \Psi_{PM} i_B \sin(\theta_e - \tfrac{2}{3}\pi) - \Psi_{PM} i_C \sin(\theta_e + \tfrac{2}{3}\pi))$$

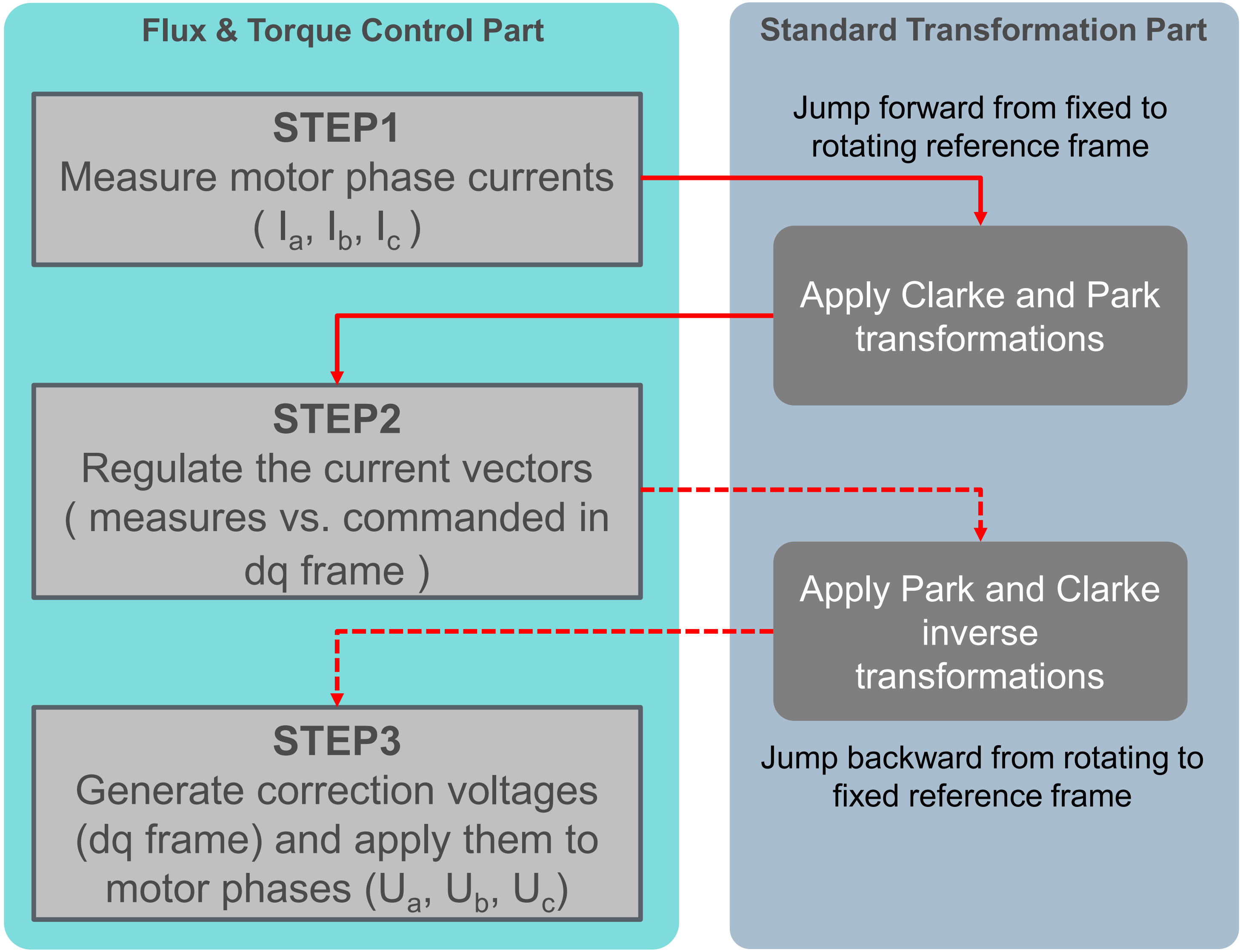
$$T_i = K(\vec{\Psi}_{PM} \otimes \vec{I}_s)$$



Maximum Torque is at +/- 90 degrees

2. Maximize the usage of PM flux: the I_s projection on rotor flux axis shall be 0 in case field weakening is not needed

PMSM – FOC Control Stages (Recap)



Flux & Torque Control Part

STEP1
Measure motor phase currents
(I_a, I_b, I_c)

STEP2
Regulate the current vectors
(measures vs. commanded in dq frame)

STEP3
Generate correction voltages (dq frame) and apply them to motor phases (U_a, U_b, U_c)

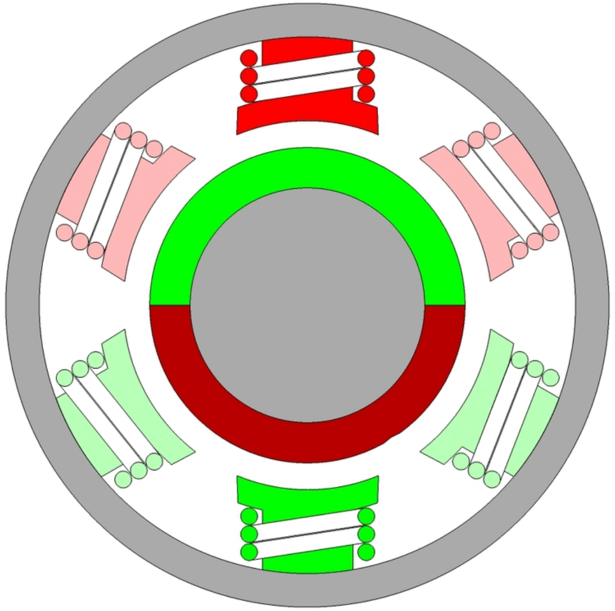
Standard Transformation Part

Jump forward from fixed to rotating reference frame

Apply Clarke and Park transformations

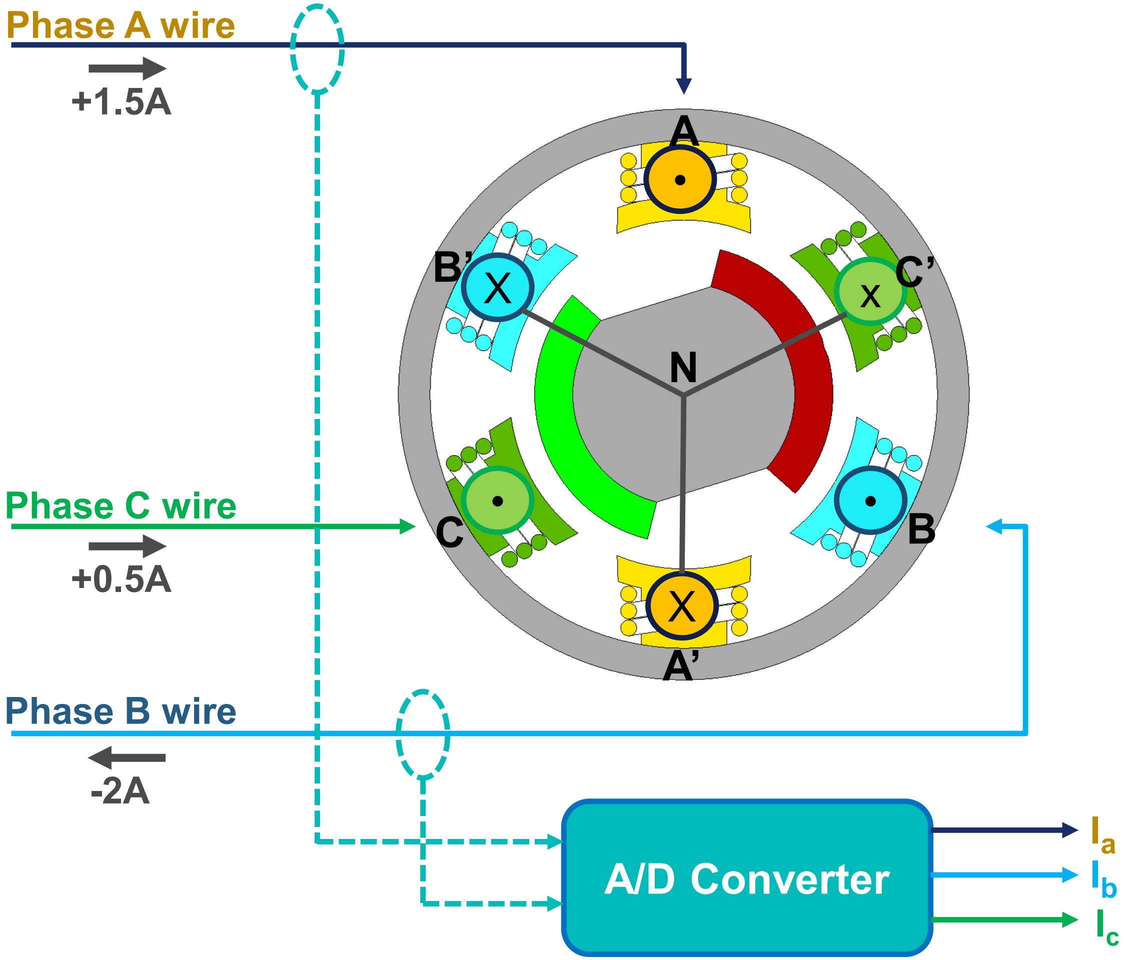
Apply Park and Clarke inverse transformations

Jump backward from rotating to fixed reference frame



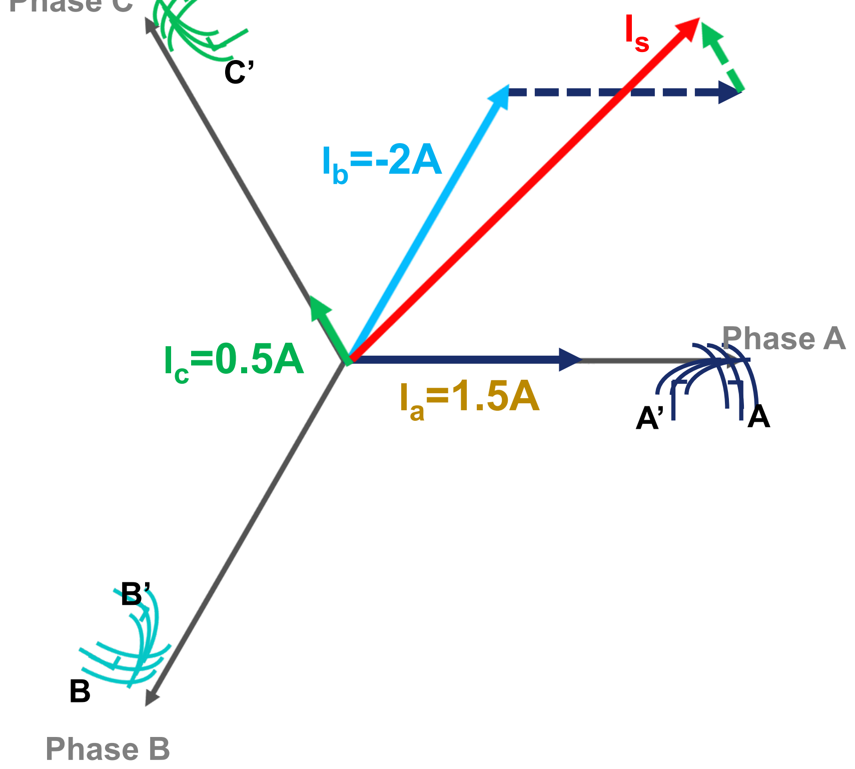
Step 1: Motor Phase Current Measurement

$$\overline{I}_s = I_a e^{j0} + I_b e^{j120^\circ} + I_c e^{j240^\circ}$$



$I_c = - (I_a + I_b)$

I_c is computed from Kirkoff's law

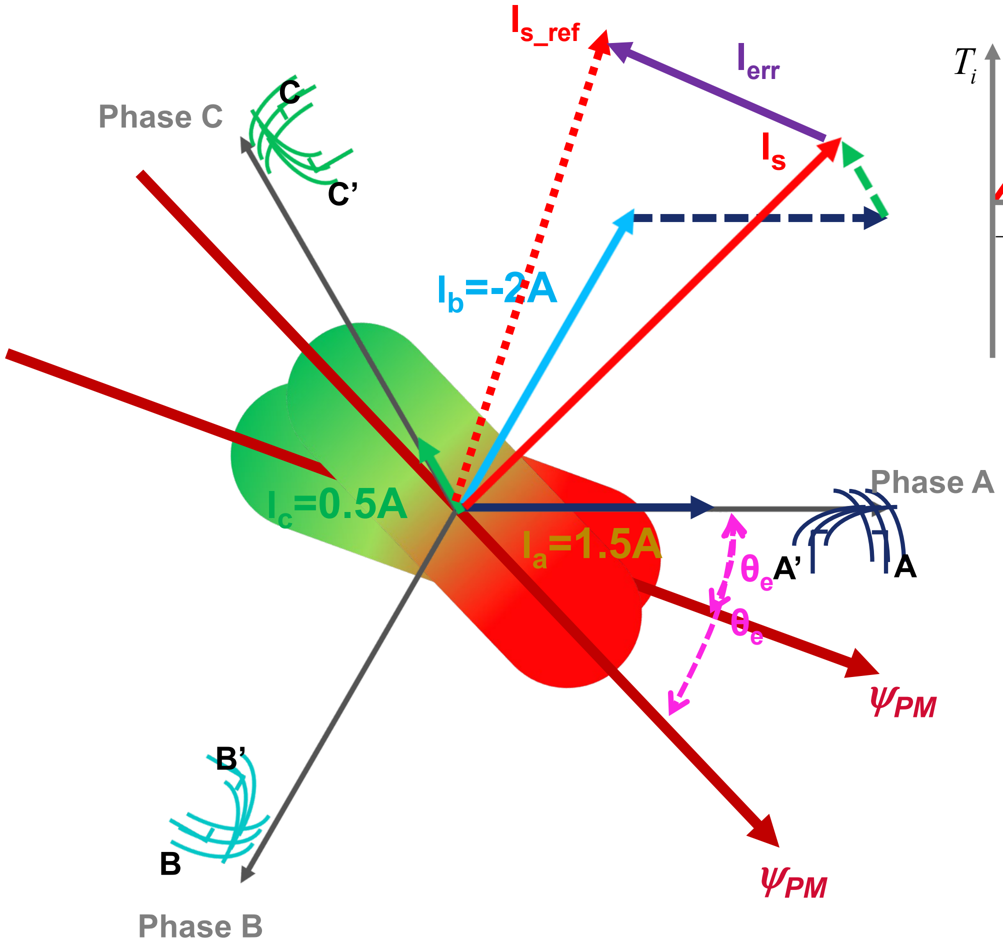


Step 2: Regulate Current Vector

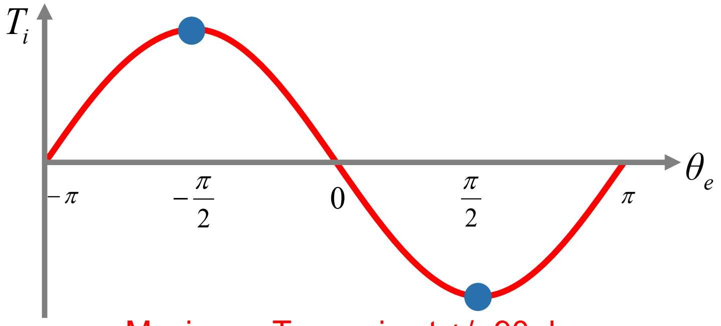
- Make sure I_s is at 90 deg relative to rotor flux axis

- Get the angle θ_e between rotor flux axis and stator Phase A axis

- Calculate the error between measured current vector I_s and the desired current vector I_{s_ref} ($I_{err} = I_{s\_ref} - I_s$)



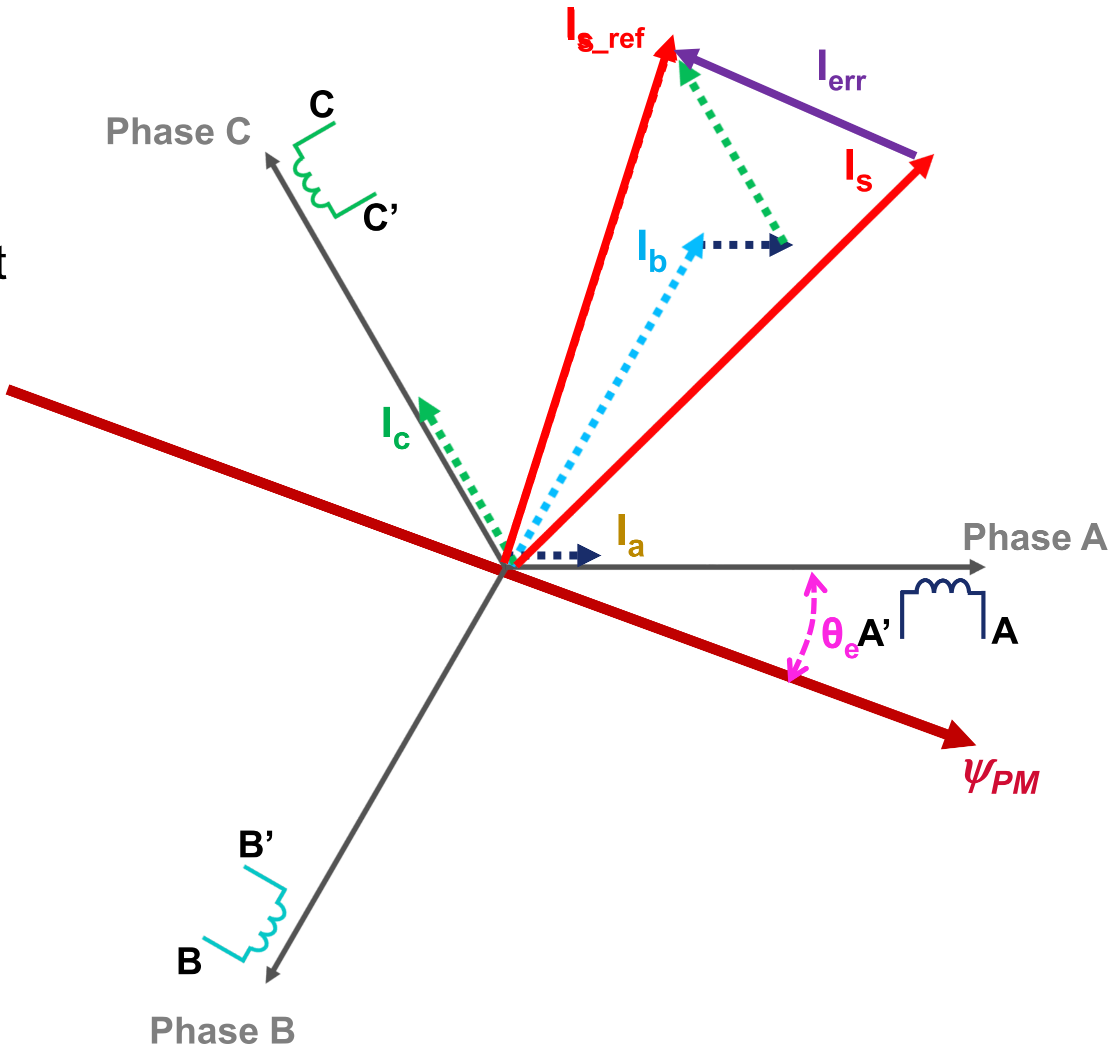
$$T_i = K(\vec{\Psi}_{PM} \otimes \vec{I}_s)$$



Maximum Torque is at +/- 90 degrees

Step 3: Generate Corrections

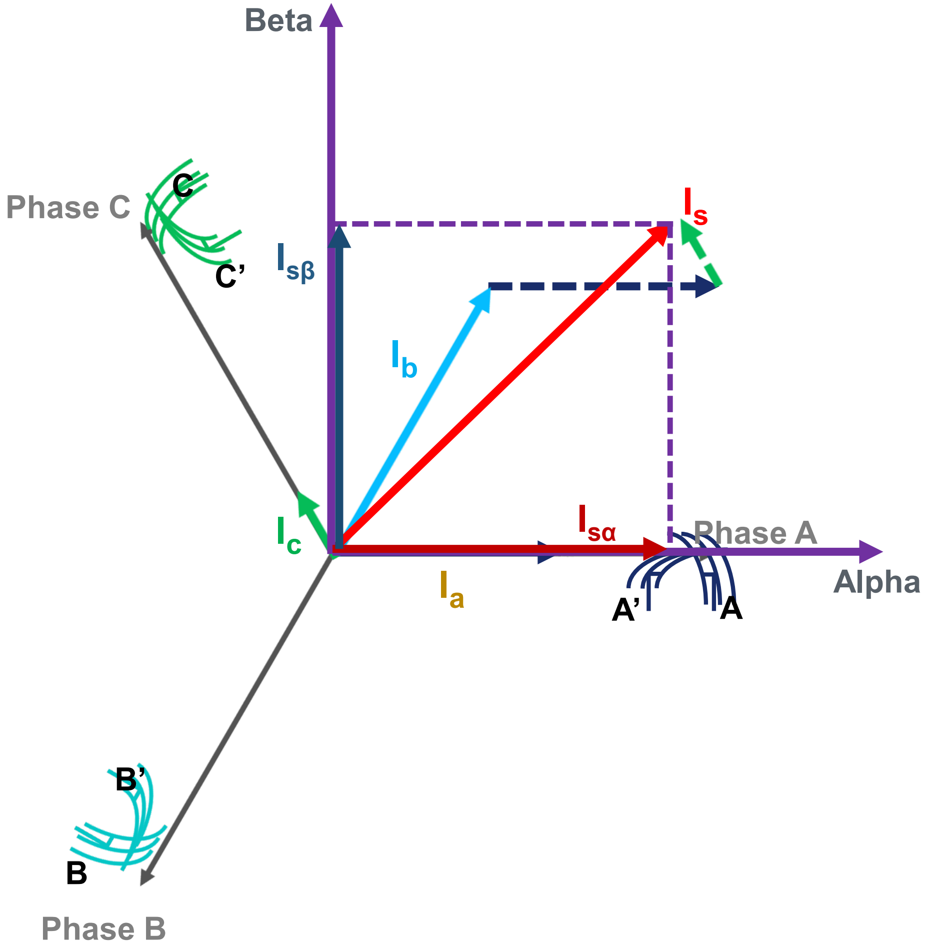
- Compute the new appropriate I_a, I_b and I_c that will minimize the I_{err} forcing the stator current vector I_s to align align with ideal I_{s_ref} stator reference vector



FOC – From Theory to Action

- The 3 control steps (Measure, Control and Correct) represent the “theory” behind Field Oriented Control;
- The “theory” is explained in static reference system (abc) which deals with time variant winding currents and voltages represented as vectors in previous diagrams;
- Even if the AC quantities control is doable in todays world of microprocessors, nobody use it since the limitations overcome the benefits. Designing a control system that tracks an time variant quantity pose big challenges in terms of controller frequency response and phase shift;
- Field Oriented Control solves this problem by controlling the current space vector directly in the **dq** reference frame of the rotor. Because the current space vector in the dq reference frame is static, the PI controllers operate on DC, rather than sinusoidal signals. Using Field Oriented Control, the quality of current control is largely unaffected by speed of rotation of the motor;
- To achieve that, the measured motor currents must be mathematically transformed from the three-phase static reference frame of the stator windings to the two axis rotating dq reference frame, prior to processing by the PI controllers. Similarly, the voltages to be applied to the motor are mathematically transformed from the dq frame of the rotor to the three-phase reference frame of the stator before they can be used for PWM output.

FOC – Implementation (abc2αβ – Transformation)

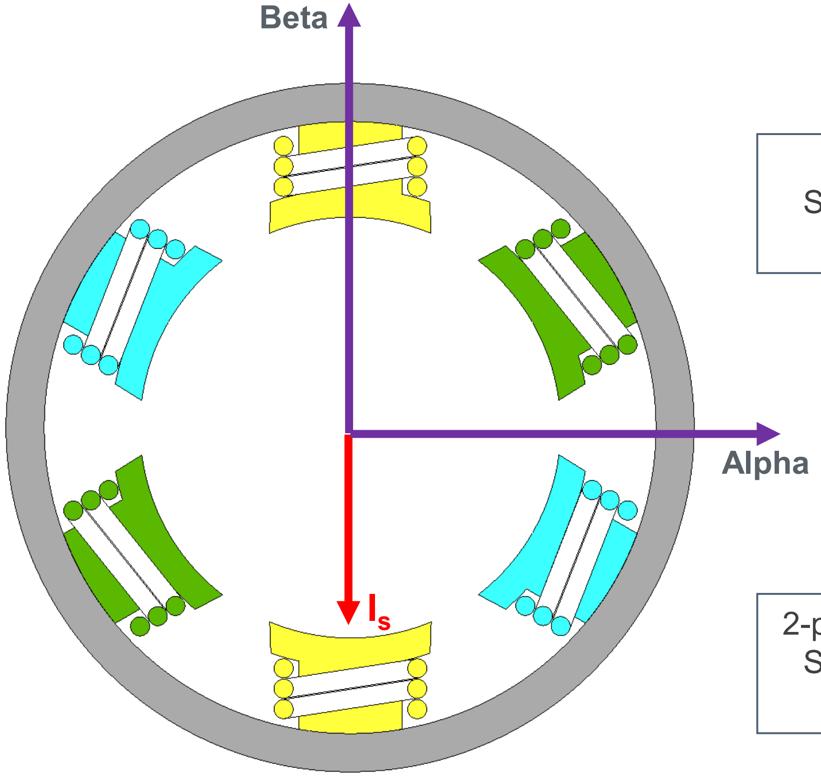


3-phase Stationary
to 2-phase Stationary
(Forward Clarke Transform)

$$\begin{bmatrix} I_{s\alpha} \\ I_{s\beta} \end{bmatrix} = \begin{bmatrix} \frac{3}{2} & 0 & 0 \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} I_a \\ I_b \\ I_c \end{bmatrix}$$

FOC – Stationary Reference Frame Transformation

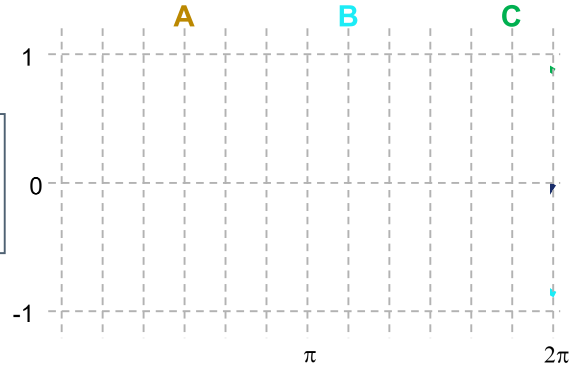
- Measured currents variation before and after Forward Clarke Transformation



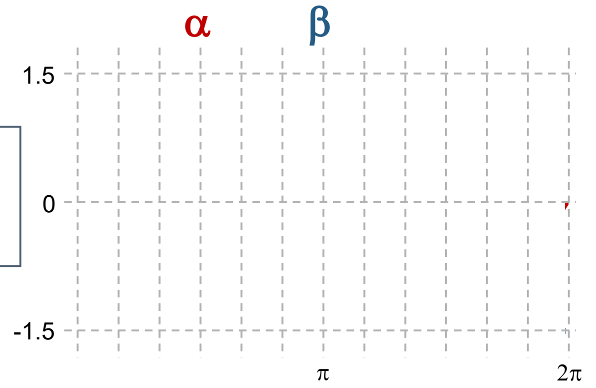
3-phase
Stationary Frame
(abc)

2-phase quadrature
Stationary Frame
(αβ)

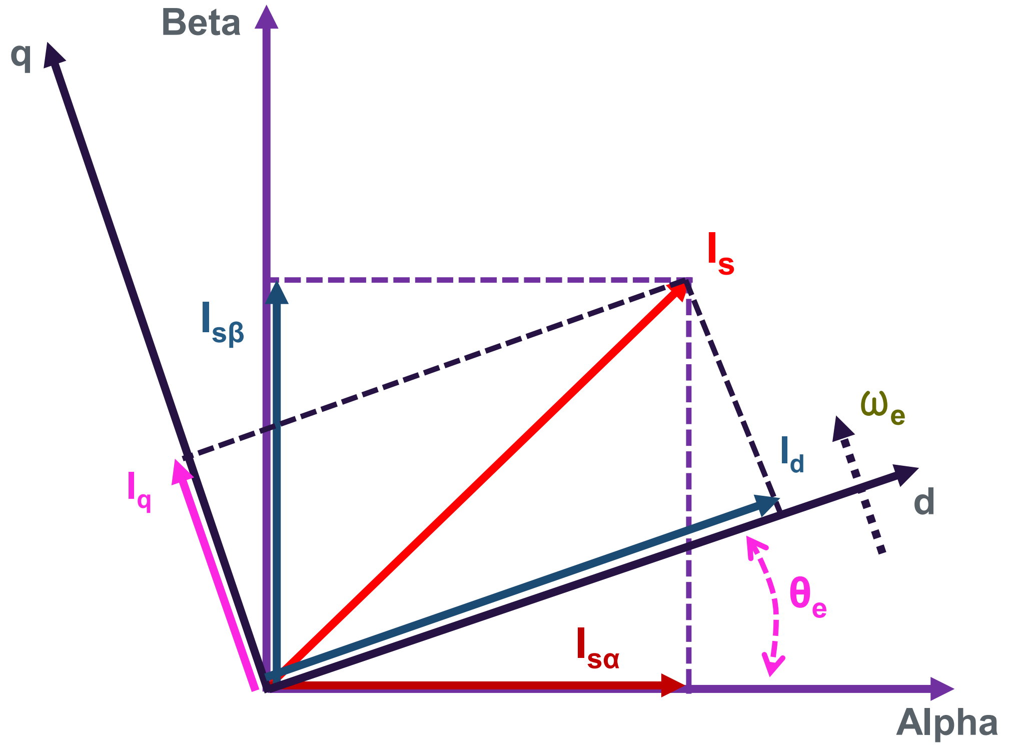
3-phase measured currents variation



2-phase transformed currents variation



FOC – Implementation (αβ2dq – Transformation)

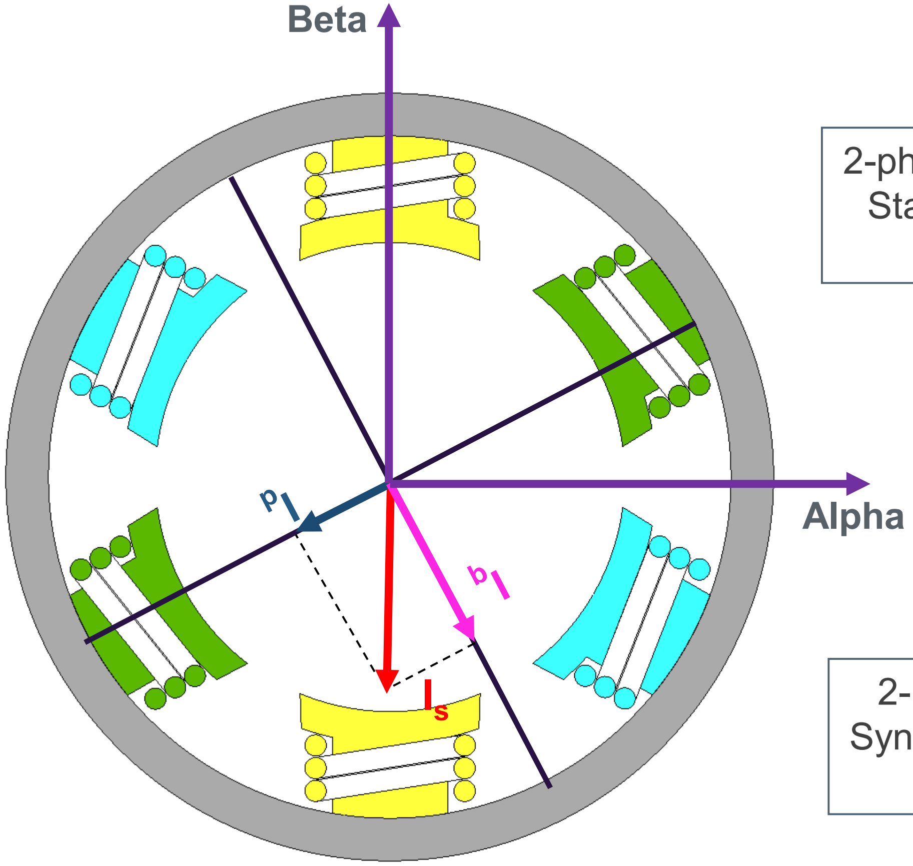


2-phase Stationary
to 2-phase Synchronous
(Forward Park Transform)

$$\begin{bmatrix} I_d \\ I_q \end{bmatrix} = \begin{bmatrix} \cos(\theta_e) & \sin(\theta_e) \\ -\sin(\theta_e) & \cos(\theta_e) \end{bmatrix} \begin{bmatrix} I_{s\alpha} \\ I_{s\beta} \end{bmatrix}$$

FOC – Synchronous Reference Frame Transformation

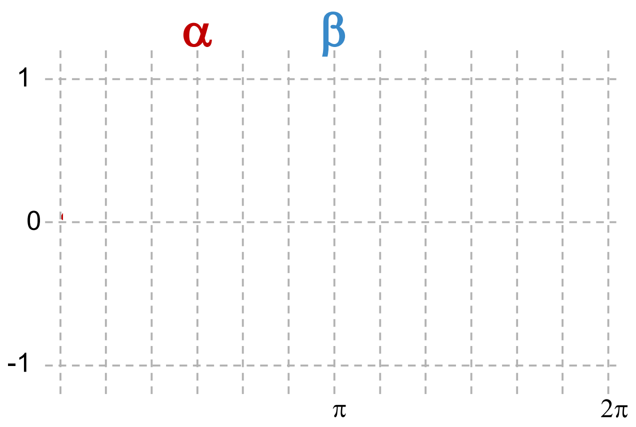
- Measured currents variation before and after Forward Park Transformation



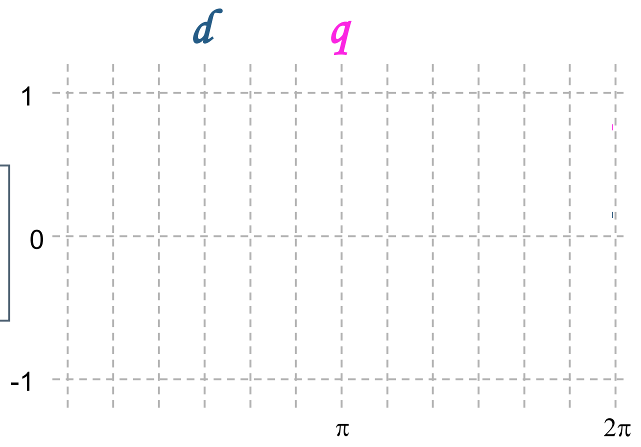
2-phase quadrature Stationary Frame (αβ)

2-phase rotating Synchronous Frame (dq)

2-phase AC currents variation

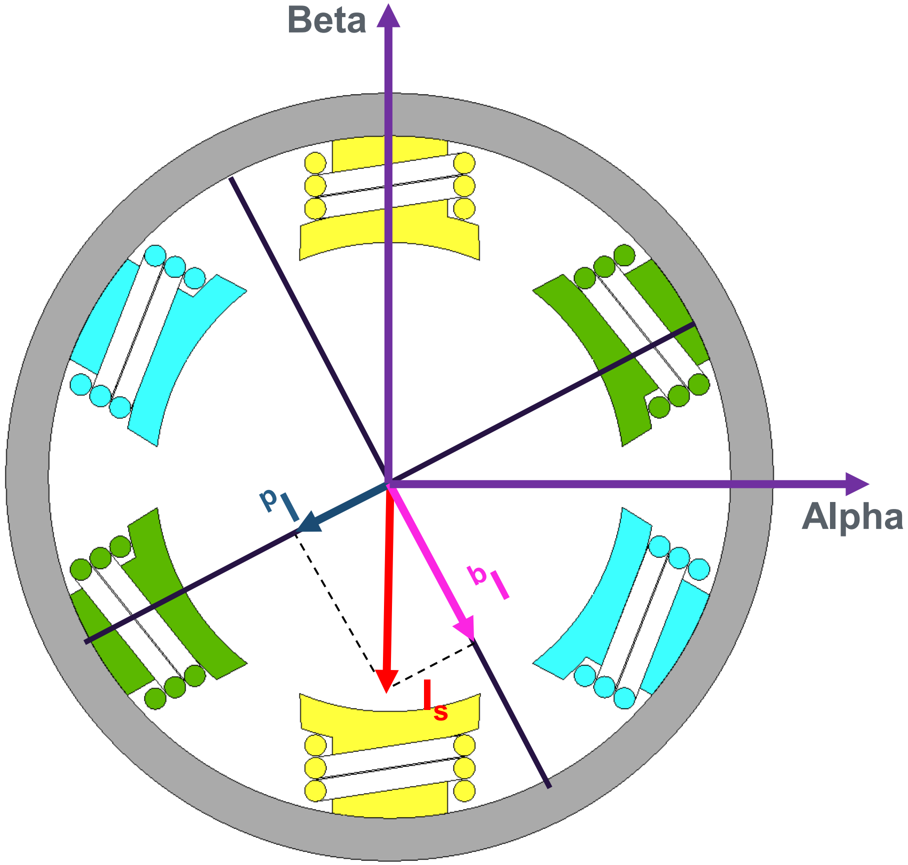


2-phase DC currents variation



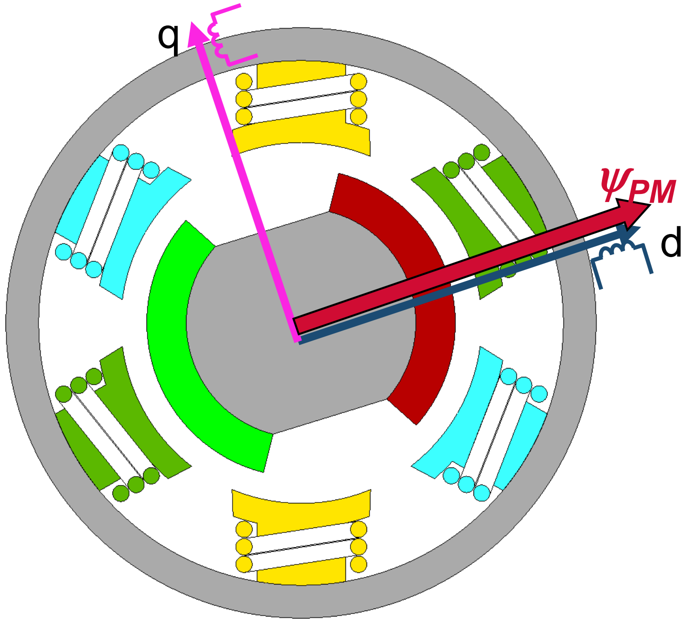
FOC – Quintessence

- In the **dq** rotating synchronous frame the space vector $\mathbf{I_s}$ is described by a constant value of d and q components;
- The position and amplitude of the space vector $\mathbf{I_s}$ can be fully controlled by two DC values.
- Control structure is simple since there are no AC values involved anymore.



FOC – Regulate PMSM Flux and Torque

Quick flashback from M2 module



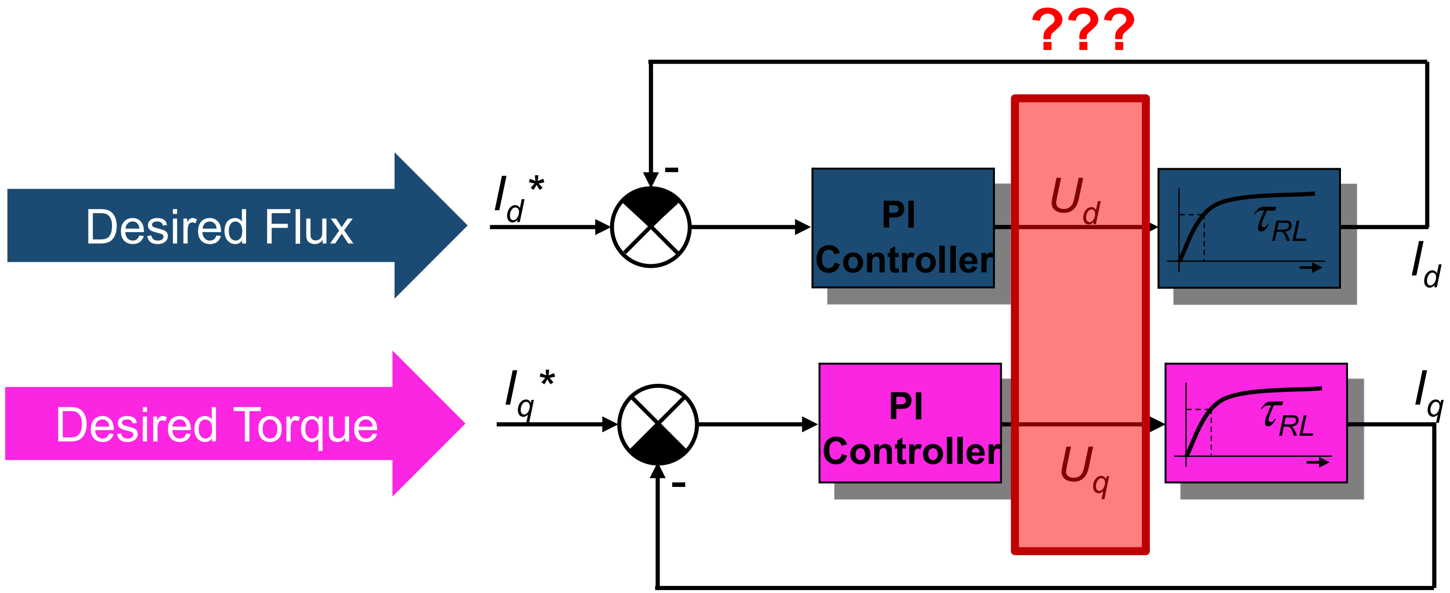
Stator voltage equations

$$\begin{bmatrix} u_d \\ u_q \end{bmatrix} = R_s \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} s & \omega_e \\ -\omega_e & s \end{bmatrix} \begin{bmatrix} \psi_d \\ \psi_q \end{bmatrix}$$

Stator linkage flux

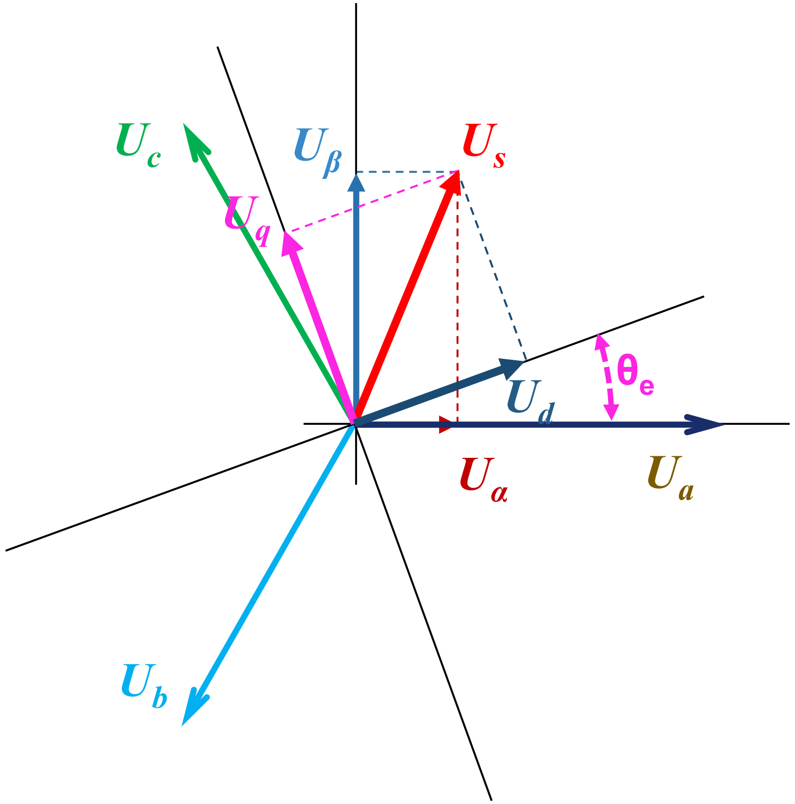
$$\begin{bmatrix} \psi_d \\ \psi_q \end{bmatrix} = \begin{bmatrix} L_d & 0 \\ 0 & L_q \end{bmatrix} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \psi_{PM} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

- I_d and I_q can be controlled independently in the rotating synchronous frame;
- In normal conditions when the stator flux vector is perpendicular to rotor flux vector then the I_d reference is ZERO;
- I_q controls the torque of the motor;



FOC – Generate Correction Voltages

- **U$_d$** and **U$_q$** can not be applied directly to motor phases due to:
  1. Exist only in the rotating reference frame (dq)
  2. There are only 2 values and we need 3 to control a PMSM



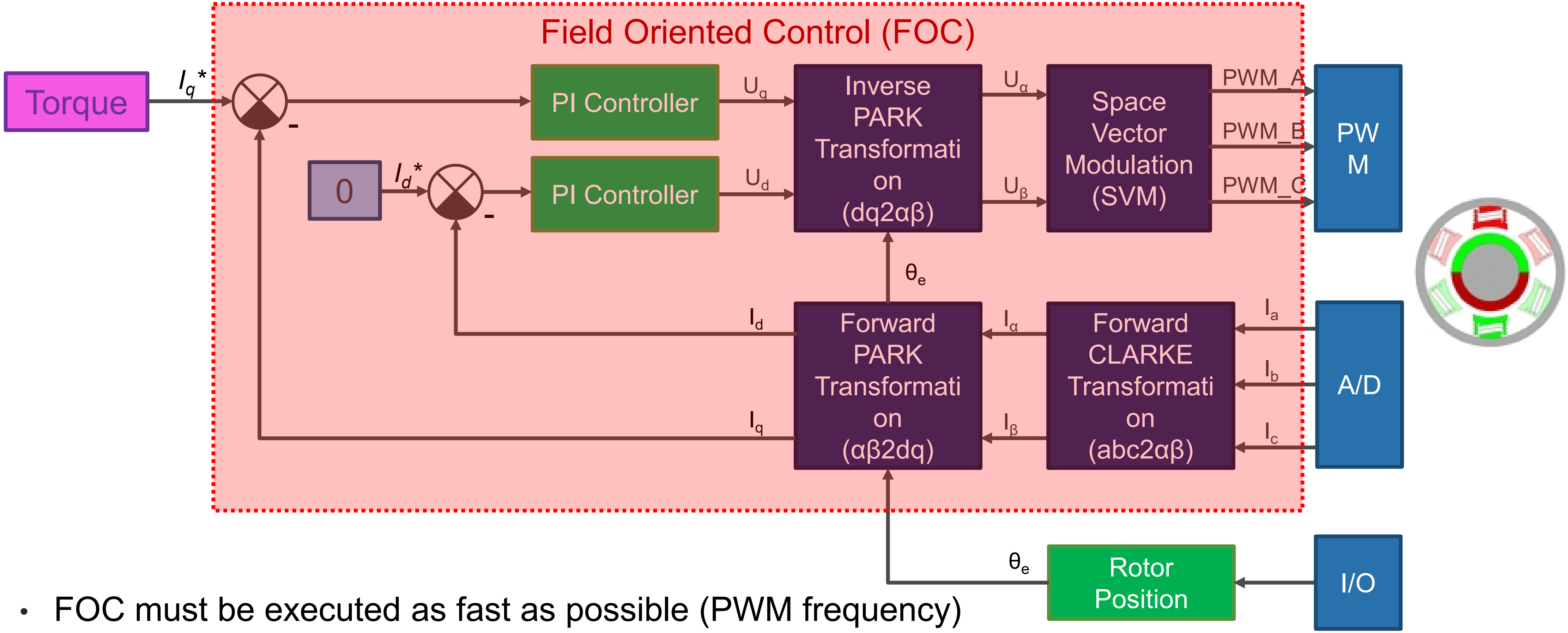
2-phase Synchronous
to 2-phase Stationary
(Inverse Park Transform)

$$\begin{bmatrix} U_\alpha \\ U_\beta \end{bmatrix} = \begin{bmatrix} \cos(\theta_e) & -\sin(\theta_e) \\ \sin(\theta_e) & \cos(\theta_e) \end{bmatrix} \begin{bmatrix} U_d \\ U_q \end{bmatrix}$$

2-phase Stationary
to 3-phase Stationary
(Inverse Clarke Transform)

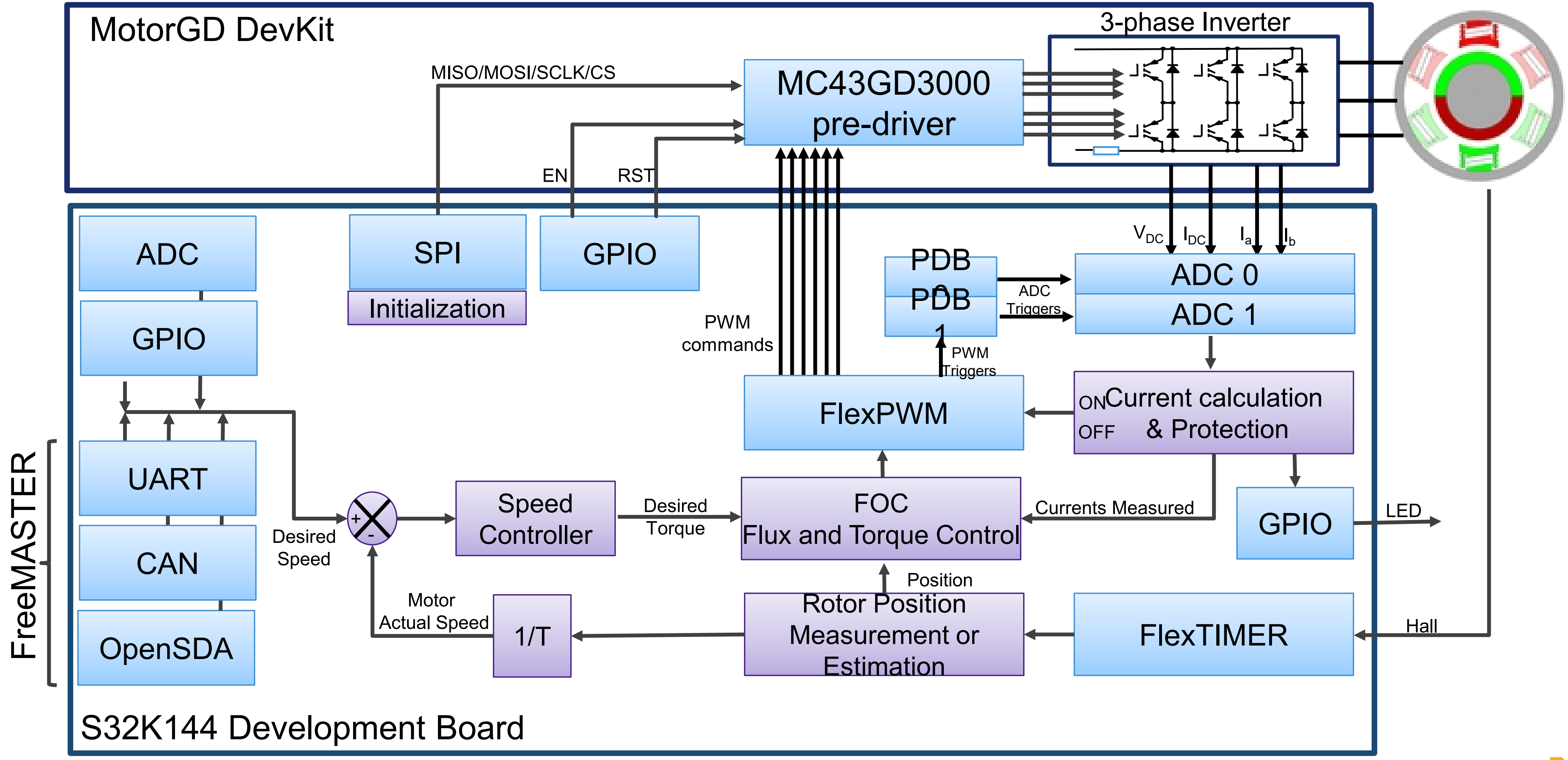
$$\begin{bmatrix} U_a \\ U_b \\ U_c \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & 0 \\ -\frac{1}{3} & \frac{1}{\sqrt{3}} \\ -\frac{1}{3} & -\frac{1}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} U_\alpha \\ U_\beta \end{bmatrix}$$

FOC – Block Diagram

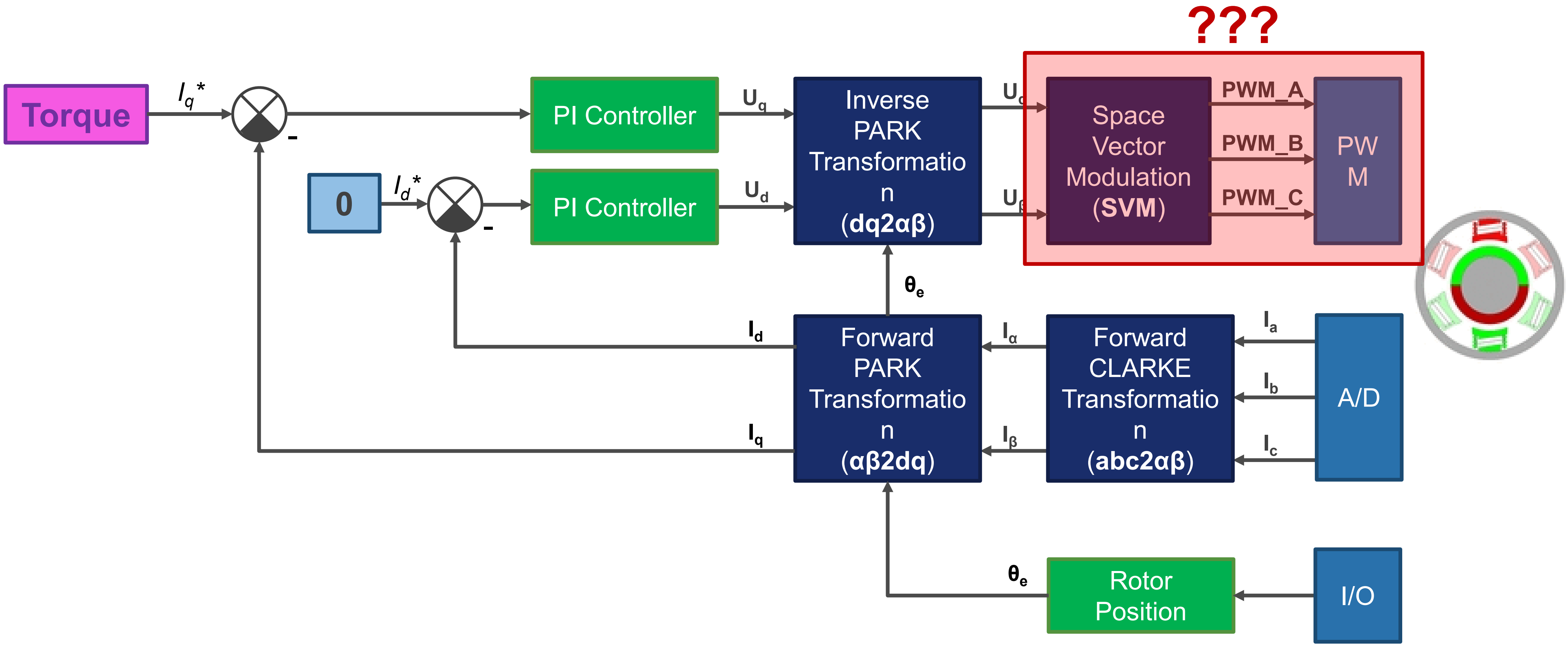


- FOC must be executed as fast as possible (PWM frequency)
- FOC implements the current control loop a.k.a FAST LOOP
- Any other loops (speed/position) that wraps over this inner loops will be referred as SLOW LOOP

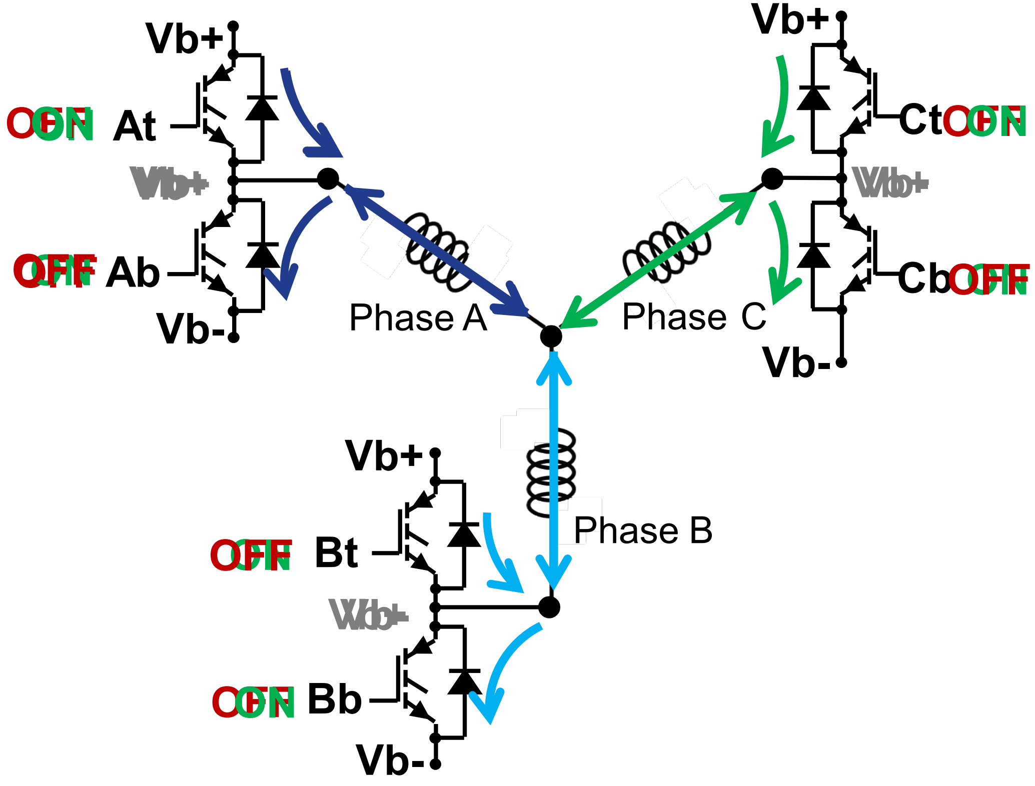
Application Mapping: SW vs. HW

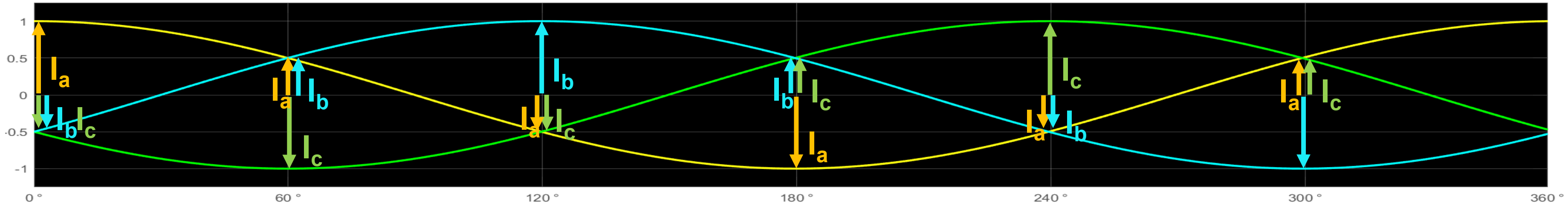


FOC – Block Diagram



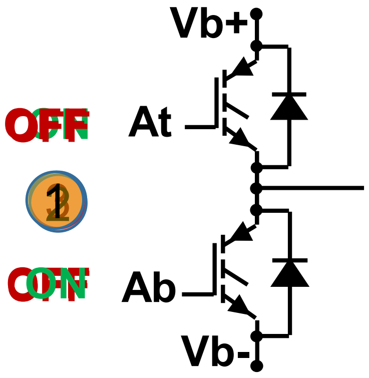
Commutation Used for PMSM

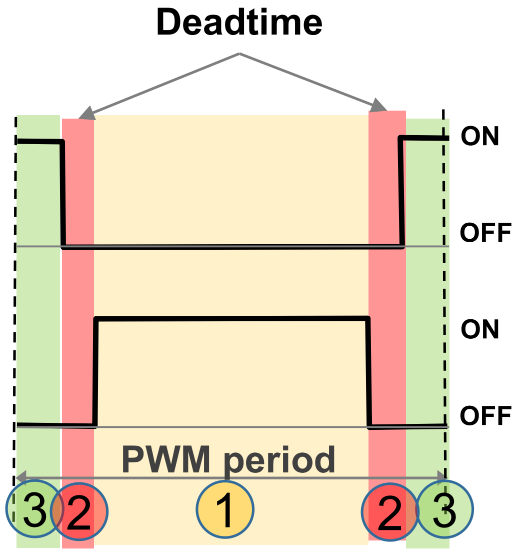




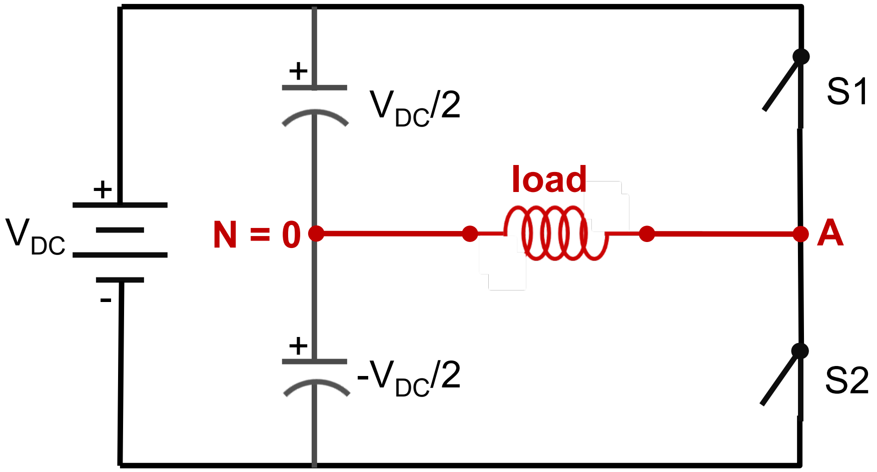
Commutation Used for PMSM (Cont'd)

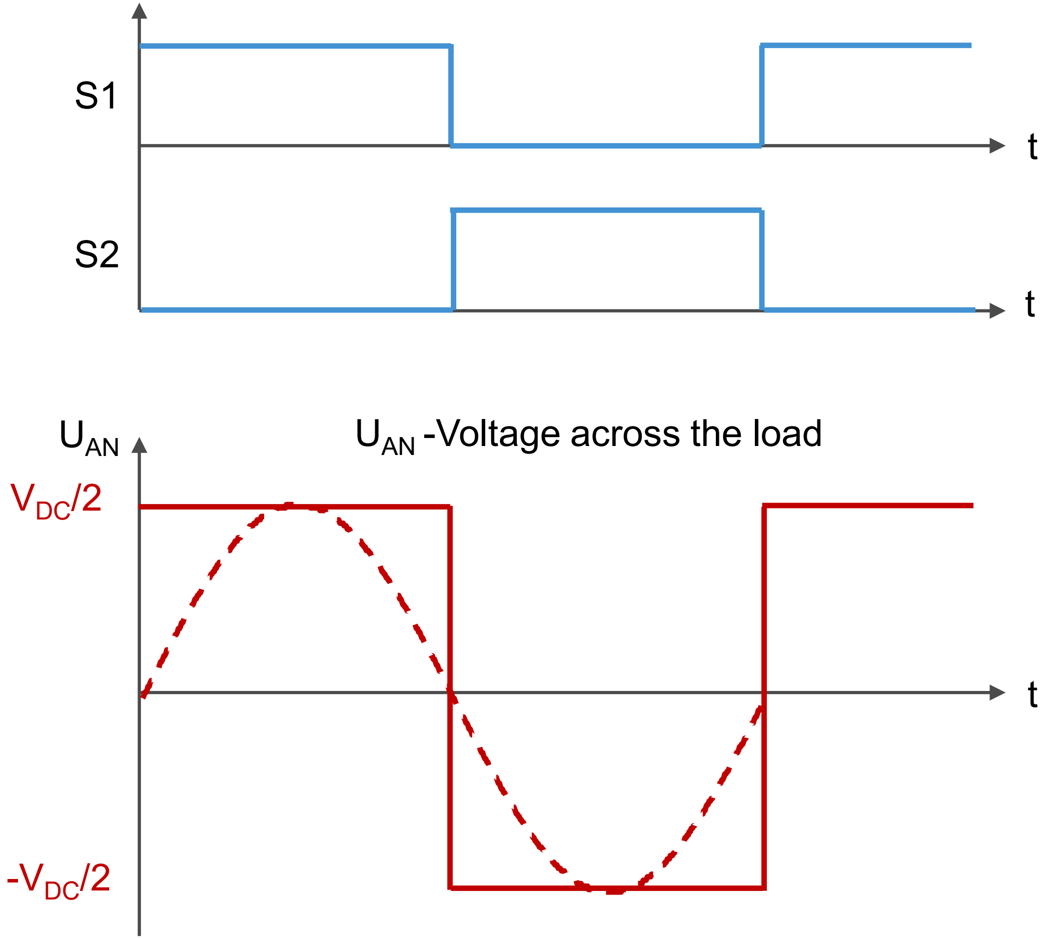
- All three inverter legs (6 transistors) are controlled at any time – transistors are either switched ON or OFF
- PWM pairs are set to complementary mode:
  - Top transistor – ON
  - Bottom transistor – OFF
  - Or vice versa
  - Deadtime is inserted to protect inverter against short circuit





How to Generate a Sinusoidal Waveform Using PWM

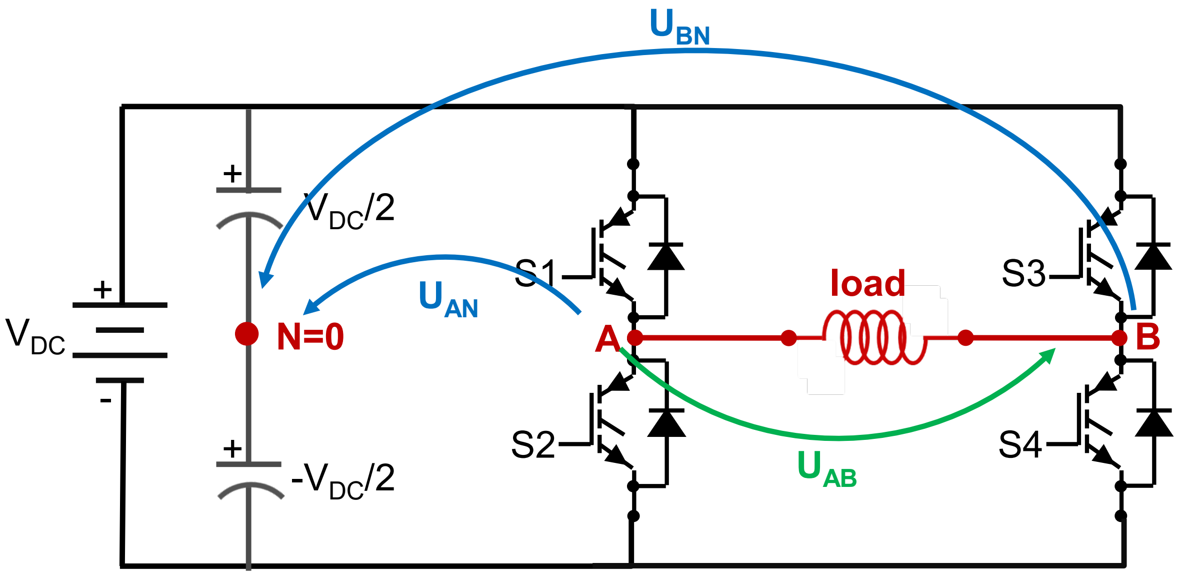




The Sine-Wave amplitude is ONLY half of the V_{DC}. The is a better way!

Single-Phase Inverter

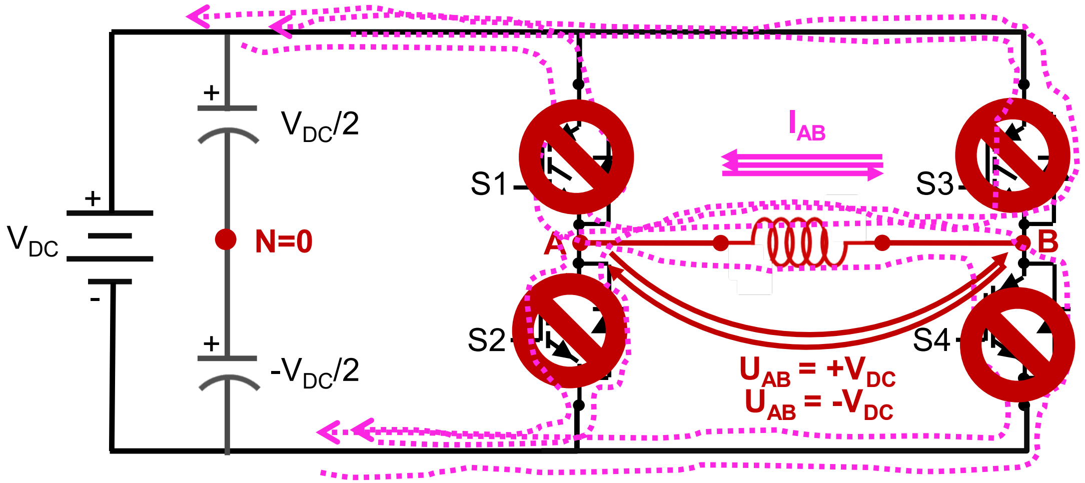
- Has 2 legs that allows 4 main combinations for the top switches
- Allows full V_{DC} voltage to be applied on the load



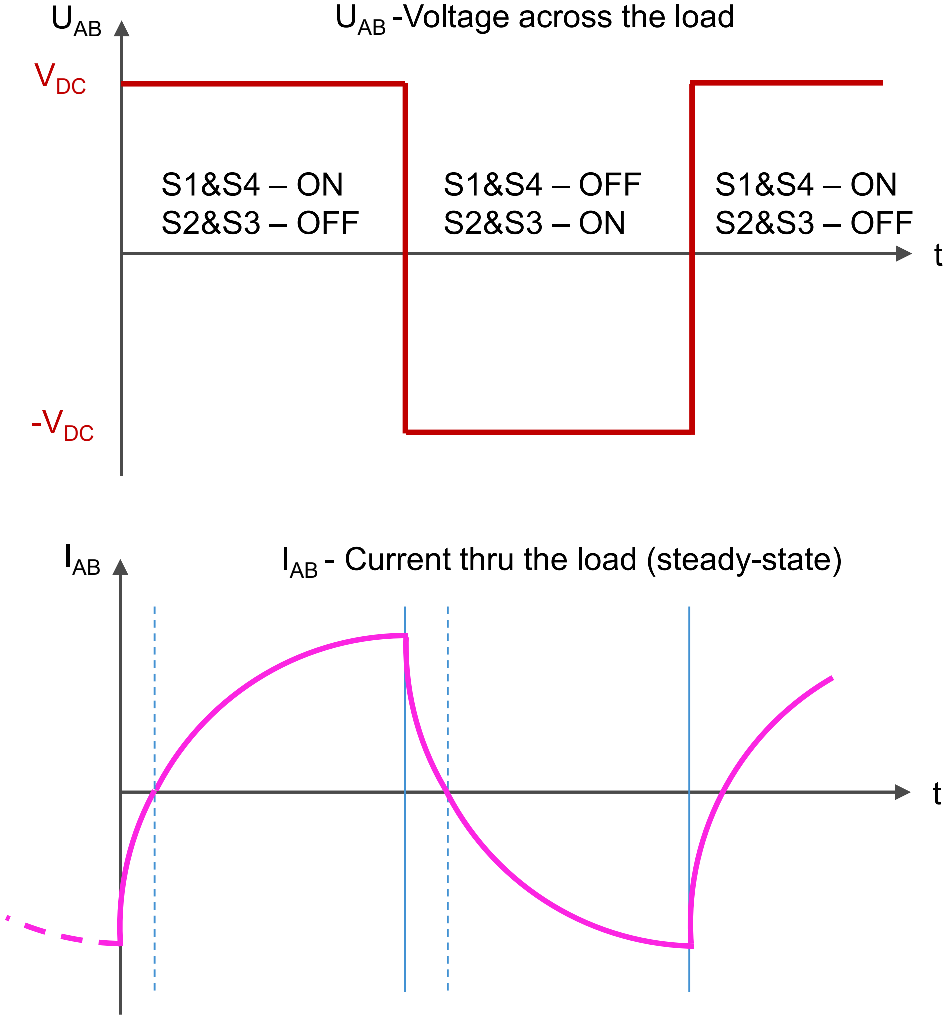
S1	S3	U_{AN}	U_{BN}	U_{AB}
OFF	OFF	$-V_{DC}/2$	$-V_{DC}/2$	0
OFF	ON	$-V_{DC}/2$	$+V_{DC}/2$	$-V_{DC}$
ON	OFF	$+V_{DC}/2$	$-V_{DC}/2$	$+V_{DC}$
ON	ON	$+V_{DC}/2$	$+V_{DC}/2$	0

- BIPOLAR Modulation -> Load Voltage changes between $–V_{DC}$ and $+V_{DC}$
- UNIPOLAR Modulation -> Load Voltage changes between 0 and $+V_{DC}$ or between 0 and $-V_{DC}$

Single-Phase Inverter – Bipolar PWM Modulation

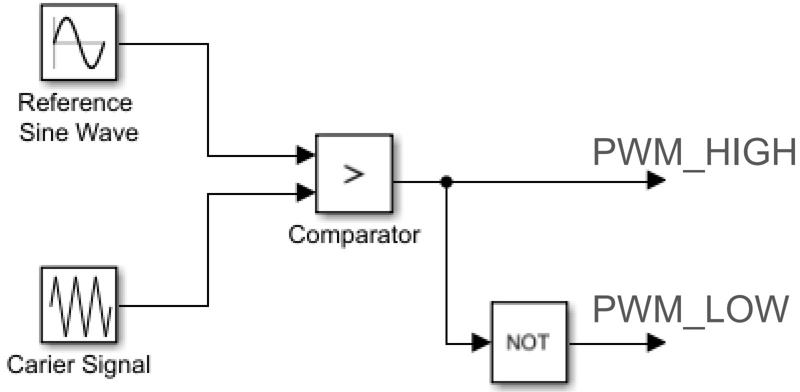


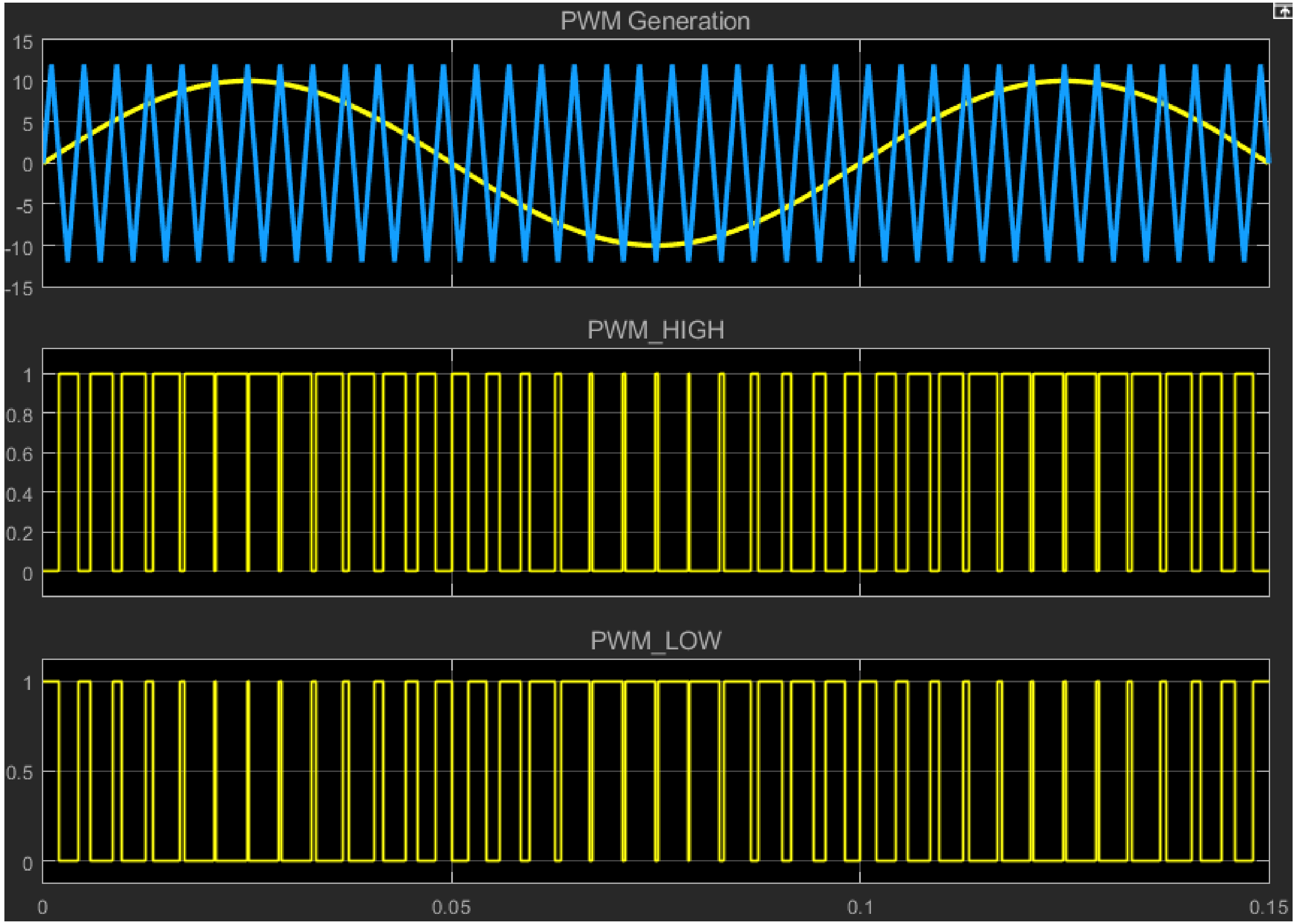
Increasing switching frequency and applying PWM make possible to control the U_{AB} rms and frequency of output voltage



Bipolar PWM Modulation for Single-Phase Use Case

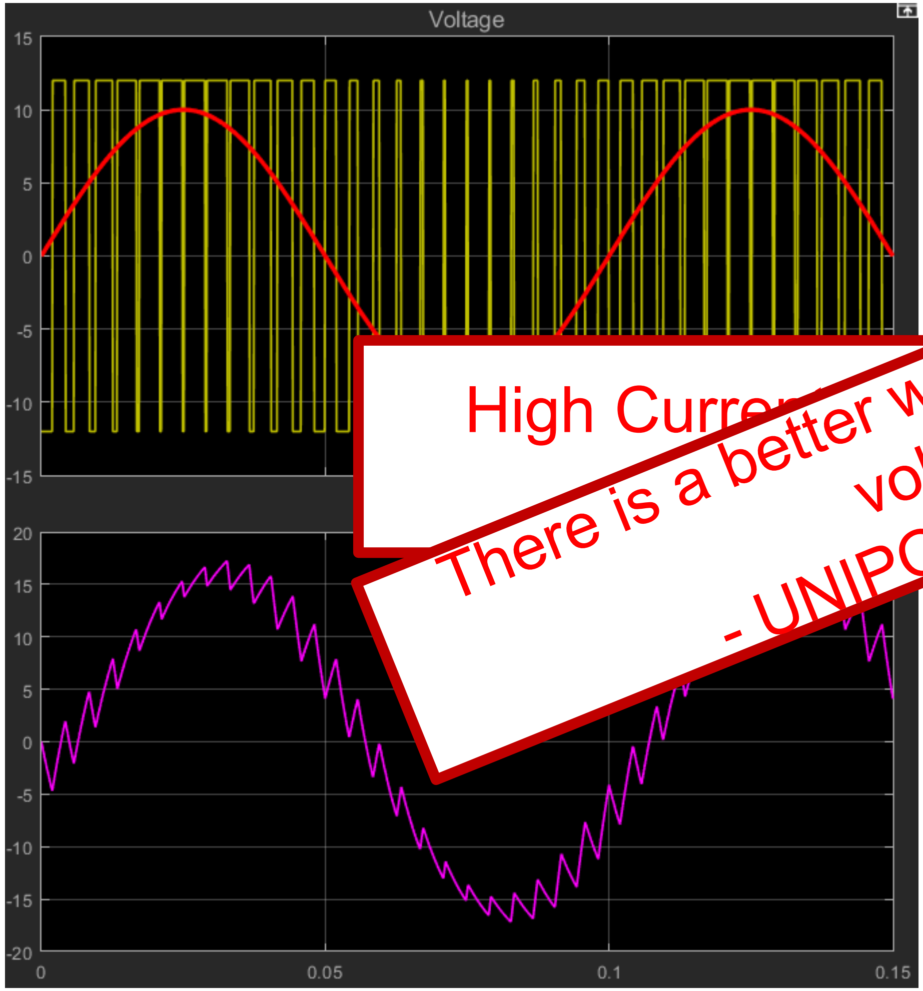
- Simple logic to generate the PWM commands for transistor's gate
  - 1kHz carrier sawtooth controls the switching frequency
  - 10Hz@10V reference sine-wave



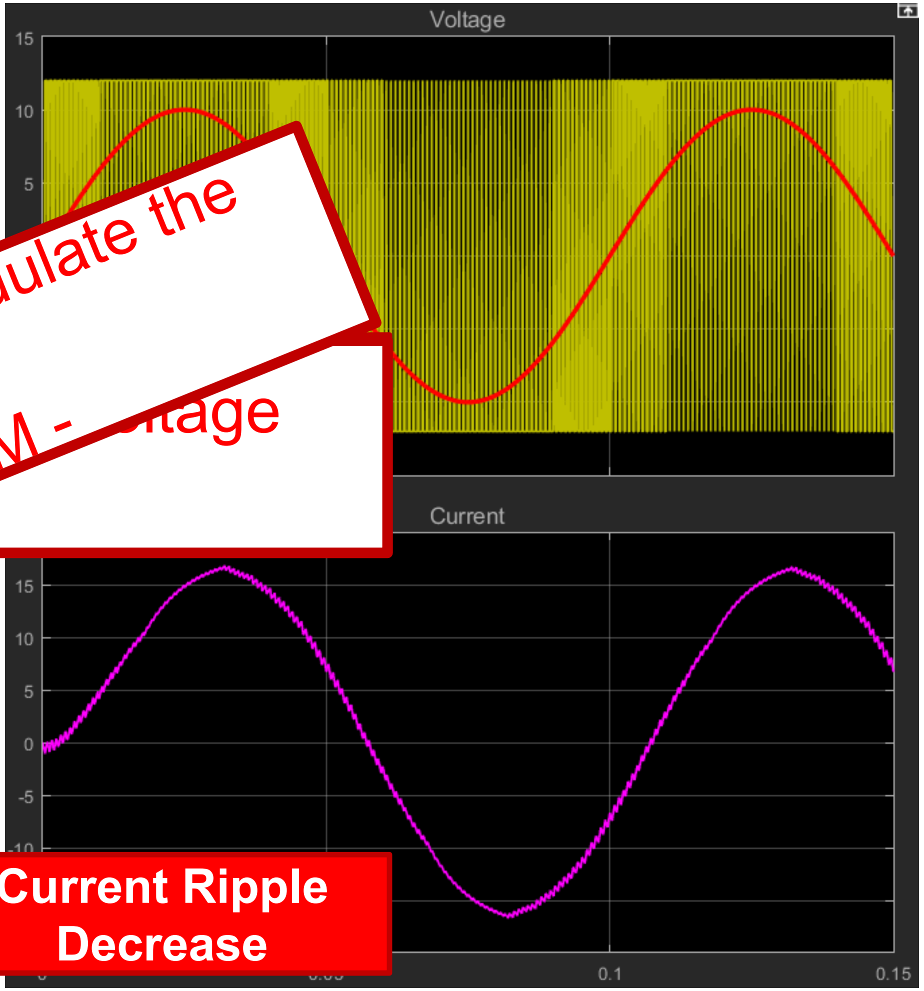


Bipolar PWM Modulation – Effect of Switching Frequency

Switching Frequency: **1kHz**



Switching Frequency: **5kHz**



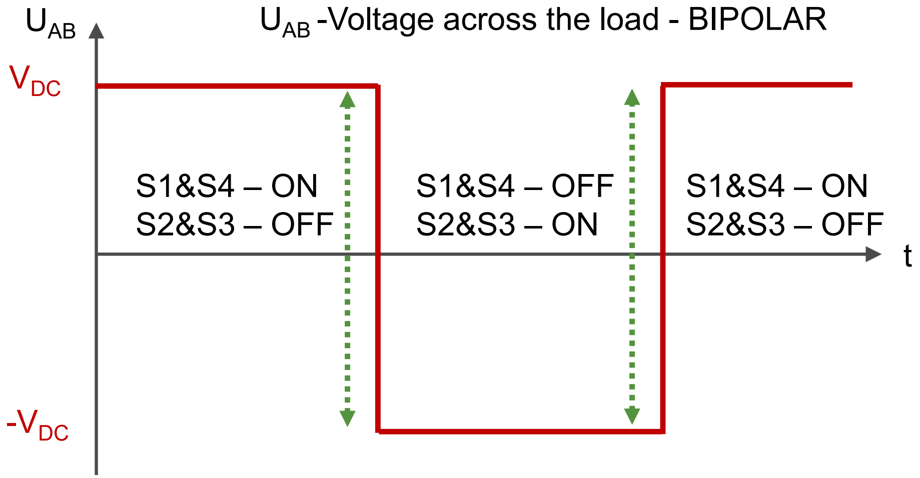
High Curren oltage

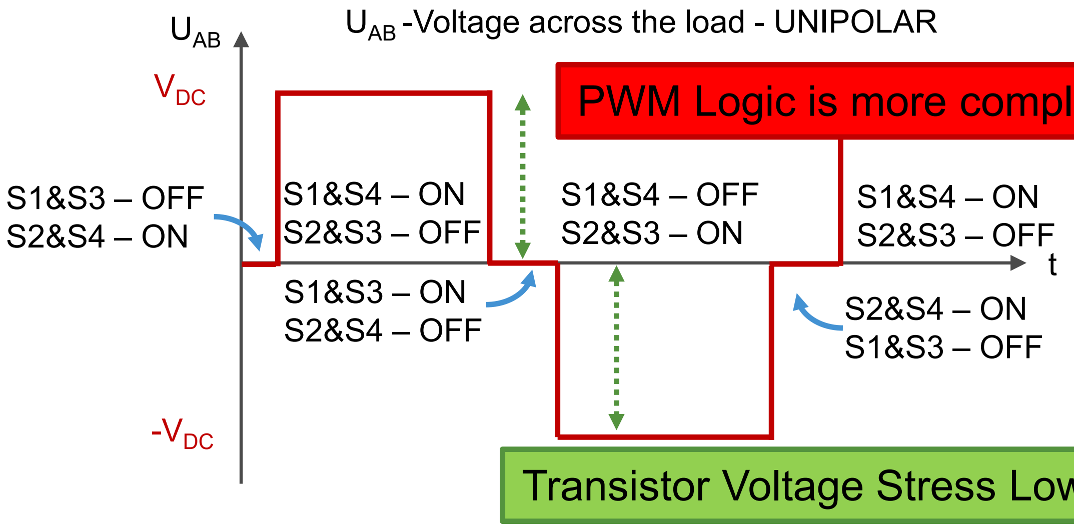
There is a better way to modulate the voltage:

- UNIPOLAR PWM -

Current Ripple Decrease

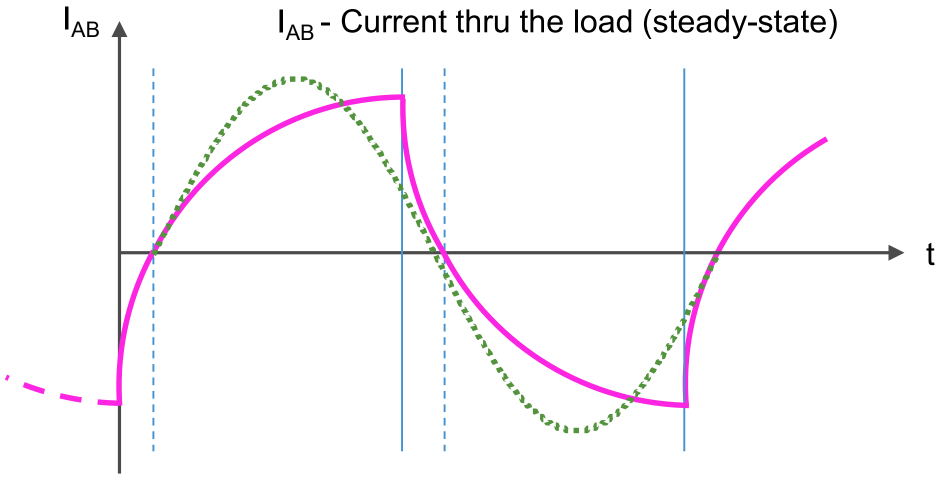
Bipolar vs. Unipolar PWM Modulation

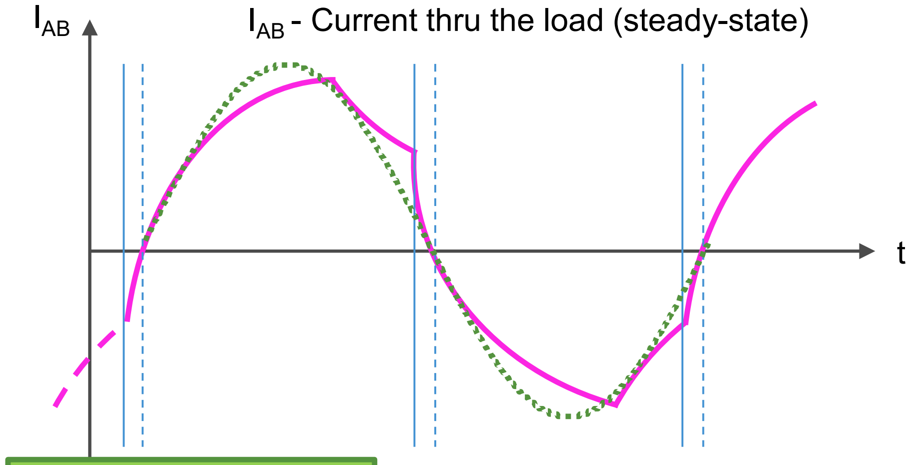




PWM Logic is more complex

Transistor Voltage Stress Lower

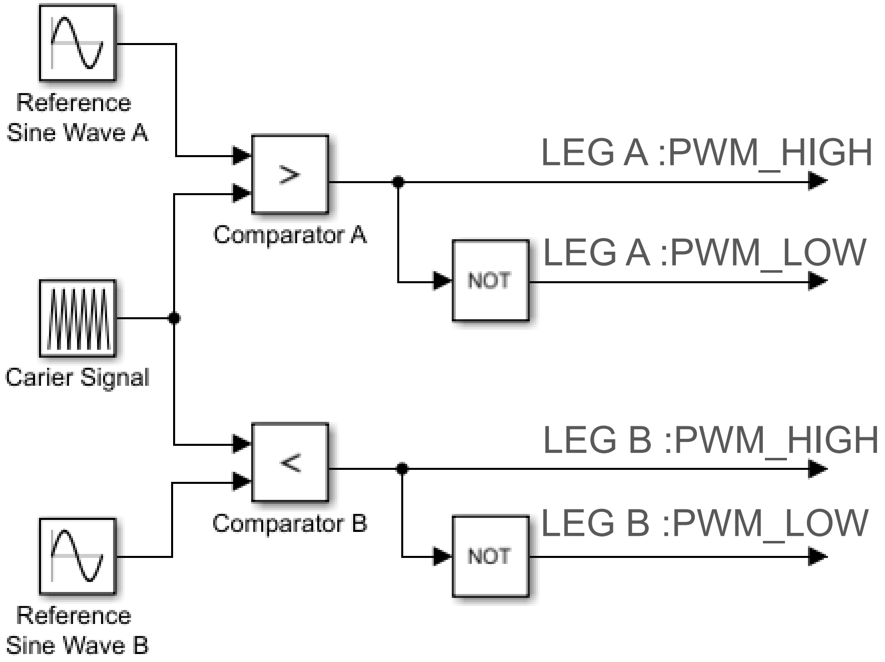


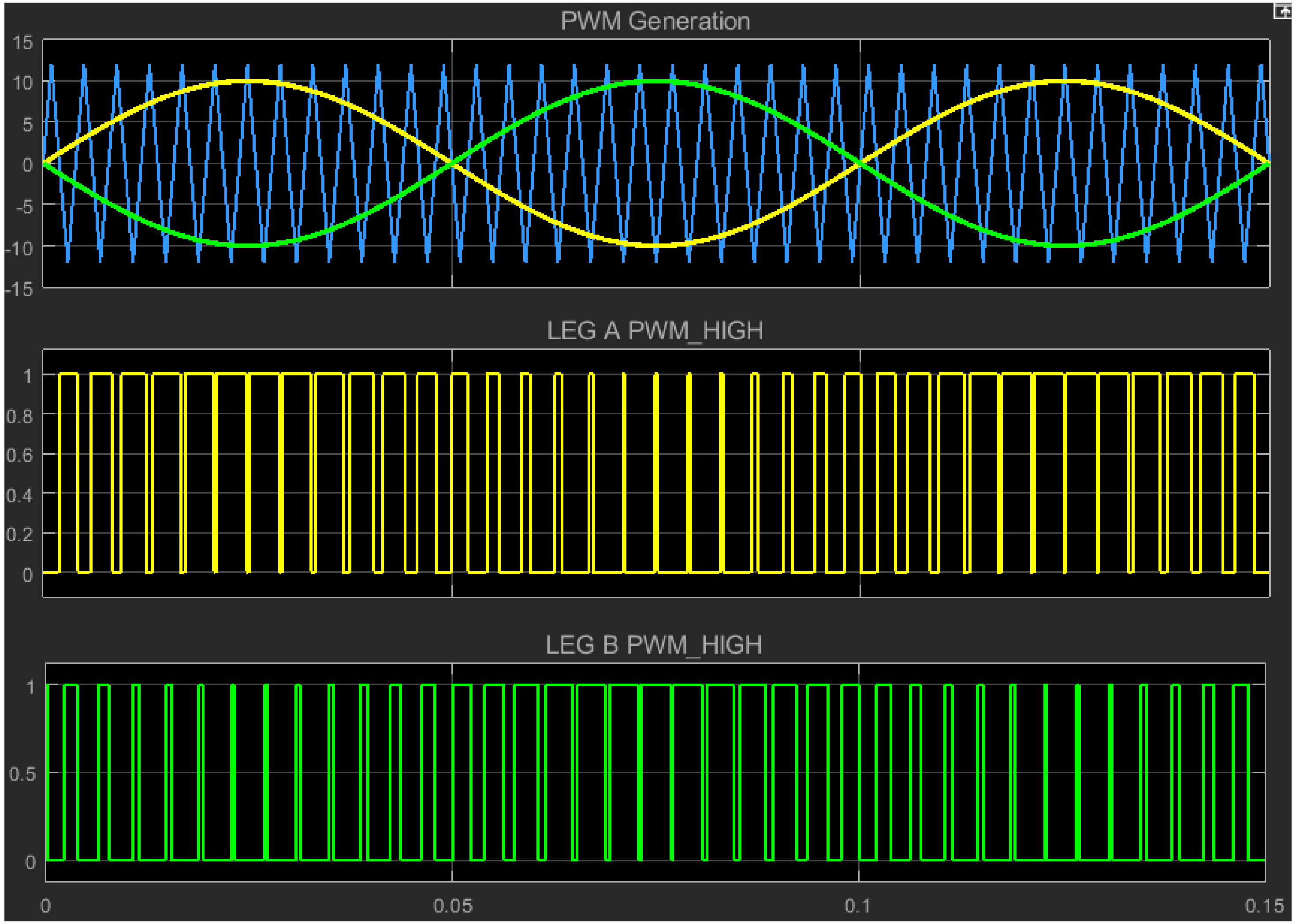


Smooth Current

Unipolar PWM Modulation for Single-Phase Use Case

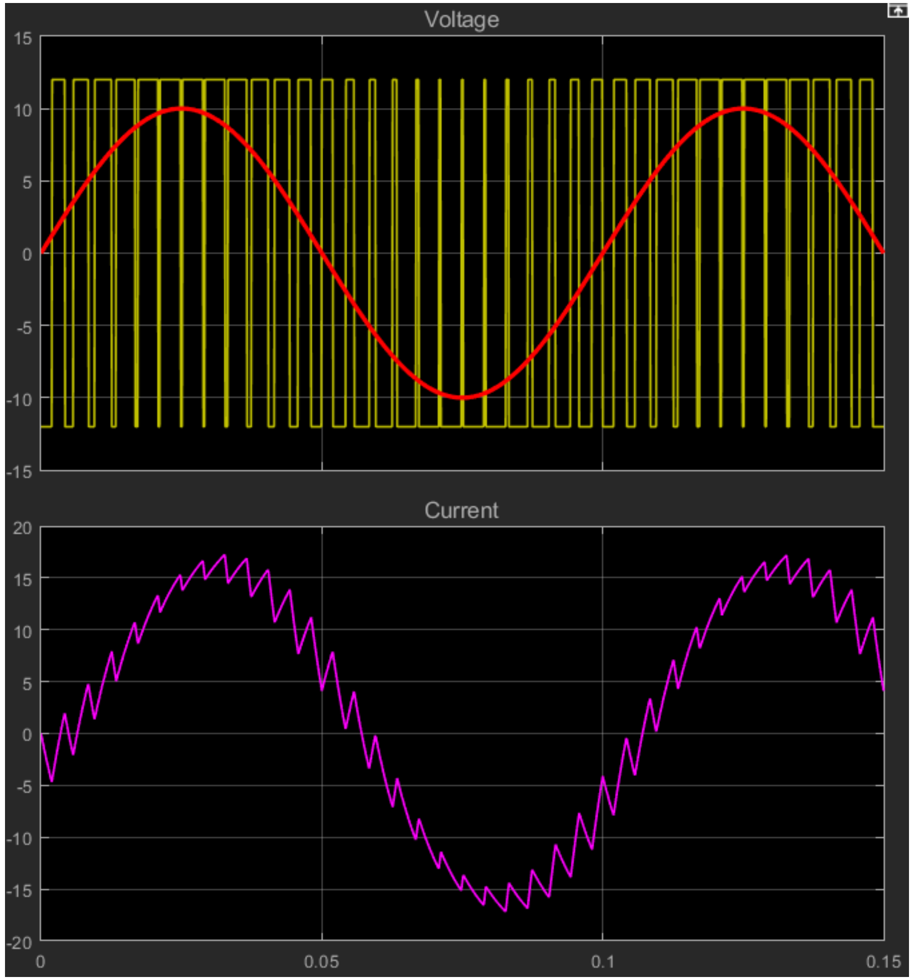
- Simple logic to generate the PWM commands for transistor's gate
  - 1kHz carrier sawtooth controls the switching frequency
  - 2x10Hz@10V reference sine-waves phased out 180deg



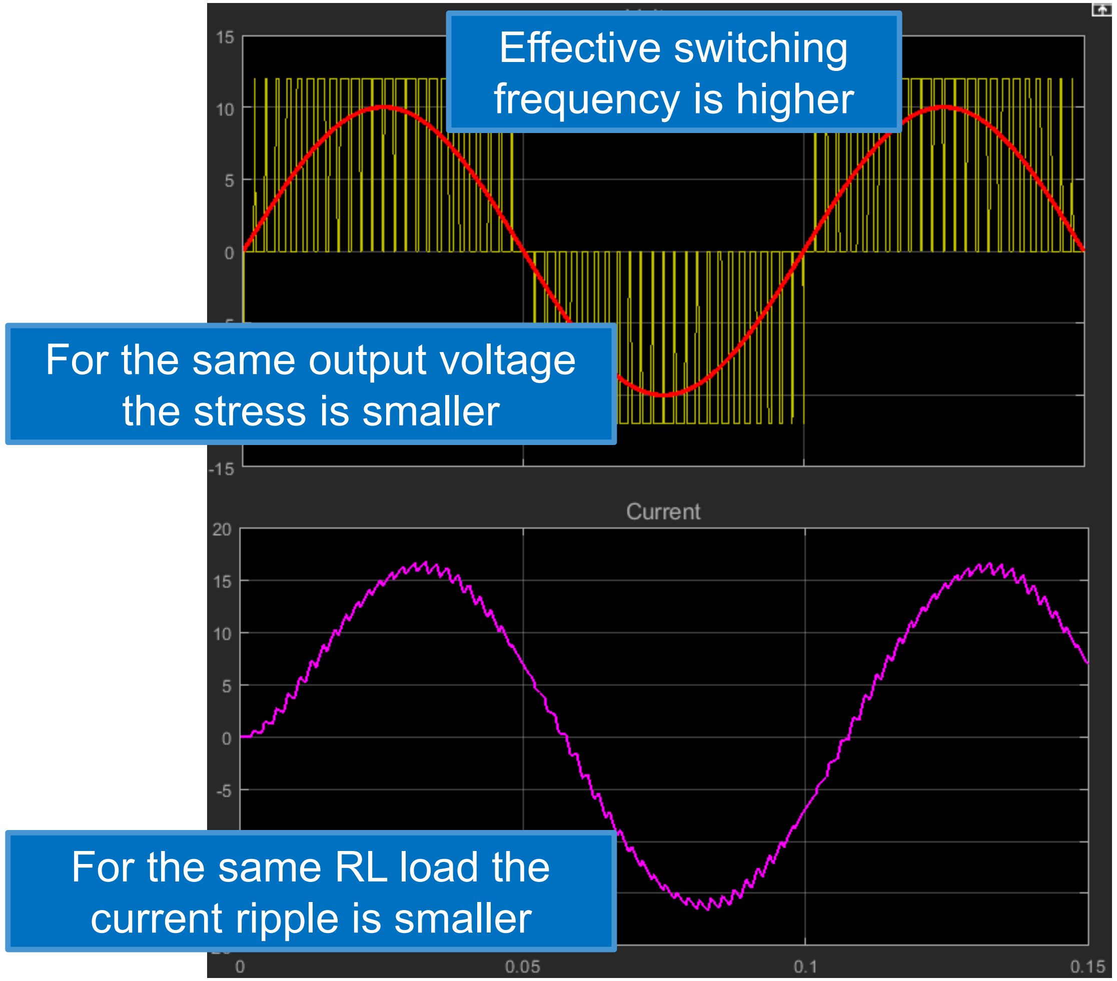


Bipolar vs. Unipolar PWM Modulation

Bipolar Switching Frequency: **1kHz**



Unipolar Switching Frequency: **1kHz**



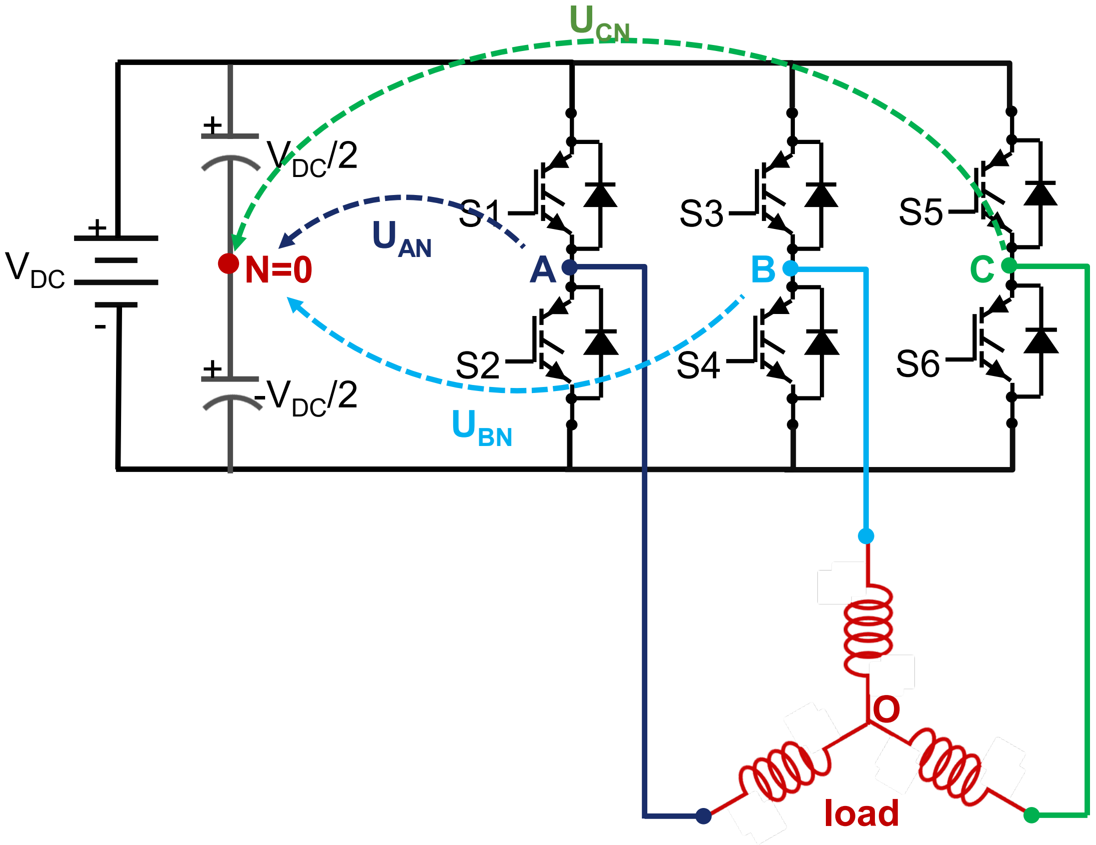
Effective switching frequency is higher

For the same output voltage the stress is smaller

For the same RL load the current ripple is smaller

3-Phase Inverter

- Has 3 legs that allows 8 main switching combinations for the top switches/transistors
- PWM pairs are set to complementary mode: S1/S2, S3/S4 and S5/S6



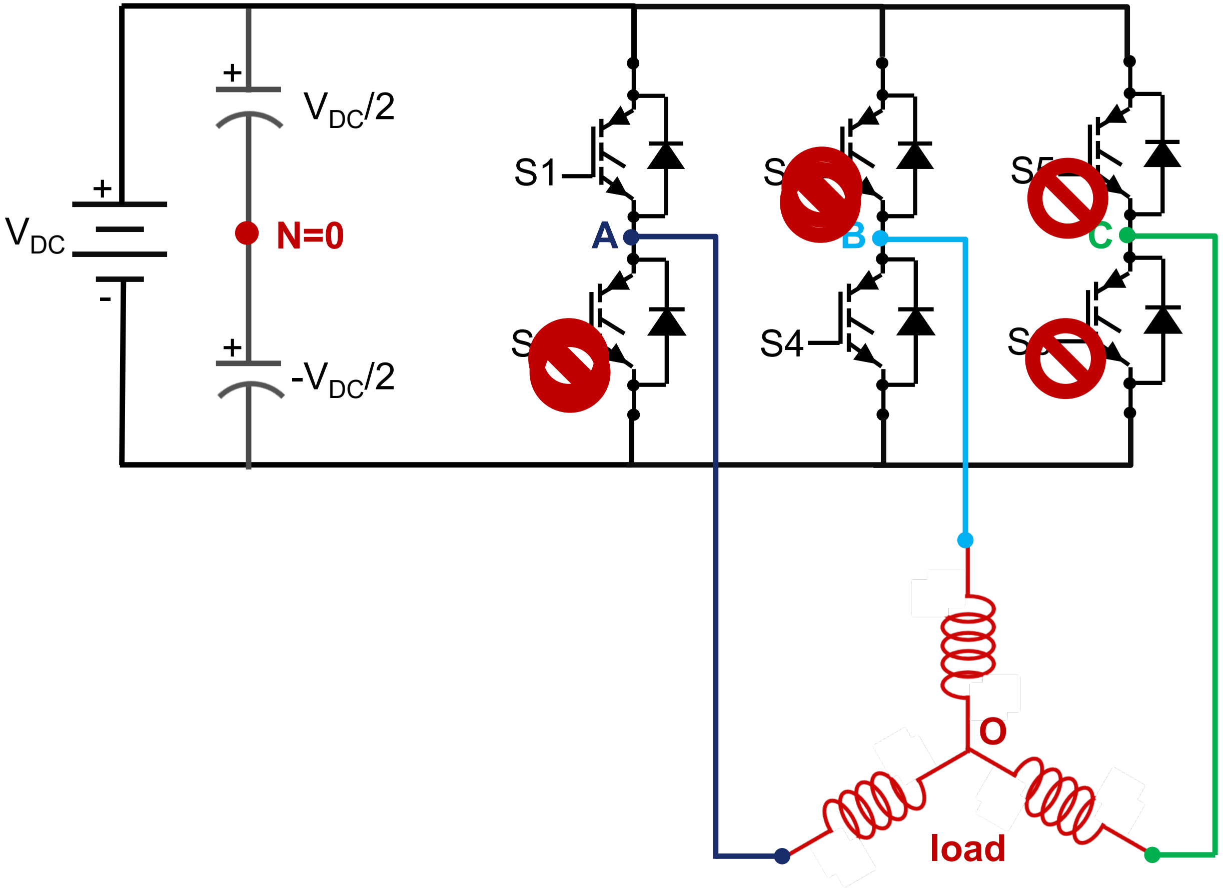
S1	S3	S5	U_{AN}	U_{BN}	U_{CN}
OFF	OFF	OFF	$-V_{DC}/2$	$-V_{DC}/2$	$-V_{DC}/2$
OFF	OFF	ON	$-V_{DC}/2$	$-V_{DC}/2$	$+V_{DC}/2$
OFF	ON	OFF	$-V_{DC}/2$	$+V_{DC}/2$	$-V_{DC}/2$
OFF	ON	ON	$-V_{DC}/2$	$+V_{DC}/2$	$+V_{DC}/2$
ON	OFF	OFF	$+V_{DC}/2$	$-V_{DC}/2$	$-V_{DC}/2$
ON	OFF	ON	$+V_{DC}/2$	$-V_{DC}/2$	$+V_{DC}/2$
ON	ON	OFF	$+V_{DC}/2$	$+V_{DC}/2$	$-V_{DC}/2$
ON	ON	ON	$+V_{DC}/2$	$+V_{DC}/2$	$+V_{DC}/2$

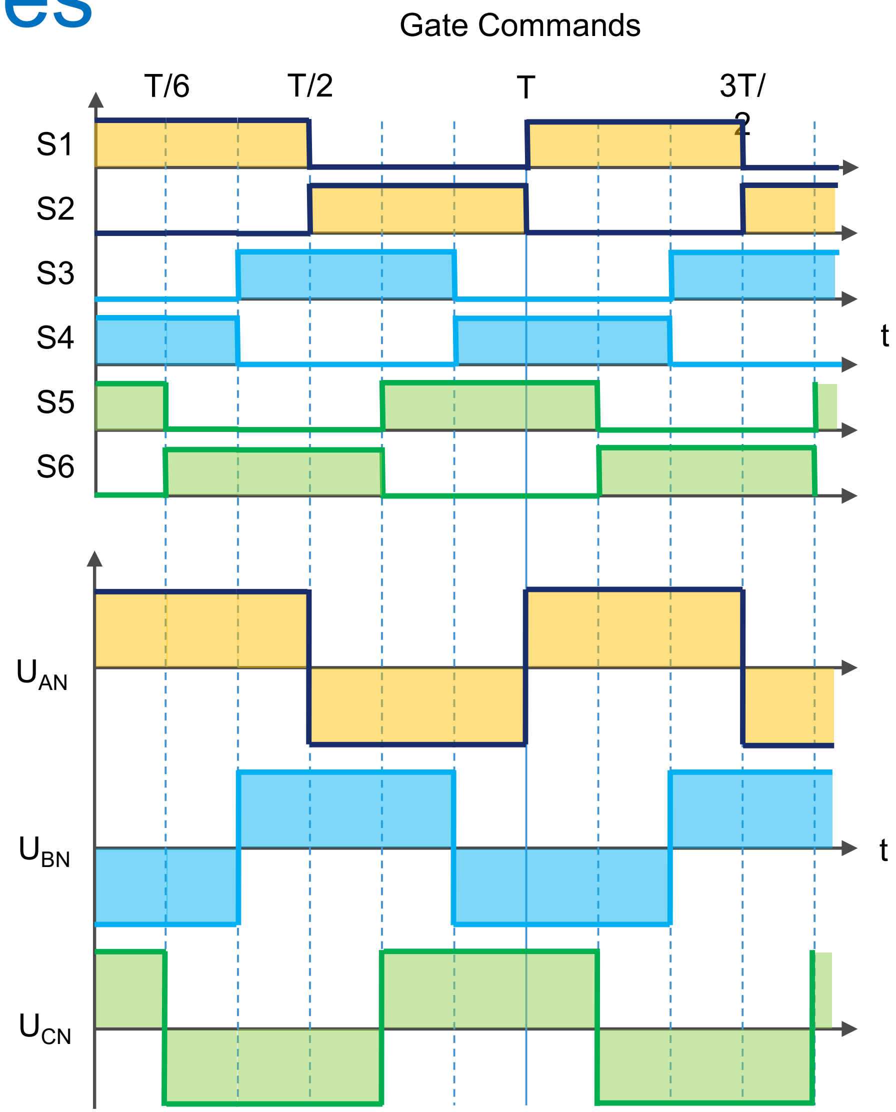
3-Phase Inverter – Terminal Voltages

- Terminal Voltage varies between $-V_{DC}/2$ and $+V_{DC}/2$

Case 101: S1=ON, S3=OFF, S5=ON

Case 100: S1=ON, S3=OFF, S5=OFF





Terminal – Phase – Line – Common Voltages

- Phase Voltage – motor winding voltage U_A, U_B and U_C
- Common Mode Voltage - PMSM neutral U_O
- Terminal Voltage – U_{AN}, U_{BN} and U_{CN}

$$U_{AN} = U_A + U_0$$
$$U_{BN} = U_B + U_0$$
$$U_{CN} = U_C + U_0$$

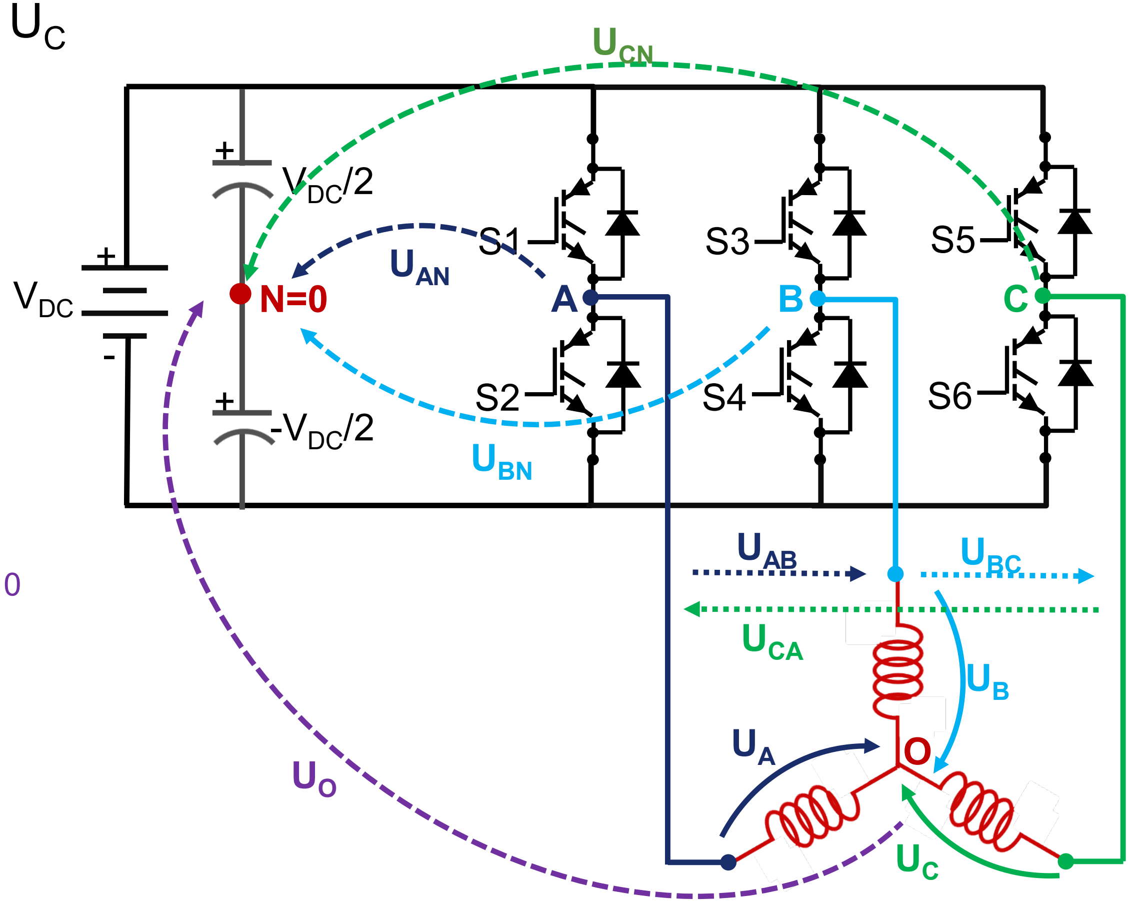
- How to compute the Common Mode Voltage ?

$$U_{AN} + U_{BN} + U_{CN} = \underbrace{(U_A + U_B + U_C)}_{\text{ZERO}} + 3{*}U_0$$

$$U_0 = (U_{AN} + U_{BN} + U_{CN}) / 3$$

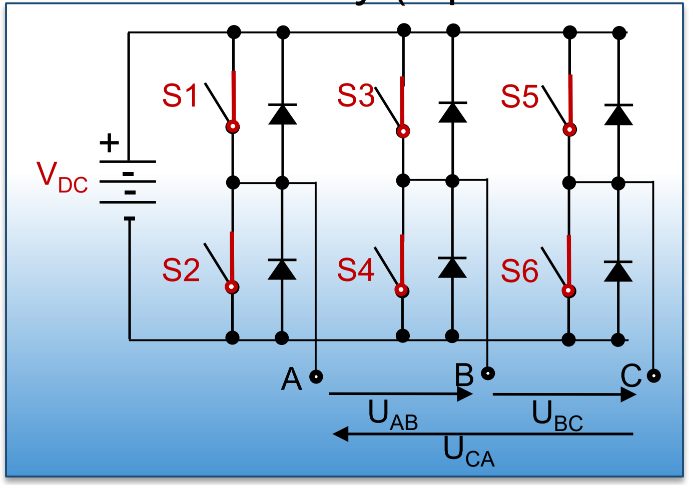
- Line Voltage – U_{AB}, U_{BC} and U_{CA}

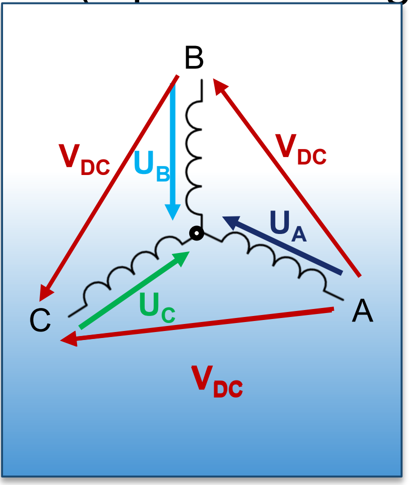
$$U_{AB} = U_A - U_B$$
$$U_{BC} = U_B - U_C$$
$$U_{CA} = U_C - U_A$$

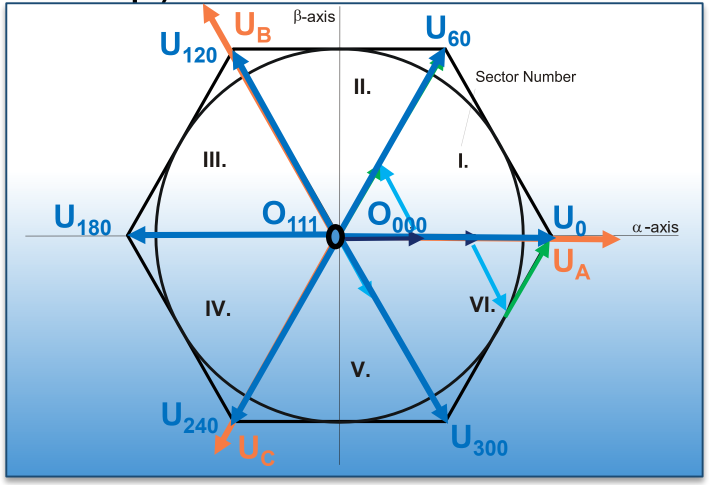


Space Vector Modulation

- A 3-phase inverter can be configured to produce 8 distinct output states (2 null and 6 active)
- Each output state produces an unique set of phase voltages that corresponds to a Voltage **Space Vector** defined in the stationary (3 phase ABC) or (2 phase orthogonal αβ) reference frames







S1	S3	S5	U_{AB}	U_{BC}	U_{CA}	U_a	U_b	U_c	Vector
0	0	0	0	0	0	0	0	0	O_{000}
1	0	0	V_{DC}	0	$-V_{DC}$	$2/3\ V_{DC}$	$-1/3\ V_{DC}$	$-1/3\ V_{DC}$	U_0
1	1	0	0	V_{DC}	$-V_{DC}$	$1/3\ V_{DC}$	$1/3\ V_{DC}$	$-2/3\ V_{DC}$	U_{60}
0	1	0	$-V_{DC}$	V_{DC}	0	$-1/3\ V_{DC}$	$2/3\ V_{DC}$	$-1/3\ V_{DC}$	U_{120}
0	1	1	$-V_{DC}$	0	V_{DC}	$-2/3\ V_{DC}$	$1/3\ V_{DC}$	$1/3\ V_{DC}$	U_{180}
0	0	1	0	$-V_{DC}$	V_{DC}	$-1/3\ V_{DC}$	$-1/3\ V_{DC}$	$2/3\ V_{DC}$	U_{240}
1	0	1	V_{DC}	$-V_{DC}$	0	$1/3\ V_{DC}$	$-2/3\ V_{DC}$	$1/3\ V_{DC}$	U_{300}
1	1	1	0	0	0	0	0	0	O_{111}

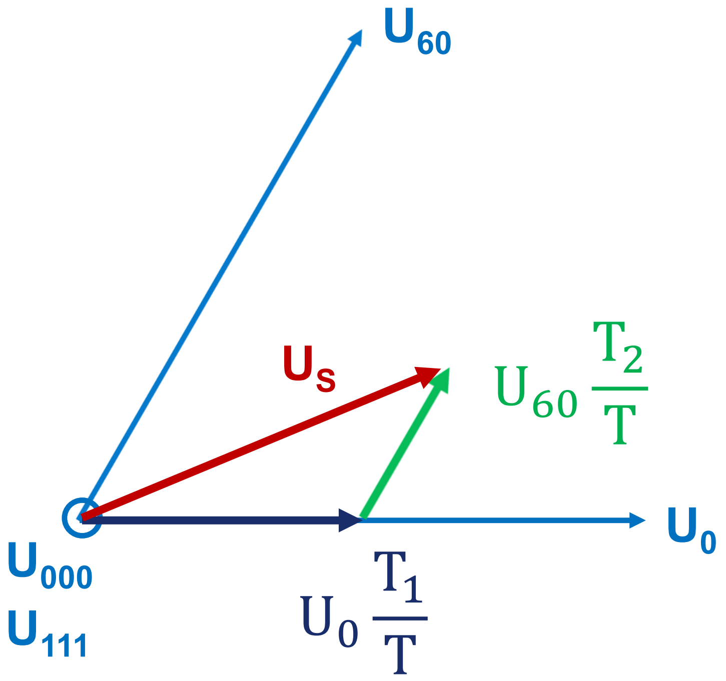
“The Essence” of Space Vector Modulation

- SVM allows creation of any reference voltage vector using a only 2 adjacent vectors and one convenient selected zero vector (000) or (111) to minimize the transistor commutations

- If T is the sampling period (aka PWM period) then:
  - U_0 is applied for T_1
  - U_{60} is applied for T_2

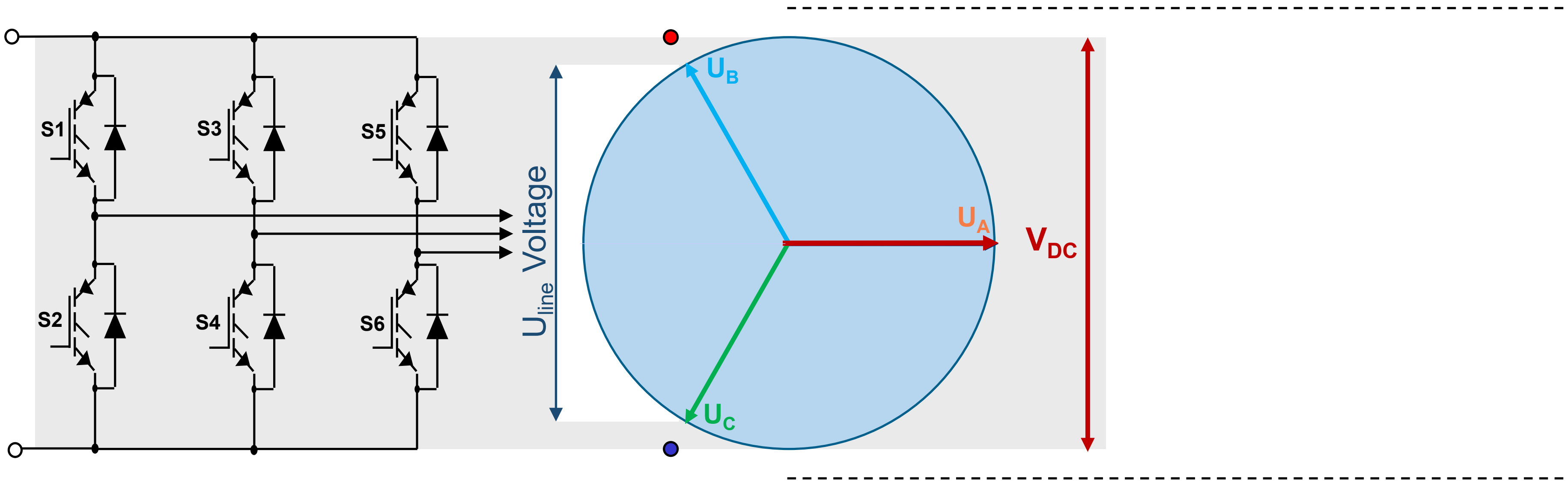
  - The zero vector is applied for the rest of the period so that the final voltage vector is obtained.

    $T_0 = T – T_1 – T_2$



Sinusoidal Modulation Limitation

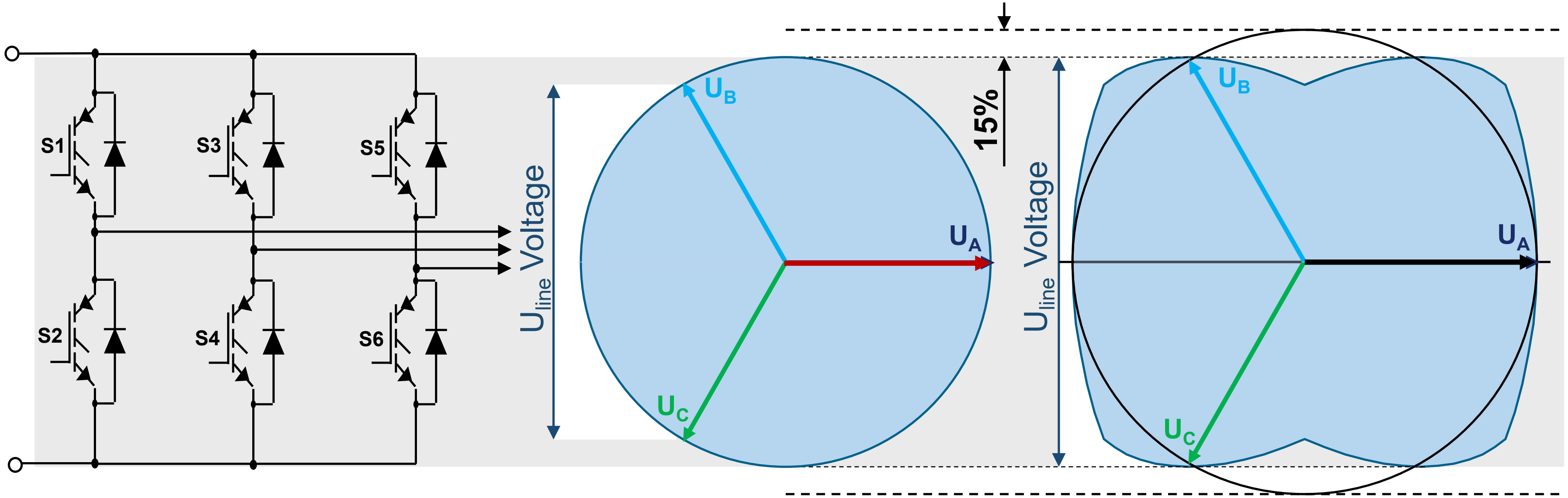
- Phase Voltage (U_A, U_B and U_C) amplitude is limited to half of the DC-bus voltage
- The Line Voltage (U_{AB}, U_{BC} and U_{CA}) is lower then the V_{DC} voltage (although such voltage can be generated between the terminals)



How should we control the inverter switches to generate full line voltage?

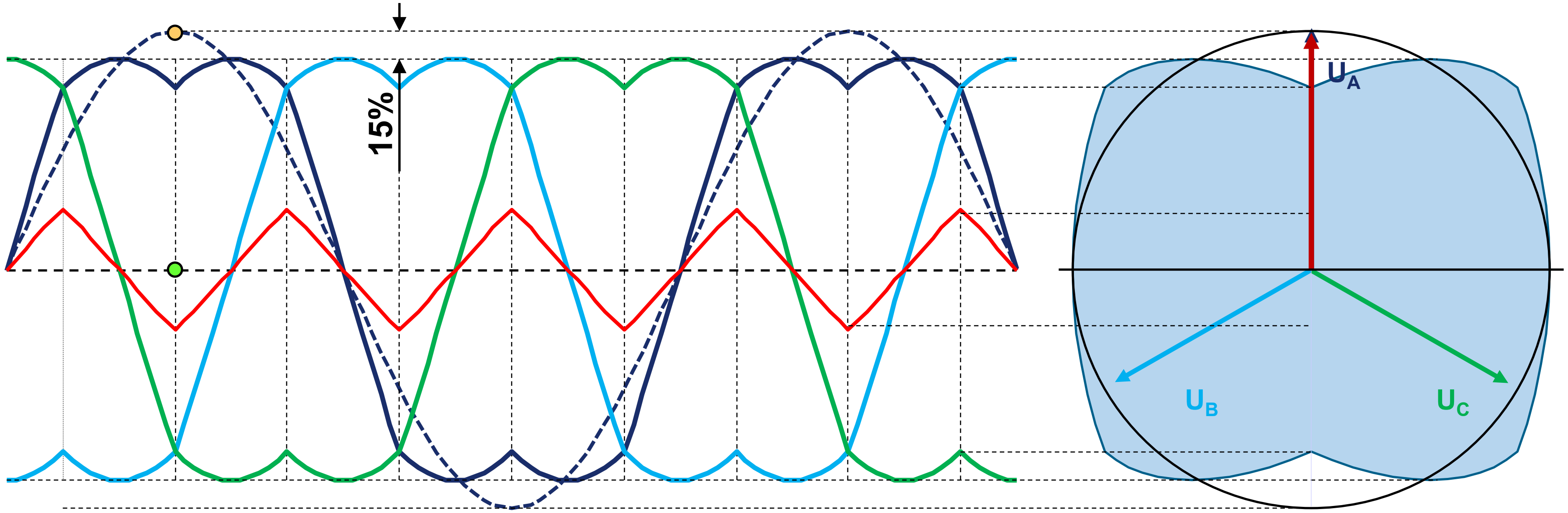
Full Line Voltage Generation

- Full phase-to-phase voltage can be generated by continuously shifting the
  3-phase voltage system
- The amplitude of the first harmonic can be then increased by 15.5%



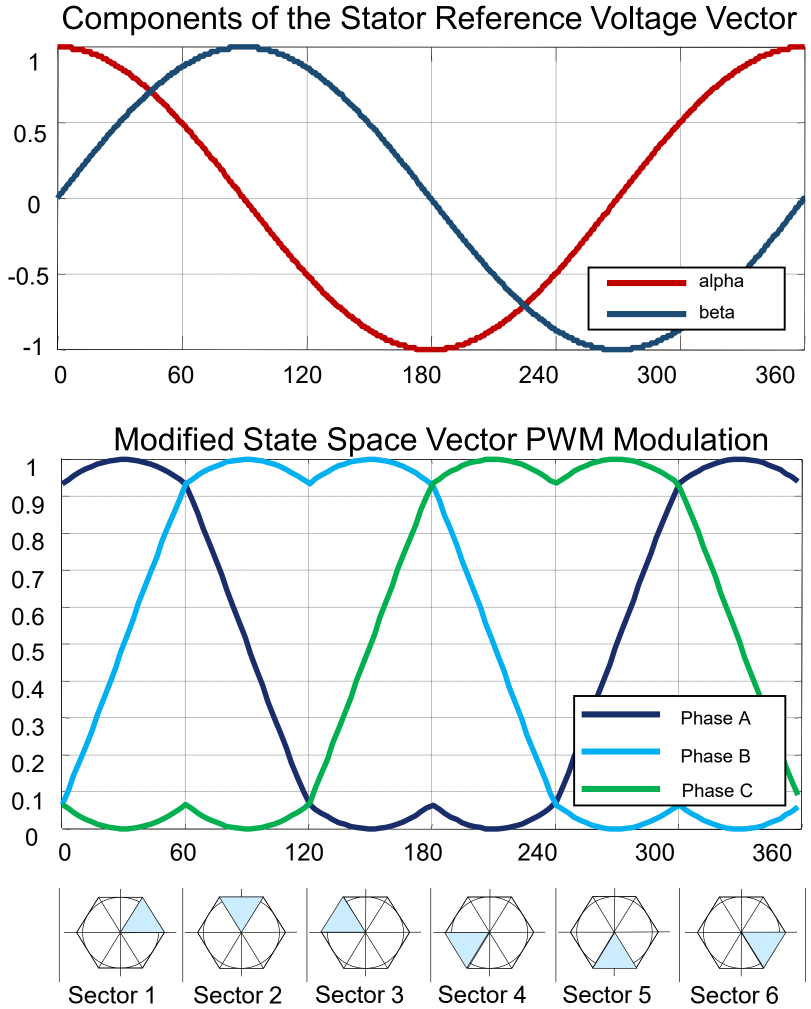
3^{rd} Harmonic Injection

- Line voltage is increased by adding the “shifting” voltage U_0
- “Shifting” voltage U_0 must be the same for all three phases, thus it can only contain 3^{rd} harmonics!



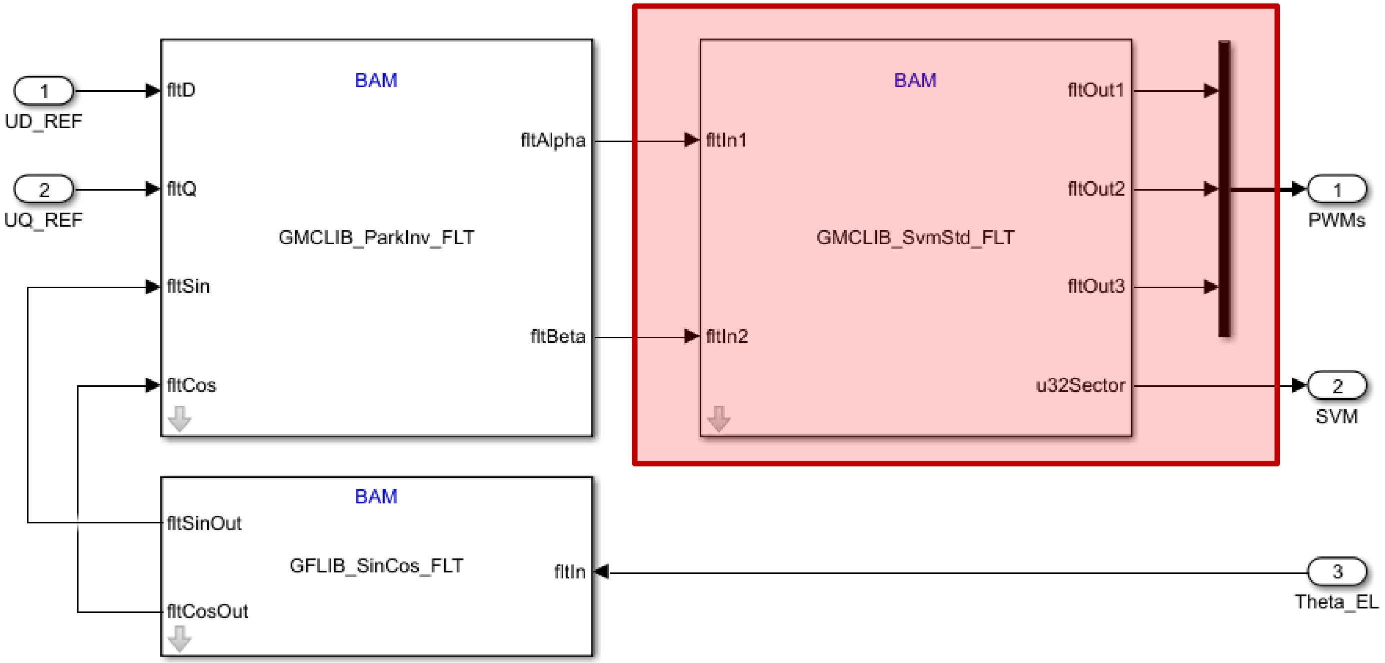
Modified Space Vector Modulation Output Waveforms

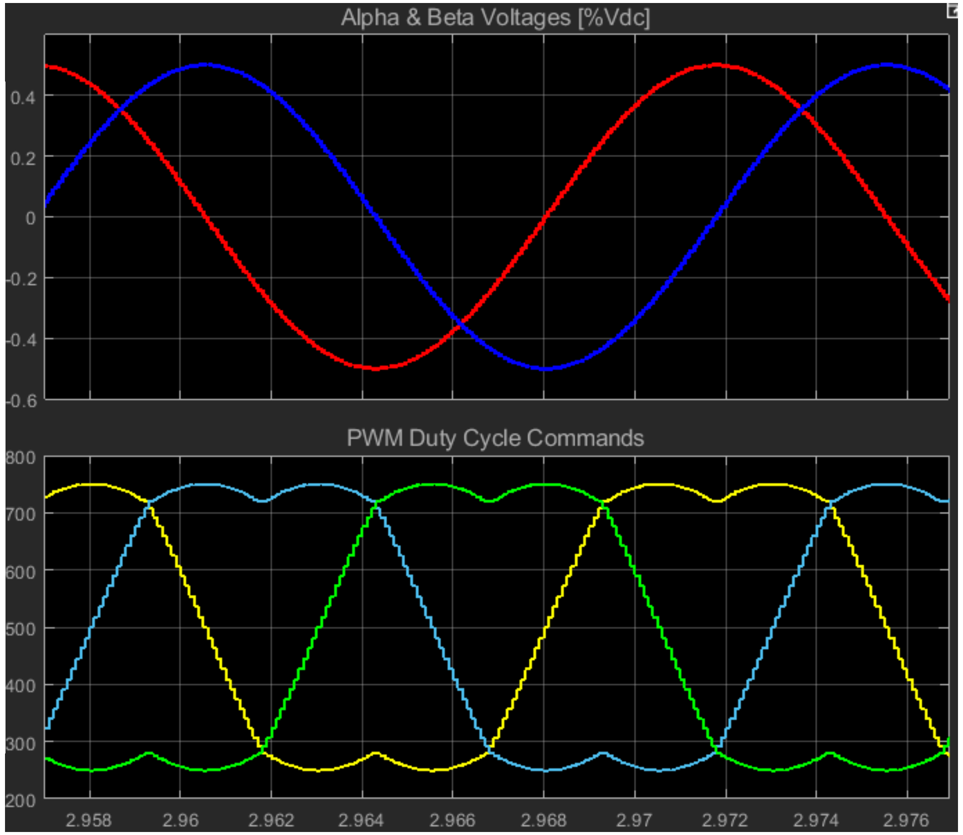
- Output voltage vector is created by switching continuously between the adjacent base vectors and the “NULL” vectors so that the vectorial time-average of the asserted base space vectors is equal to the commanded voltage:
  - Time spent in zero vectors is divided equally between (0,0,0) and (1,1,1);
  - Switching waveforms should be symmetric, utilizing center-aligned PWM;
- Generates maximum phase voltage $0.57V_{DC}$ instead of just $0.5V_{DC}$;
- Both nulls O000 and O111 are generated at each cycle



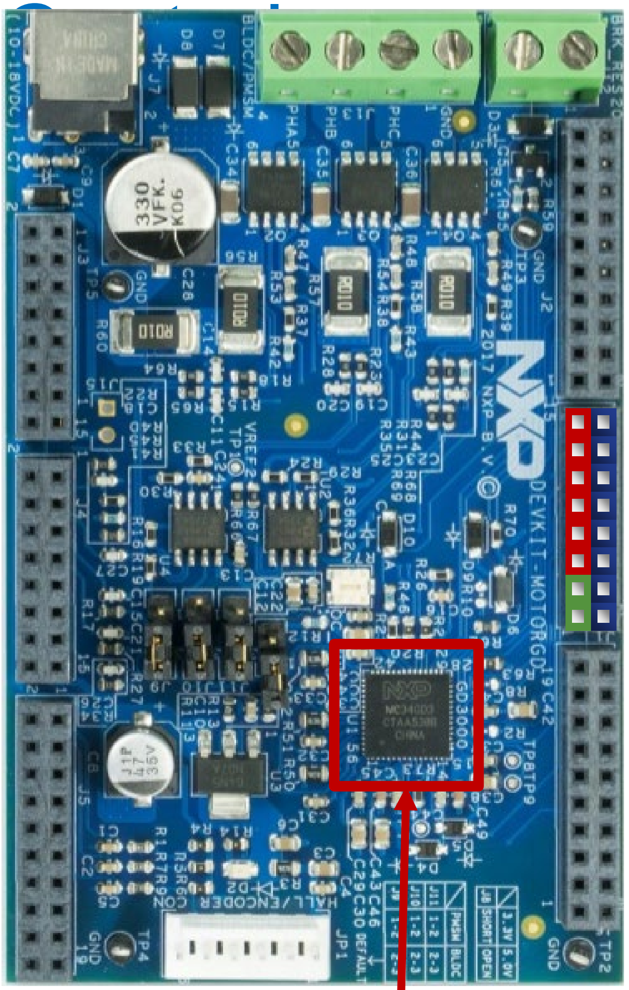
Space Vector Modulation – Implementation

- AMMCLIB provides a specialized block that computes the PWM duty cycles using modified space vector modulation.





MotorGD DevKit – Peripherals Used for Power Stage Control



Pre-driver
MC34GD3000

PORT J1	PIN	Function
PWMC_LS/PS_LS	J1-15	Low switch phase C PWM command
PWMC_HS_B/PC_HS	J1-13	High switch phase C PWM command
PWMB_LS/PB_LS	J1-11	Low switch phase B PWM command
PWMB_HS_B/PB_HS	J1-09	High switch phase B PWM command
PWMA_LS/PA_LS	J1-07	Low switch phase A PWM command
PWMA_HS_B/PA_HS	J1-05	High switch phase A PWM command
GD_RST_B/RST	J1-03	FET pre-driver reset pin
GD_EN/EN1,EN2	J1-01	Enable FET pre-driver gate drive output

MC34GD3000 FET Pre-driver Configuration Sequence

A valid initialization sequence is:

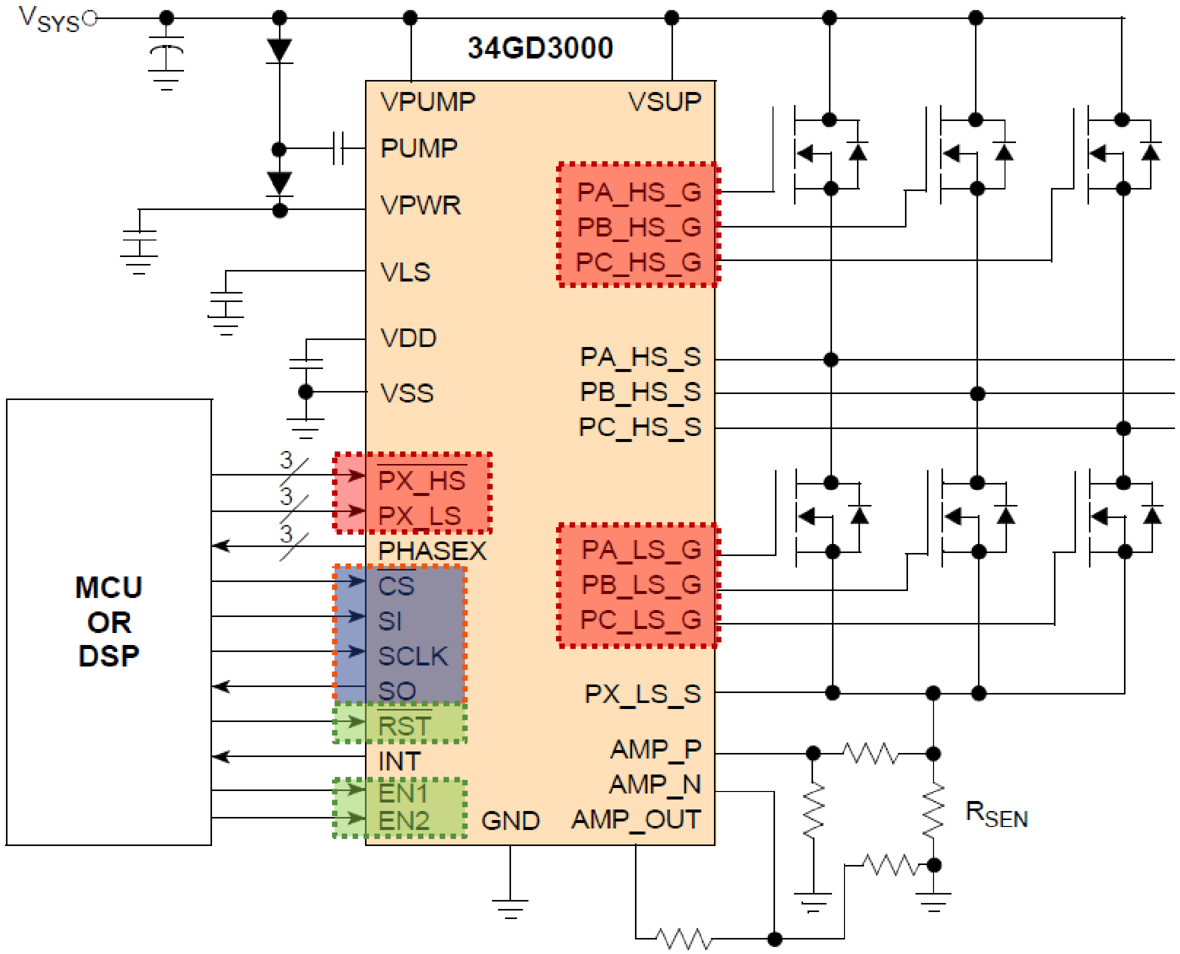
1. RST goes high (EN1 and EN2 remain low)

2. 5. EN1 and EN2 are set high

6. PA_LS_G, PB_LS_G, and PC_LS_G are toggled high for about 1.0 µs (HS outputs are enabled, but not latched)

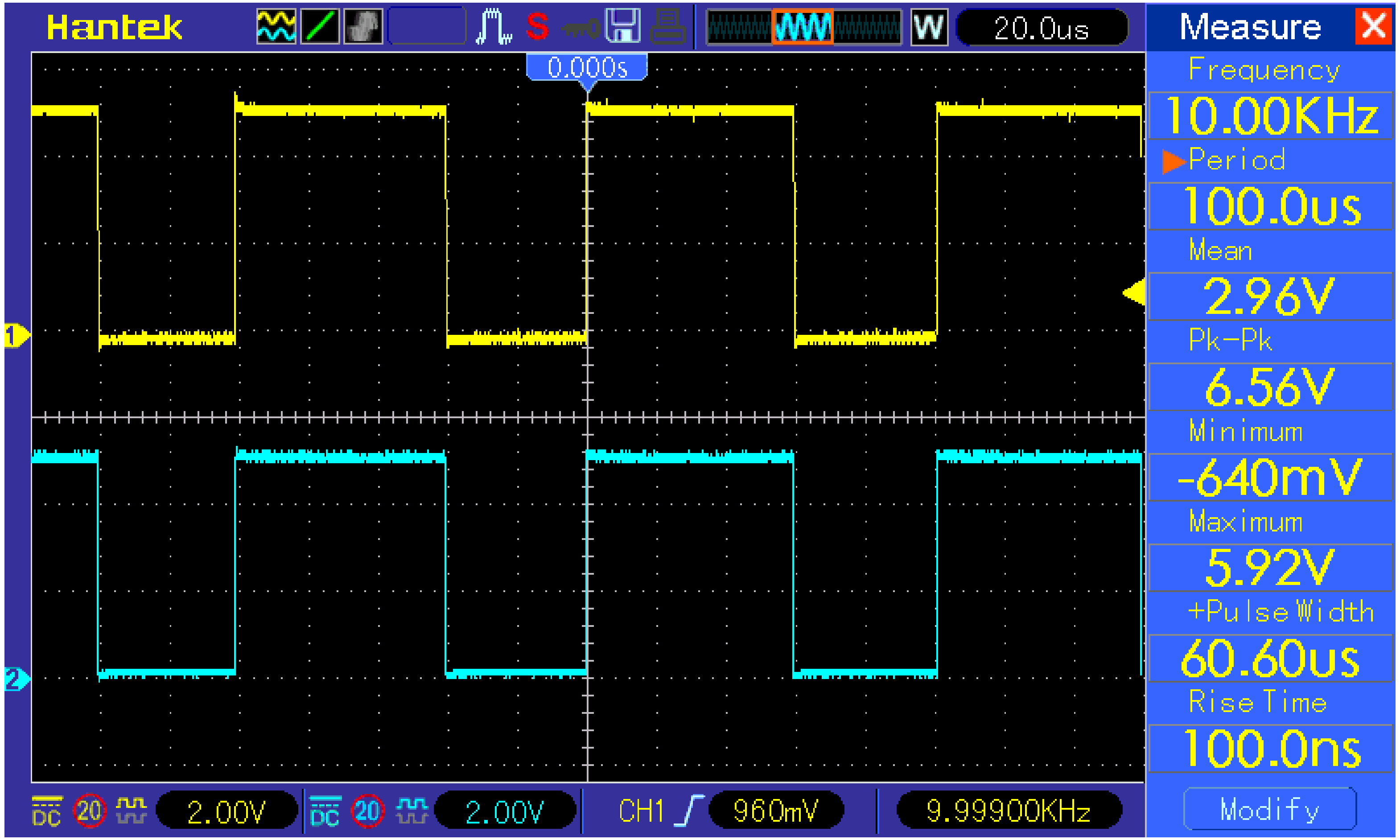
7. Toggle PA_HS_G, PB_HS_G, and PC_HS_G Low for deadtime plus at least 0.1 µs

Note: PX_HS signal are inverted – active low



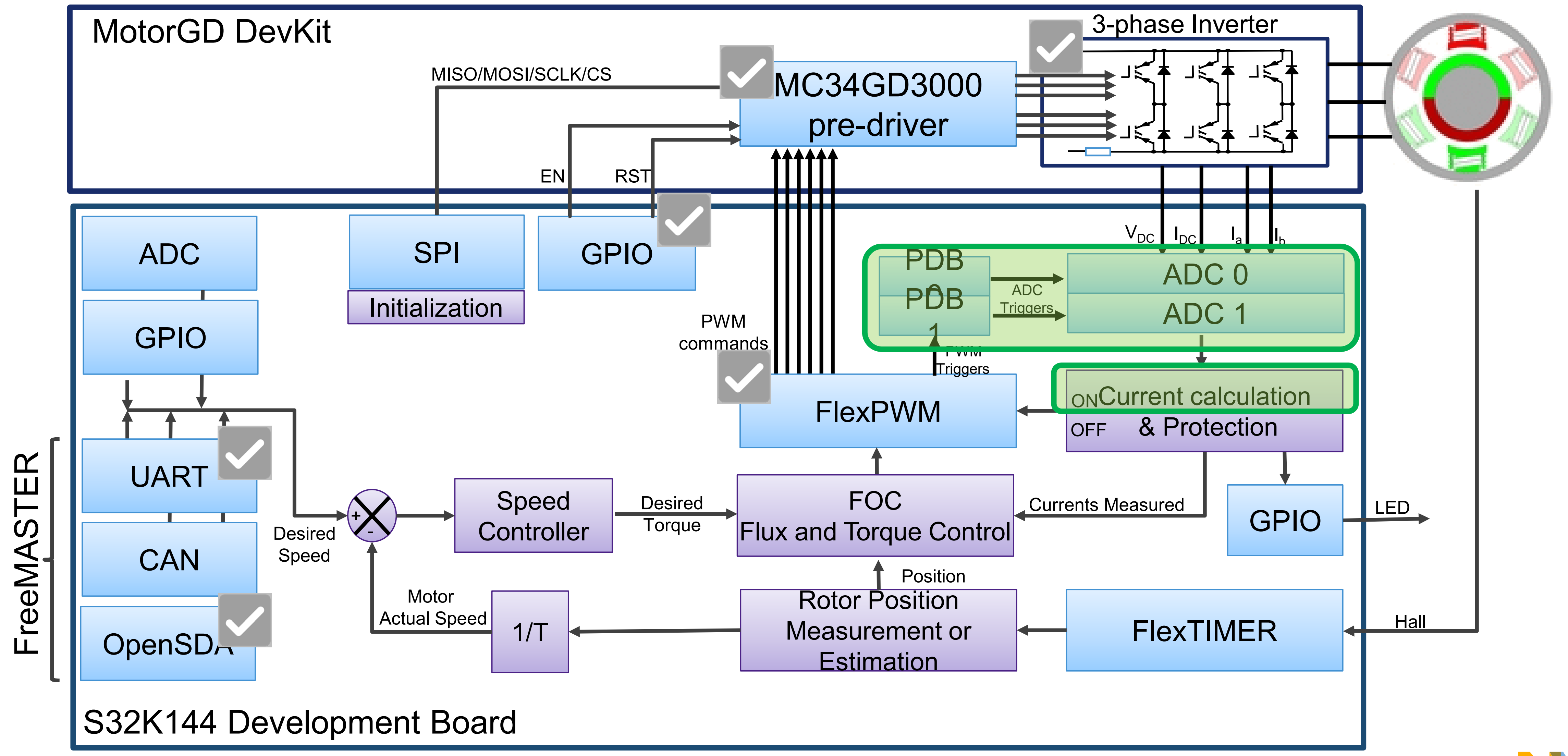
Source: MC34GD3000 datasheet Rev. 3.0, 5/2016

PWM Generation: Top MOSFET vs. Bottom MOSFET Gate Commands



Frequency
10.00KHz
▶Period
100.0us
Mean
2.96V
Pk-Pk
6.56V
Minimum
-640mV
Maximum
5.92V
+Pulse Width
60.60us
Rise Time
100.0ns

Application Mapping: SW vs. HW



Current Sensing Methods

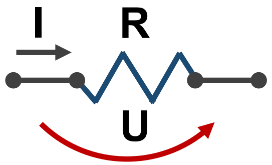
- Motor drive systems need to measure current accurately for:
  - Control Feedback;
  - Protection for overcurrent, short-circuit, i2t, etc;
  - Information about power consumption levels;
- There are 4 major methods to measure the current with the advantages and disadvantages of each

Current Sense Method	Advantages	Disadvantages
Shunt resistor	Low cost, accuracy and wideband	High power dissipation
Conductor	No-cost, PCB tracks, reduce BOM	Poor accuracy due temperature variations
Hall effect	Galvanic isolation, low power dissipation	High cost, limited bandwidth
Transformer	Isolation, high current sensing capabilities	Large size & mass, AC only

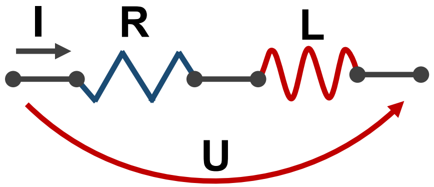
- In general due to low cost and good accuracy, the **shunt resistors** are frequently the best answer for most of the applications

Shunt Resistor Method

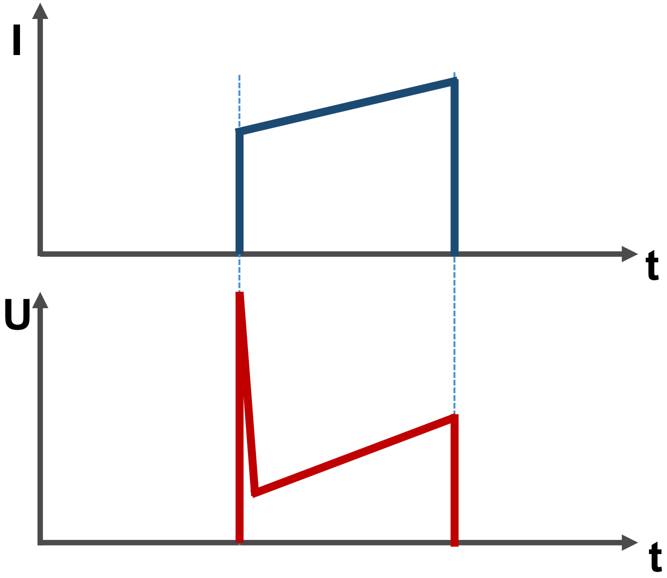
- Current is proportional with the voltage drop across the resistor: I = V/R

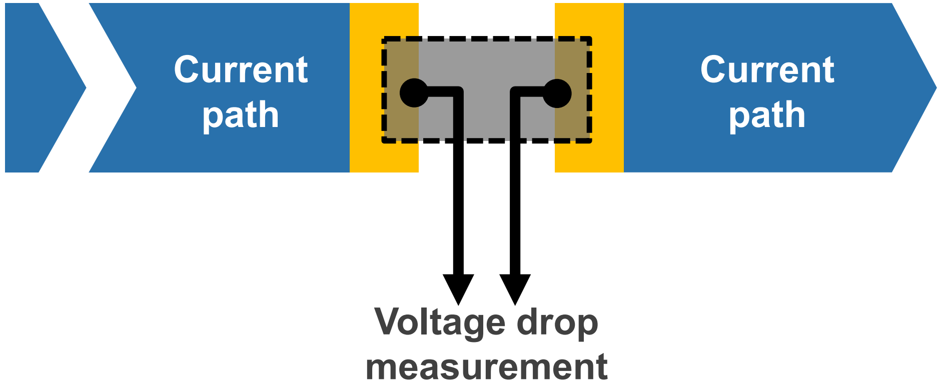


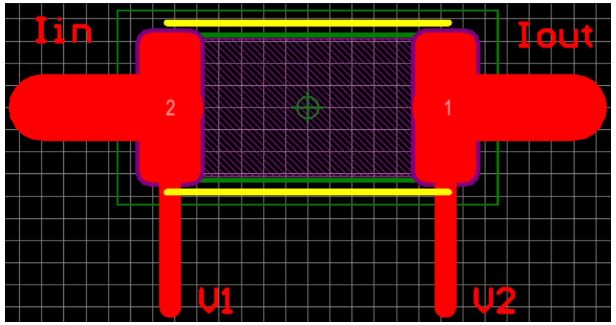
- Parasitic inductance due to inductance components



- To reduce/eliminate the parasitic inductances special resistors and PCB designs are used







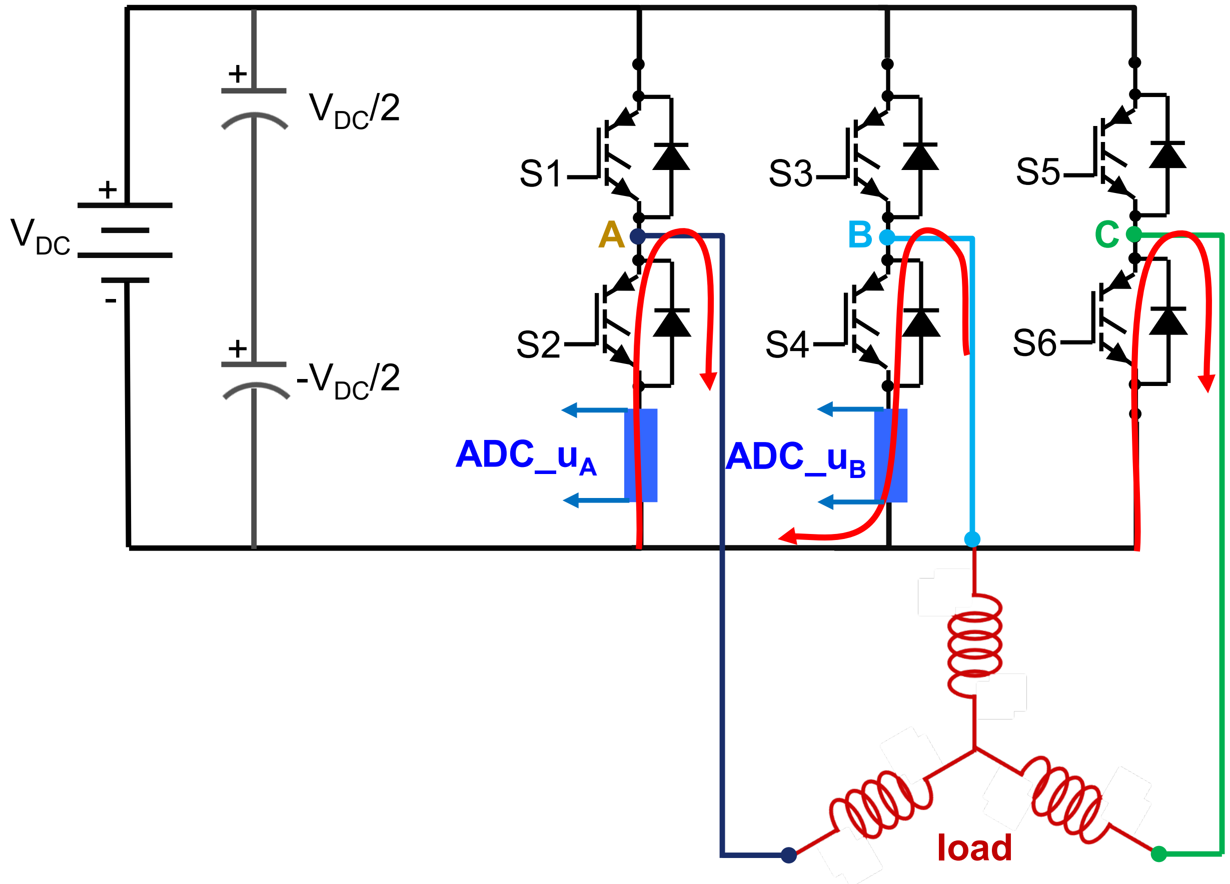
Motor Control: Dual Shunt Current Sensing

- For isolated 3 phase symmetric systems only two phase currents are needed to be sampled by the ADC, the third current being determined from Kirchhoff circuit law

- Measurements can be done only when the current flows thru lower switch

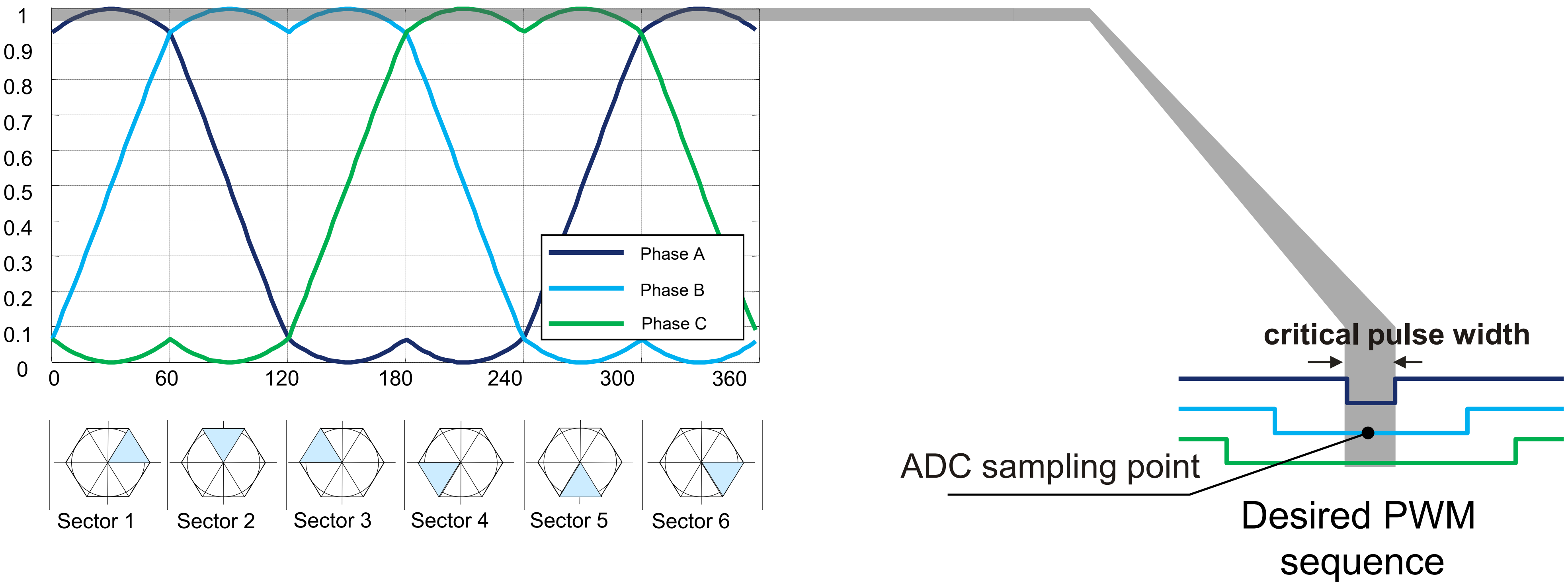
- ADC dual-sampling is required to read currents in the same moment

- As general rule: a positive current is defined as the current that flows into the motor winding while the negative current is defined a the one that flows out of the motor winding.



Motor Control: Current Sensing Considerations

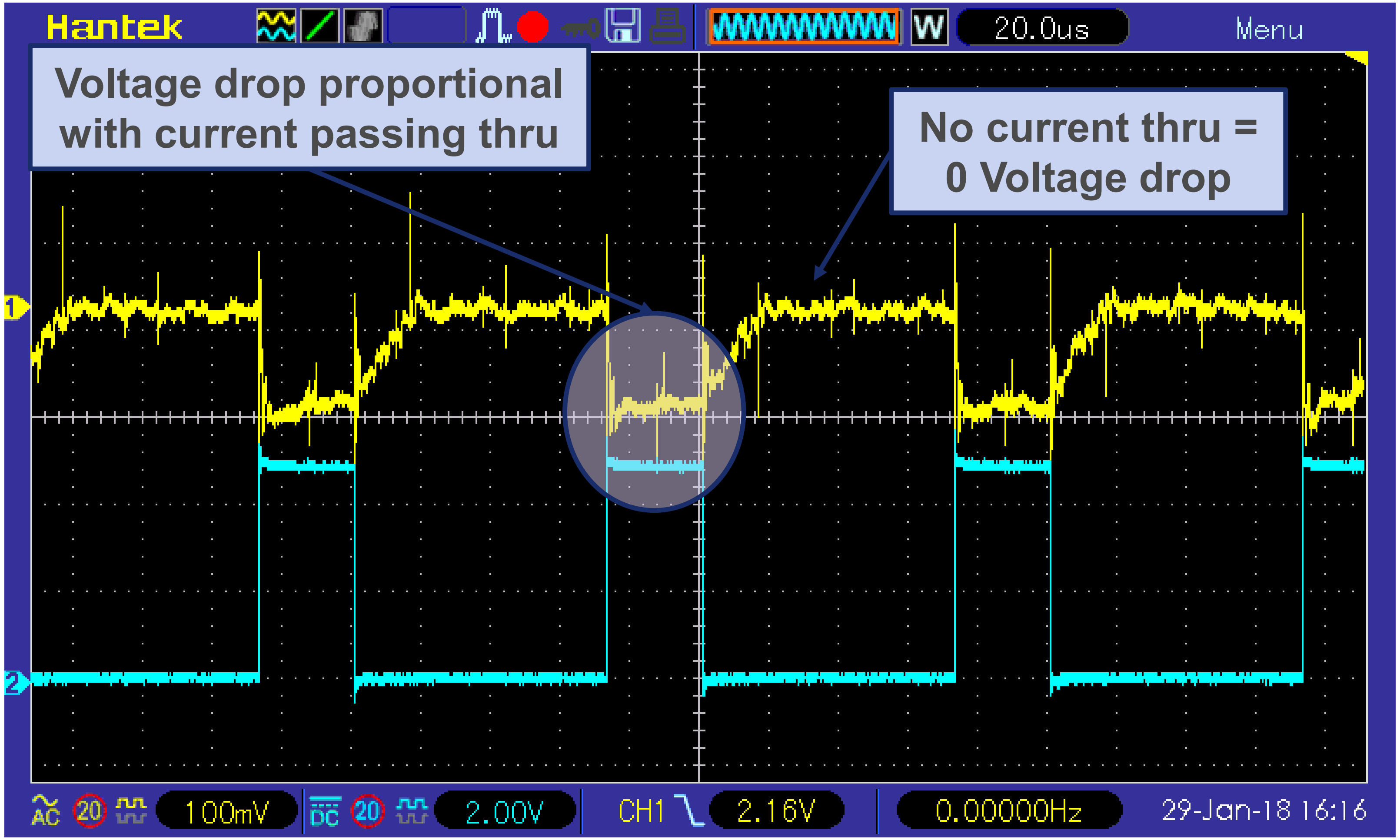
- Bottom transistor must be switched on at least for a critical pulse width to get stabilized current shunt resistor voltage drop
- At any time, this rule needs to be accomplished for the legs where the shunts are located.
- Minimum pulse width defined by system delays and ADC sampling time



Voltage Drop Across the Shunt Resistor

Shunt Voltage drop due to current passing thru

10kHz PWM command for lower MOSFET

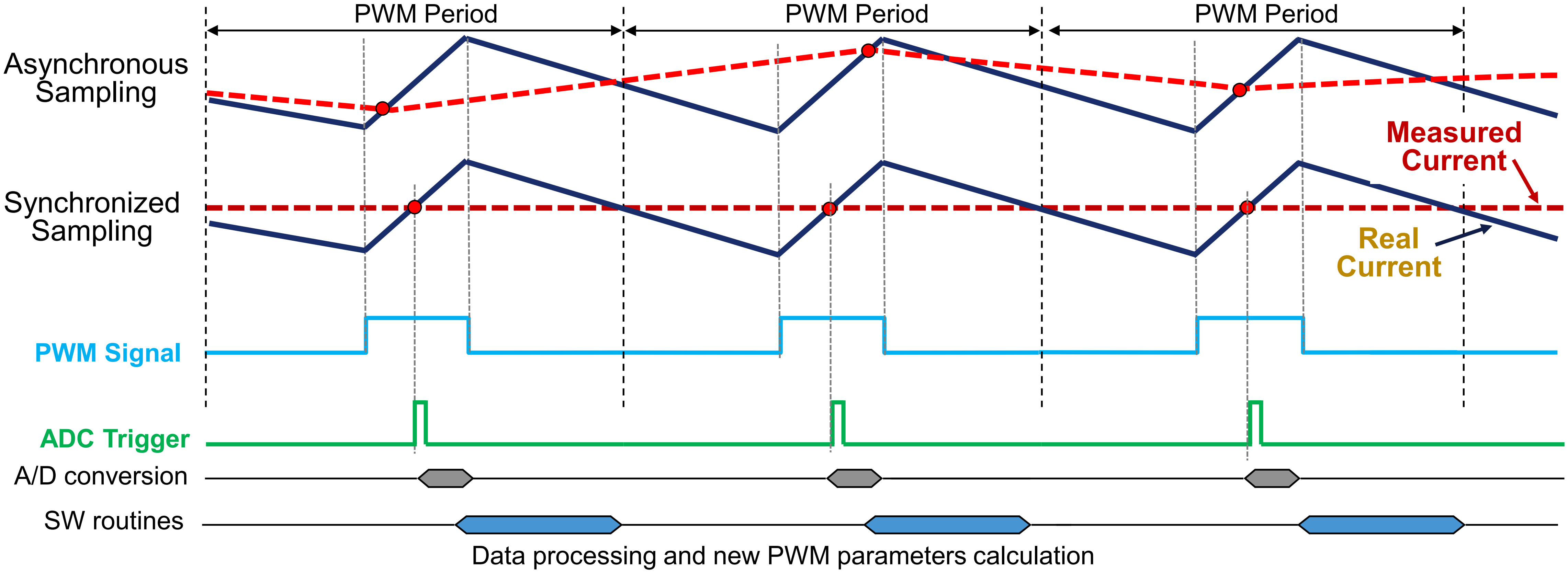


Voltage drop proportional with current passing thru

No current thru = 0 Voltage drop

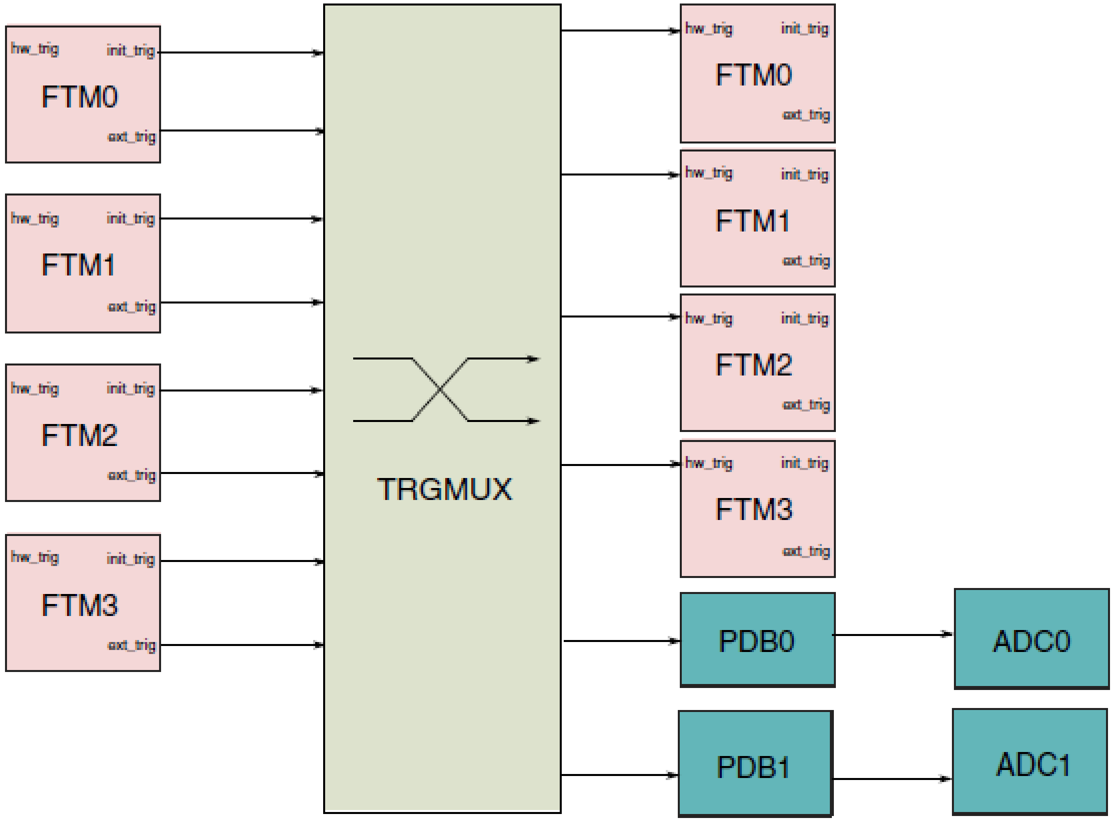
PWM – ADC Synchronization

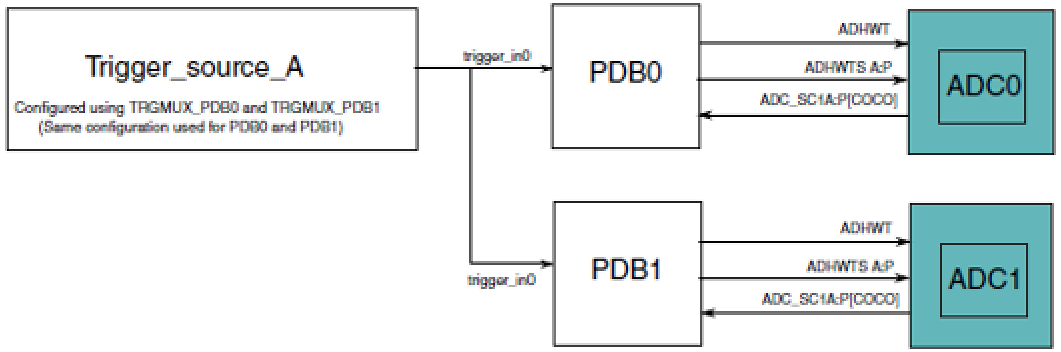
- Enforce ADC sampling at regular time intervals & helps to filter the measured current - antialiasing



PWM – ADC Synchronization (Cont'd)

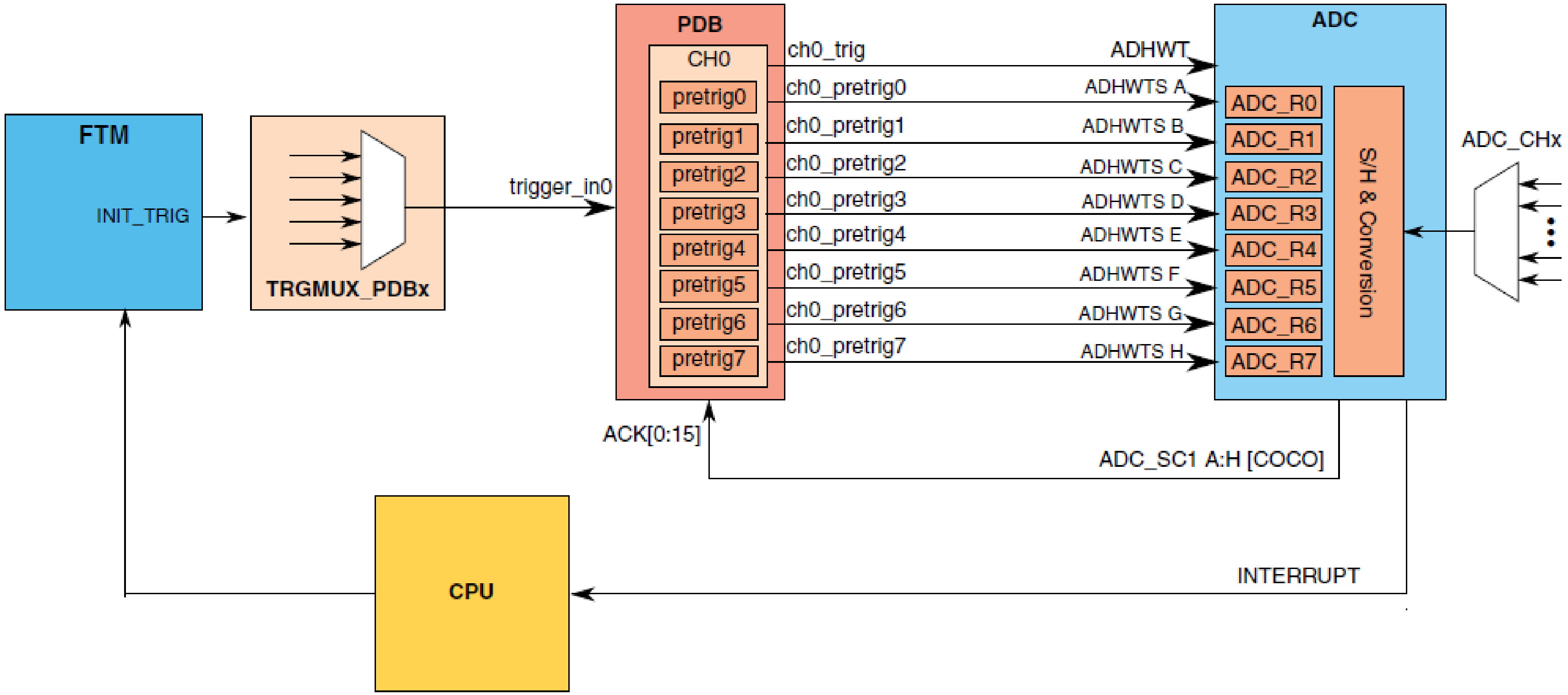
- On S32K MCU the synchronization can be done via Programable Delay Block (PDB).
- The FlexTimer trigger outputs are used as trigger source for PDB that in turns triggers the ADC
- Same FTM trigger is used to start both ADC instances





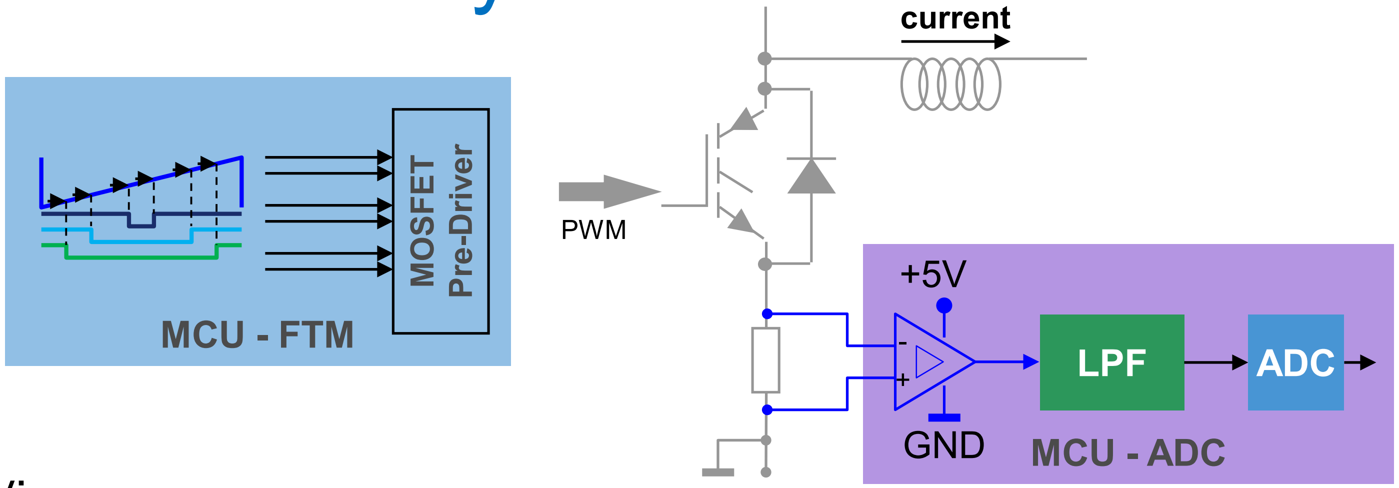
ADC Trigger Concept

- The FTM supports the counter **INIT_TRIG** which will be used as a trigger input of the PDB. The PDB then triggers the ADC.
- Each ADC channel in the PDB module supports up to **8 pre-triggers** used as the ADC hardware channel selection. After a pre-trigger, the ADC trigger initiates the ADC conversion.



Delays Involved in PWM – ADC Synchronization

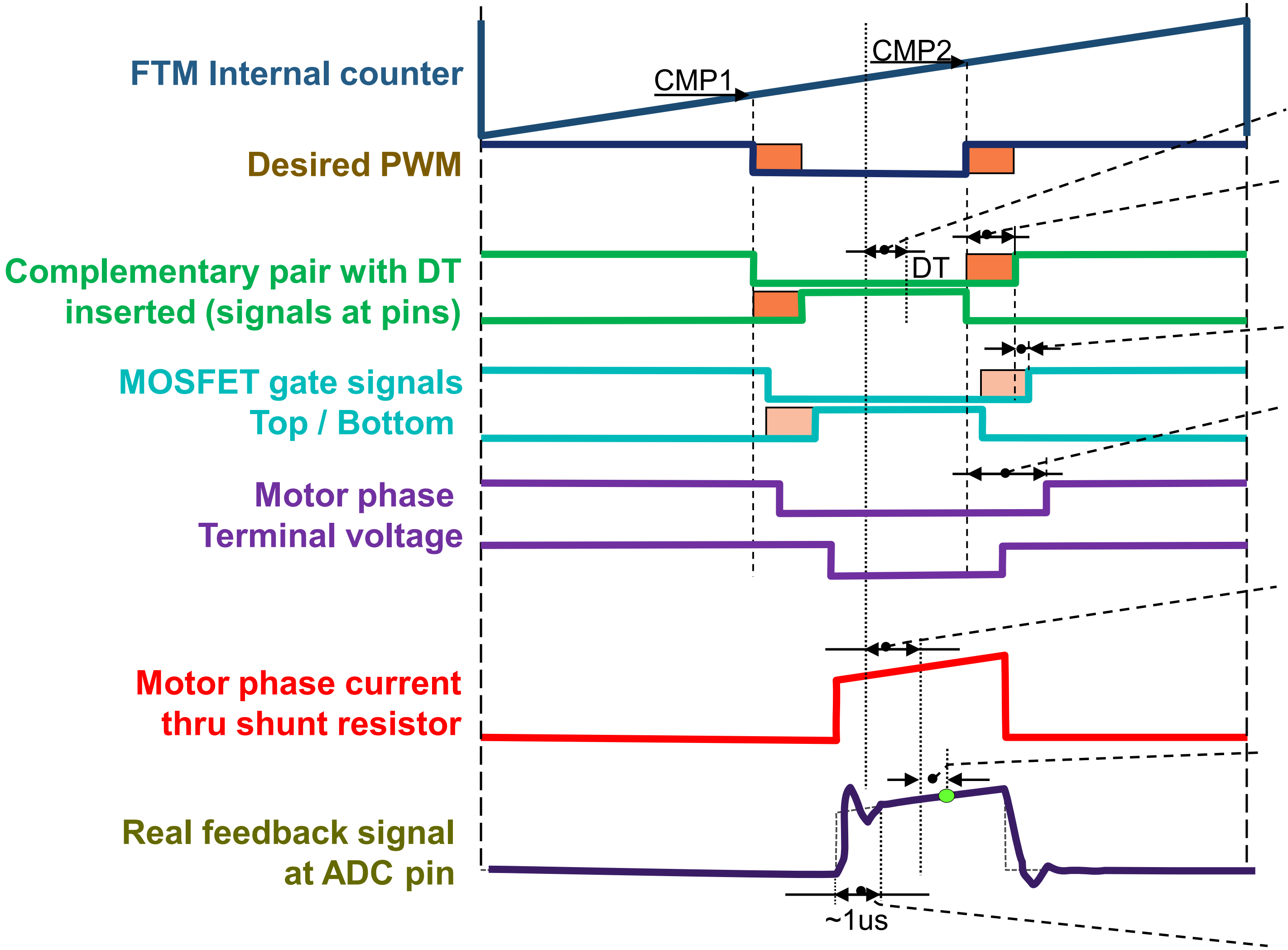
- Current sensing processing chain



- Delays are chained and are caused by:

Delay Type	Delay impact	Time
PWM Dead time insertion		100ns – 1/2us
Opto-coupler propagation delay		40ns – 300ns
MOSFET pre-driver propagation delay		~50ns
MOSFET turn ON/OFF times		100ns – 1us
Amplifier slew rate		<1us
Low-pass filter (LPF) delay		1-2us
ADC delays		50ns-200ns

Delays Involved in PWM – ADC Synchronization (Cont’d)


FTM Internal counter
CMP1
CMP2
Desired PWM
DT
Complementary pair with DT inserted (signals at pins)
MOSFET gate signals Top / Bottom
Motor phase Terminal voltage
Motor phase current thru shunt resistor
Real feedback signal at ADC pin
~1us

Mid point shifts by DT

Rising edge is shifted by DT

Delay1 = T_{opto}+T_{preDRV}= ~300 ns + ~50 ns

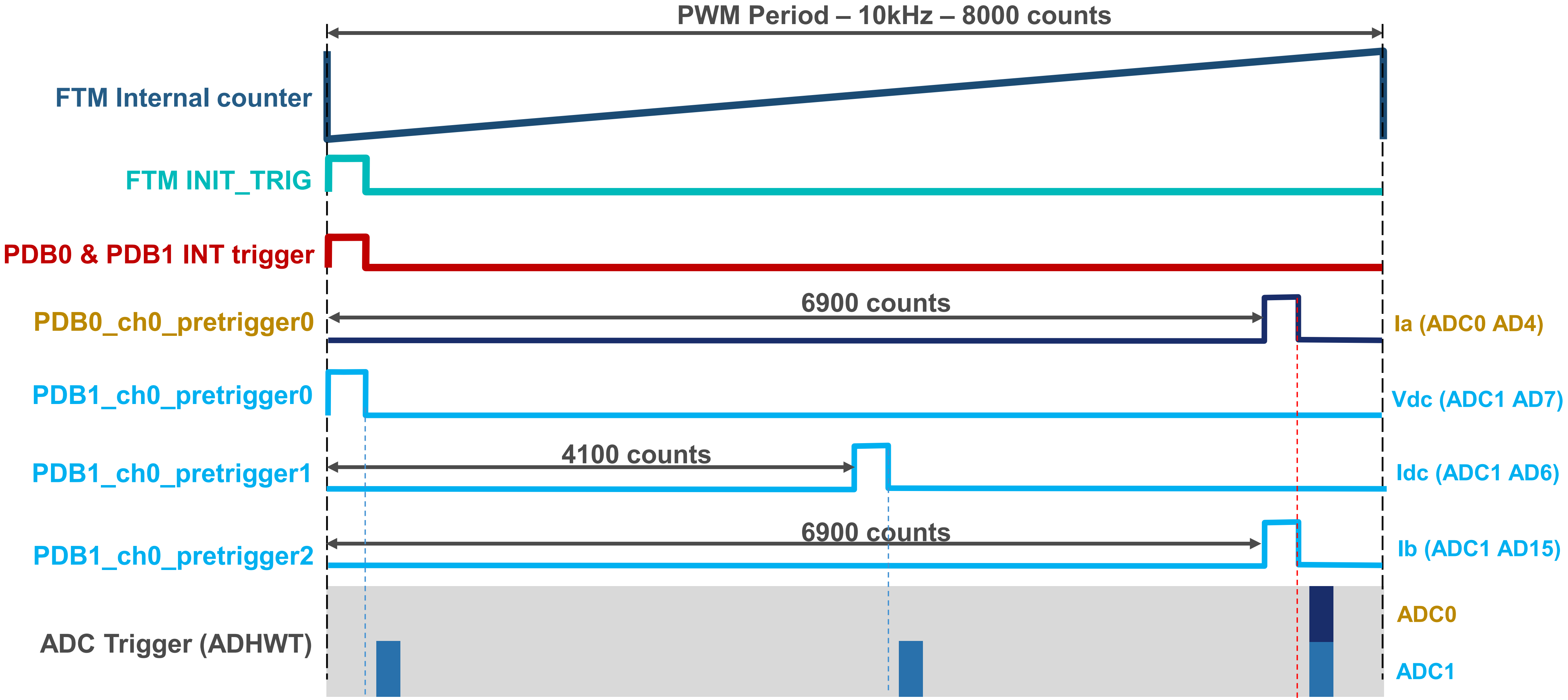
Delay2 = T_{opto}+T_{preDRV}+T_{on}/2+DT
Delay2 = ~350ns +(0.1÷1 us)/2+(0.3÷4 us)
Delay2 = ~0.5 ÷ 5 us

Mid point shifts
= T_{opto}+T_{preDRV}+(T_{on}+T_{off})/4 +(DT1 or DT2)/2 =
= ~300 ns + ~50 ns + (0.2÷2 us)/4 + (0.3÷4 us)/2 =
= ~0.4 ÷ 5 us

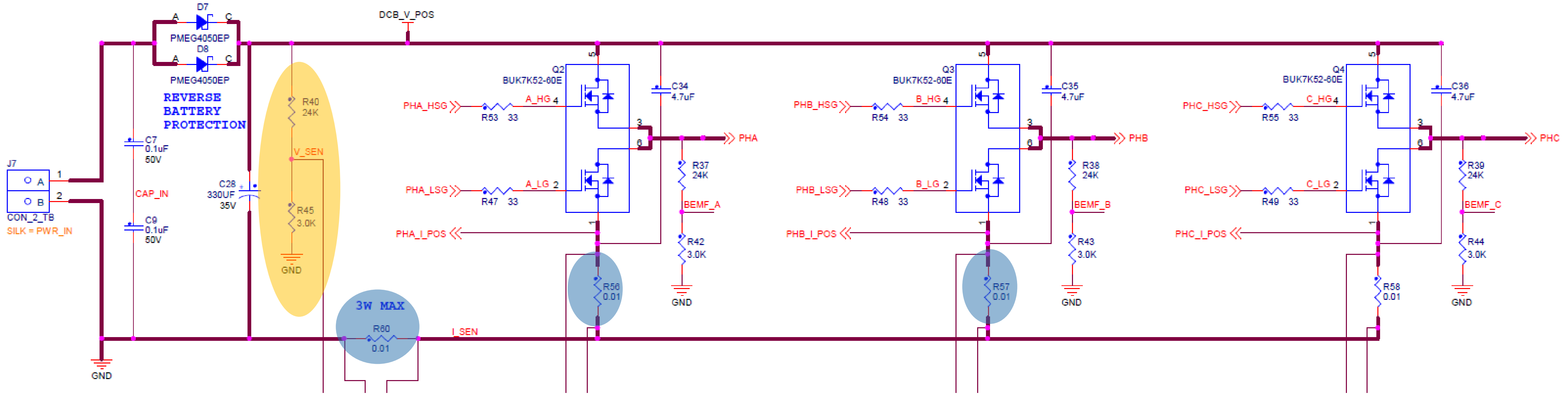
Low pass filter delay + T_{opto}: ~1us

Amplifier slew rate: <1 us/full range

PWM-PDB-ADC Synchronization



MotorGD DevKit: Inverter Schematic



Voltage divider
Measure **Vdc**

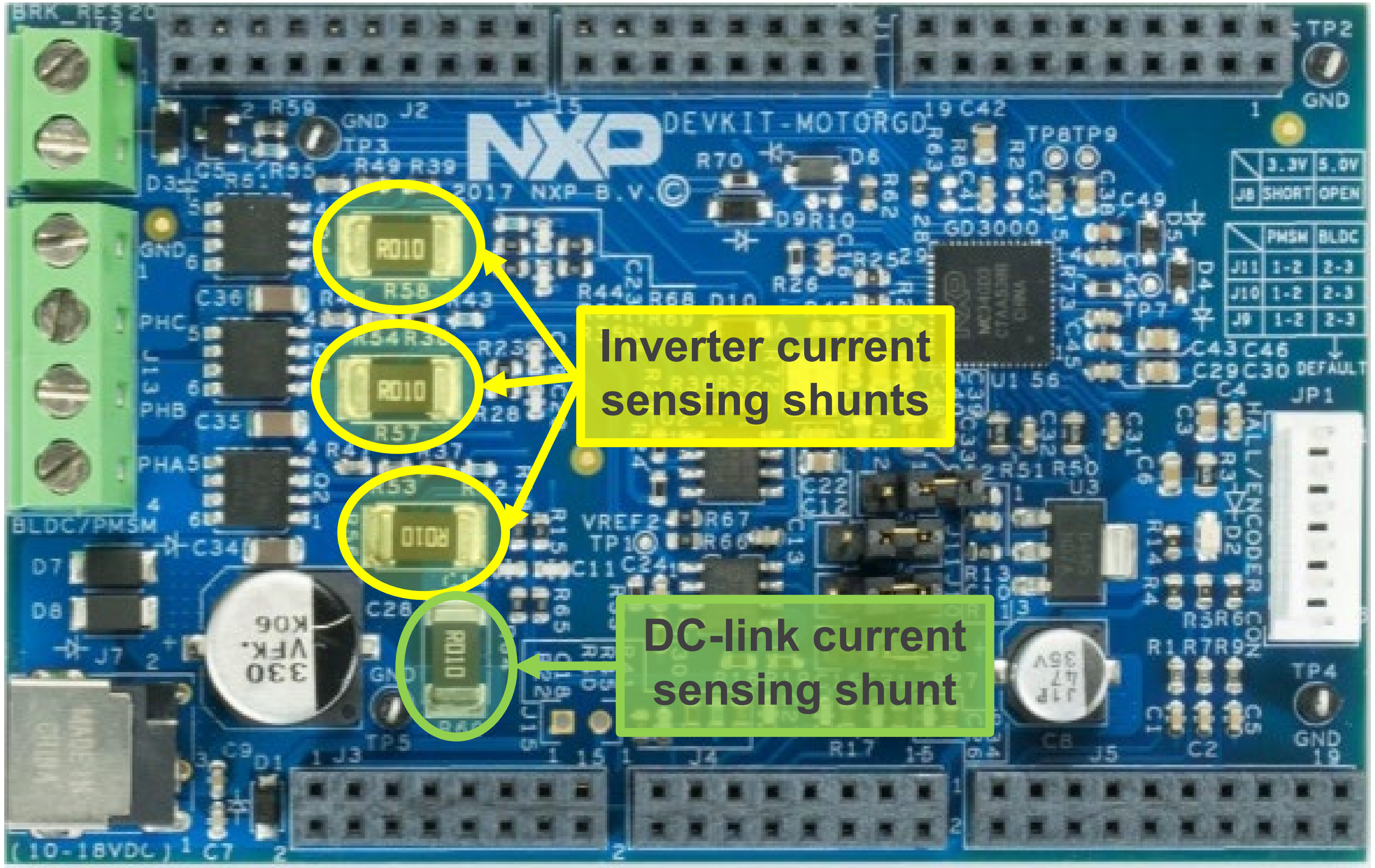
Shunt Resistor
Measure **Idc**

Shunt Resistor
Measure **Ia**

Shunt Resistor
Measure **Ib**

Not used

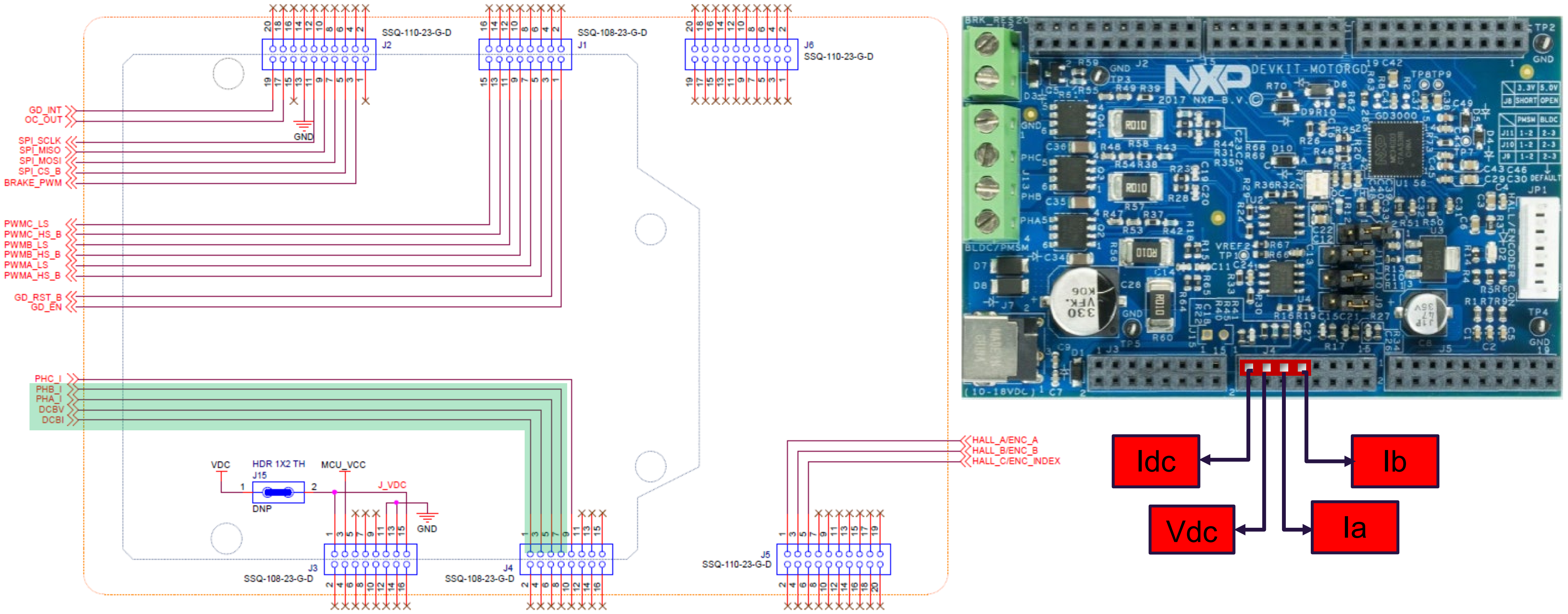
MotorGD DevKit: Inverter Schematic (Cont'd)



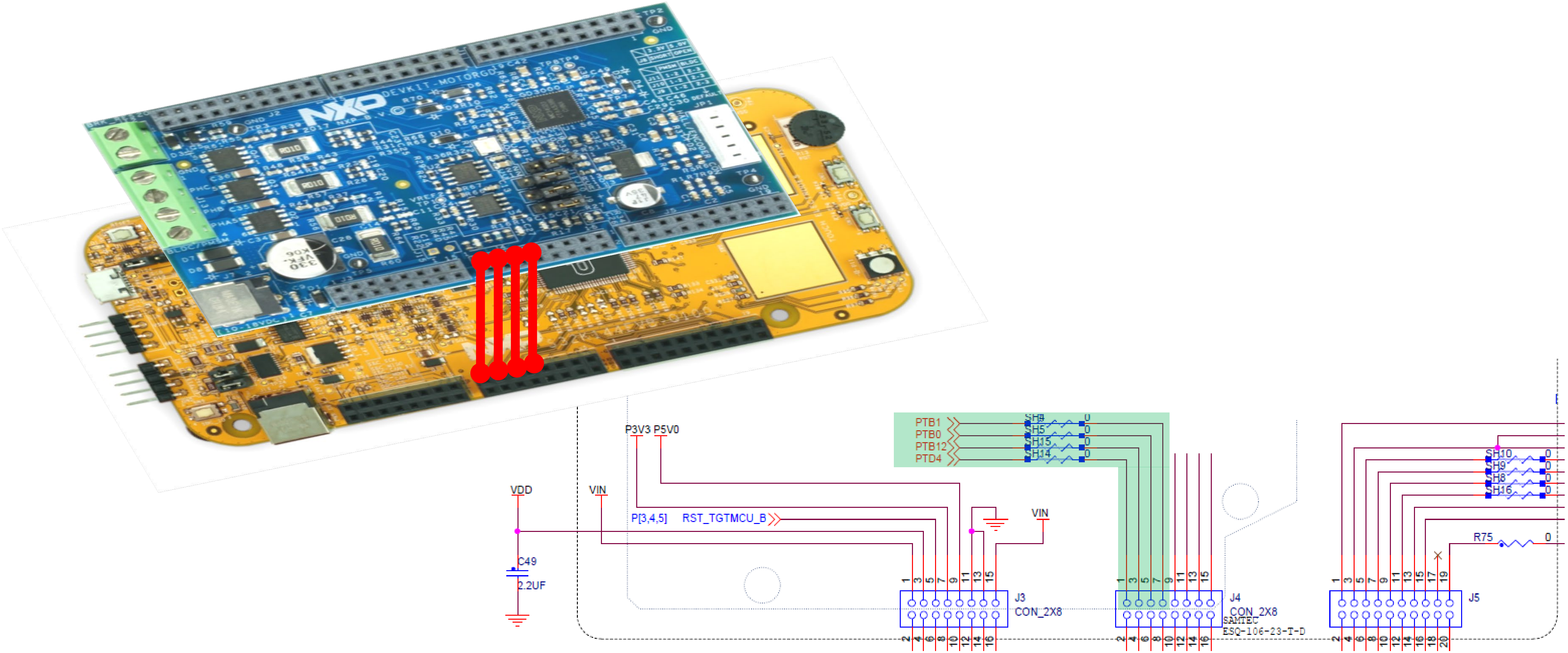
Inverter current sensing shunts

DC-link current sensing shunt

MotorGD DevKit: Pins Used for ADC Measurements



S32K144 MCU Signal Assignment



S32K144 MCU Signal Assignment (Cont’d)

- ADC default pin assignment for sensing currents: **can’t synchronize the phase current sensing**

MotorGD DevKit	MCU pin	ADC instance
Phase A current	PTB0	ADC0/SE4
Phase B current	PTB1	ADC0/SE5
DC bus voltage	PTB12	ADC1/SE7
DC bus current	PTD4	ADC1/SE6

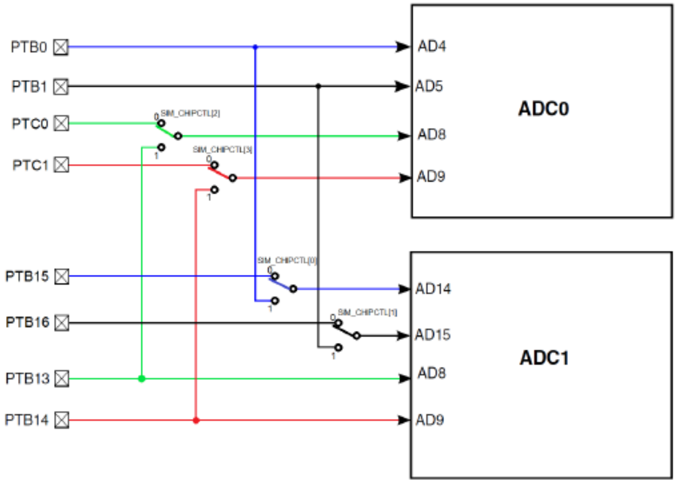
ADC HW Interleave
ADC0/SE4
ADC1/SE16
ADC1/SE7
ADC1/SE6

- Solution is **ADC Hardware Interleaved Channels**

In the hardware interleaved mode, a signal on the pin PTB1 can be sampled

by both ADC0 and ADC1 in the same time. The interleaved mode is enabled

by SIM_CHIPCTL[ADC_INTERLEAVE_EN] bits.



MC34GD3000 FET Pre-driver Configuration Sequence

A valid initialization sequence is:

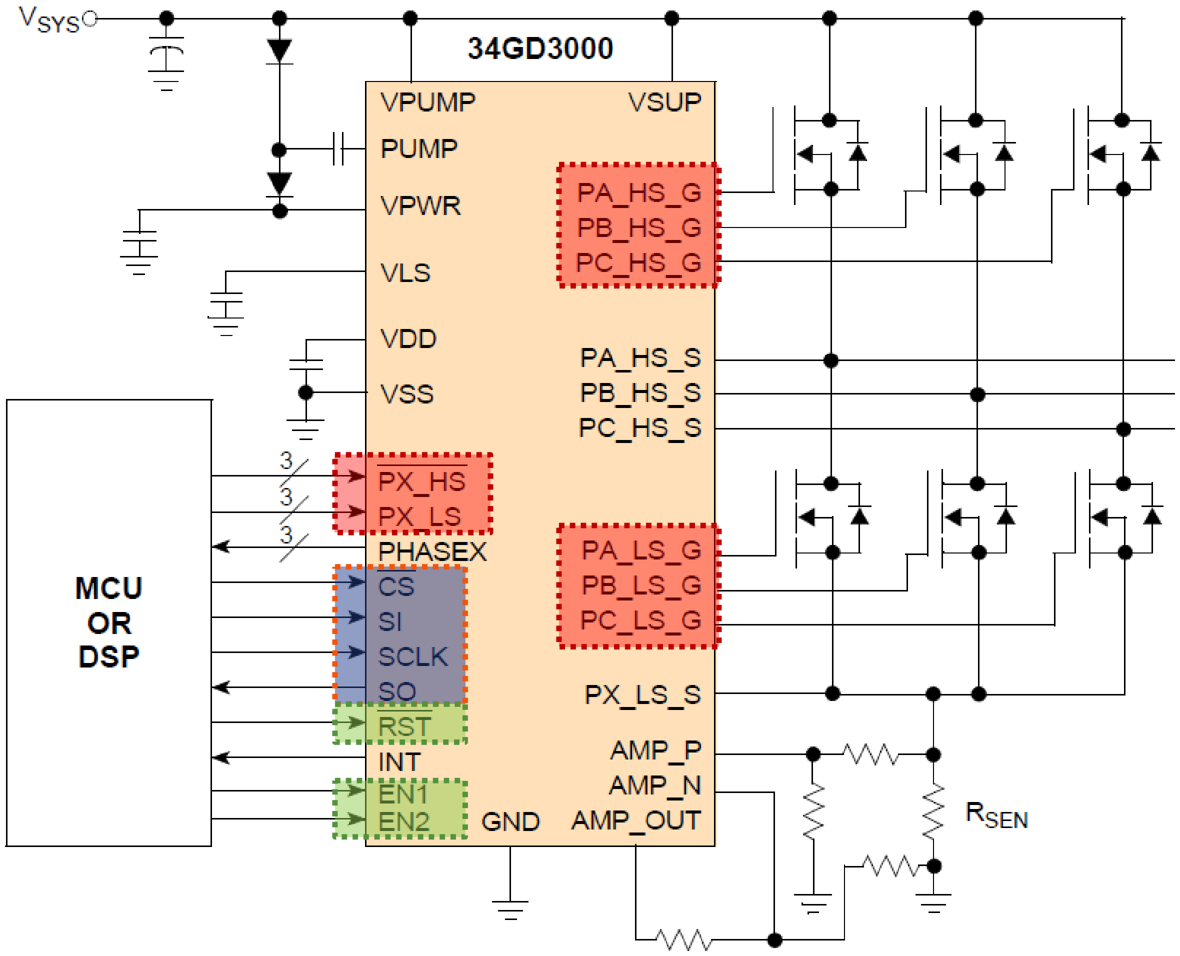
1. RST goes high (EN1 and EN2 remain low)

2. EN1 and EN2 are set high

3. PA_LS_G, PB_LS_G, and PC_LS_G are toggled high for about 1.0 μs (HS outputs are enabled, but not latched)

4. Toggle PA_HS_G, PB_HS_G, and PC_HS_G Low for deadtime plus at least 0.1 μs

Note: PX_HS signal are inverted – active low

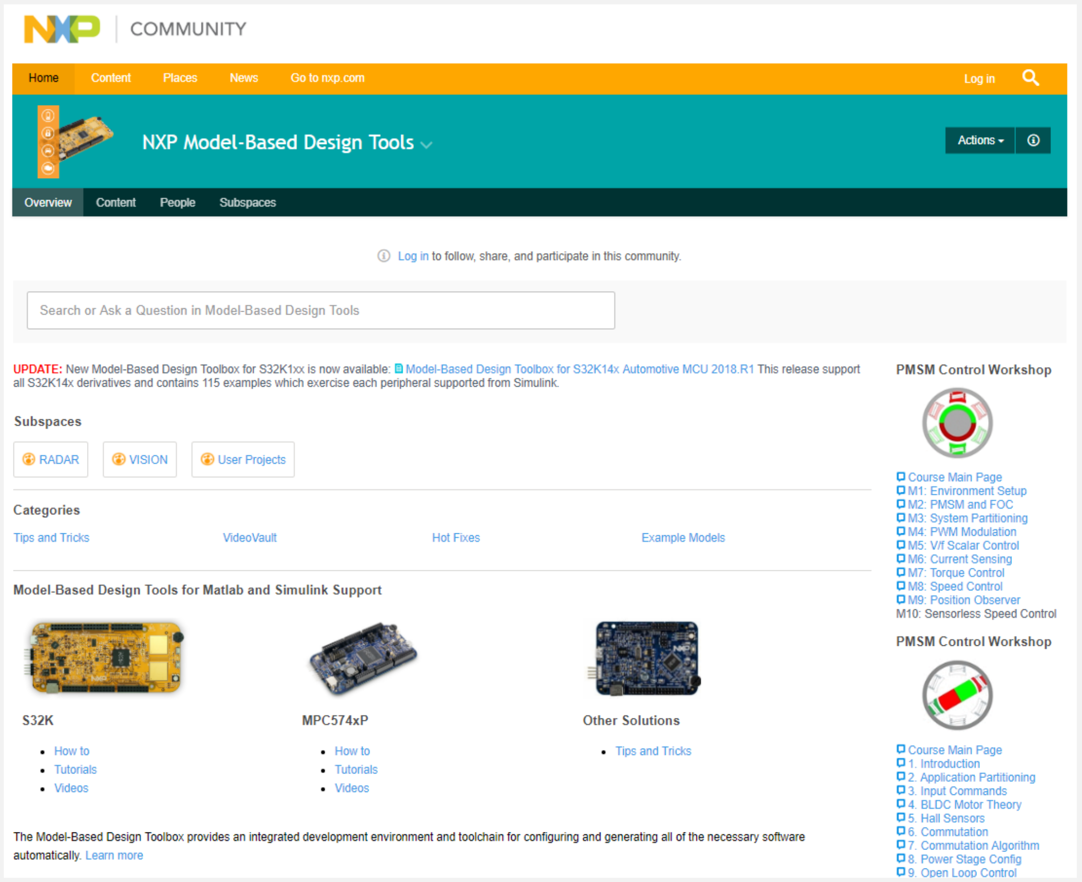


Source: MC34GD3000 datasheet Rev. 3.0, 5/2016

Conclusions

Getting Help

MBDT Online Community Examples & Help



COMMUNITY

Home | Content | Places | News | Go to nxp.com | Log in

NXP Model-Based Design Tools

Actions ▾ | ⓘ

Overview | Content | People | Subspaces

ⓘ Log in to follow, share, and participate in this community.

Search or Ask a Question in Model-Based Design Tools

UPDATE: New Model-Based Design Toolbox for S32K1xx is now available: Model-Based Design Toolbox for S32K14x Automotive MCU 2018.R1 This release support all S32K14x derivatives and contains 115 examples which exercise each peripheral supported from Simulink.

Subspaces

RADAR | VISION | User Projects

Categories

Tips and Tricks | VideoVault | Hot Fixes | Example Models

Model-Based Design Tools for Matlab and Simulink Support

S32K

- How to
- Tutorials
- Videos

MPC574xP

- How to
- Tutorials
- Videos

Other Solutions

- Tips and Tricks

The Model-Based Design Toolbox provides an integrated development environment and toolchain for configuring and generating all of the necessary software automatically. Learn more

PMSM Control Workshop

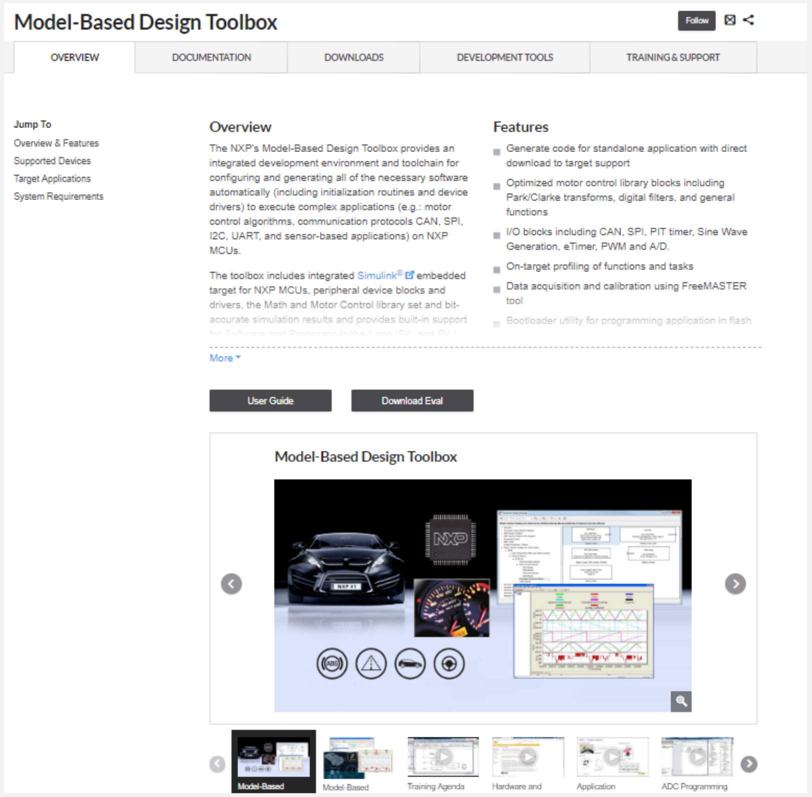
- Course Main Page
- M1: Environment Setup
- M2: PMSM and FOC
- M3: System Partitioning
- M4: PWM Modulation
- M5: V/f Scalar Control
- M6: Current Sensing
- M7: Torque Control
- M8: Speed Control
- M9: Position Observer

M10: Sensorless Speed Control

PMSM Control Workshop

- Course Main Page
- 1. Introduction
- 2. Application Partitioning
- 3. Input Commands
- 4. BLDC Motor Theory
- 5. Hall Sensors
- 6. Commutation
- 7. Commutation Algorithm
- 8. Power Stage Config
- 9. Open Loop Control

MBDT home page
www.nxp.com/mbdt



Model-Based Design Toolbox

Follow

OVERVIEW | DOCUMENTATION | DOWNLOADS | DEVELOPMENT TOOLS | TRAINING & SUPPORT

Jump To

Overview & Features
Supported Devices
Target Applications
System Requirements

Overview

The NXP's Model-Based Design Toolbox provides an integrated development environment and toolchain for configuring and generating all of the necessary software automatically (including initialization routines and device drivers) to execute complex applications (e.g.: motor control algorithms, communication protocols CAN, SPI, I2C, UART, and sensor-based applications) on NXP MCUs.

The toolbox includes integrated Simulink® embedded target for NXP MCUs, peripheral device blocks and drivers, the Math and Motor Control library set and bit-accurate simulation results and provides built-in support for Software and Processor-in-the-Loop (SIL and PIL).

Features

- Generate code for standalone application with direct download to target support
- Optimized motor control library blocks including Park/Clarke transforms, digital filters, and general functions
- I/O blocks including CAN, SPI, PIT timer, Sine Wave Generation, eTimer, PWM and A/D.
- On-target profiling of functions and tasks
- Data acquisition and calibration using FreeMASTER tool
- Bootloader utility for programming application in flash

More ▾

User Guide | Download Eval

Model-Based Design Toolbox

Model-Based | Model-Based | Training Agenda | Hardware and | Application | ADC Programming

SECURE CONNECTIONS
FOR A SMARTER WORLD