Hands-on Workshop: Brushless DC (BLDC) Motor Control Using MagniV S12ZVM MCU

AMF-AUT-T0526

Carlos Vazquez Goyarzu
Automotive MCU Applications Engineer

September 2013

Session Objectives

- By the end of this session, you will be able to:
  - Identify the modules integrated in the S12ZVM for BLDC and PMSM motor drive applications
  - Know the MTRCKTSBNZVM128 motor control kit based on the MagniV S12ZVM microcontroller
  - Create an application using CodeWarrior IDE to control the speed of a BLDC motor based on a potentiometer reading
  - Use FreeMaster for non-intrusive debugging

Agenda

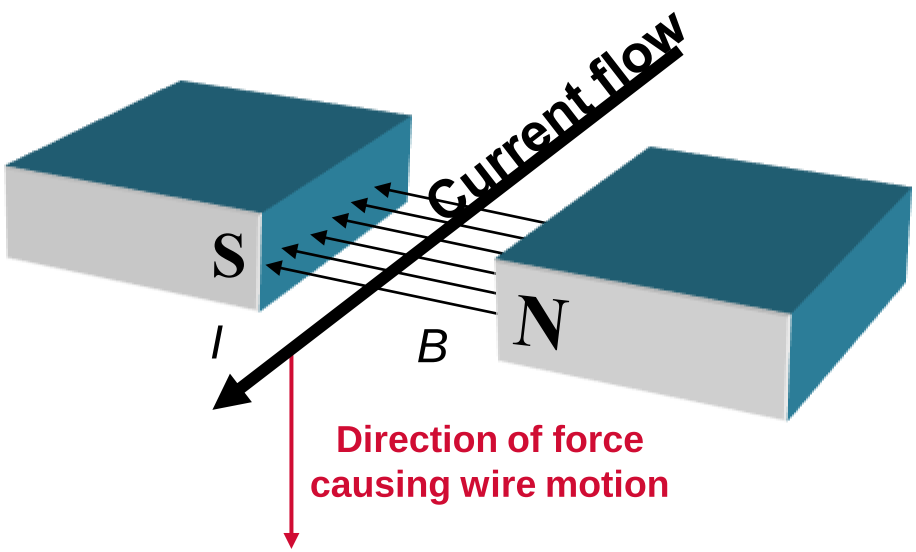
- BLDC motor overview
- S12ZVM microcontroller introduction
- Hardware & Software Development Tools
- Lab #1: Starting a CodeWarrior project for S12ZVM
- Lab #2: The FreeMaster interface and the ADC
- A BLDC motor control application
- Lab #3: Advanced Math and Motor Control Library
- Lab #4: The PMF, PTU, and GDU
- Lab #5: Driving a BLDC motor based on Hall sensors
- Q & A

BLDC Motor Basics

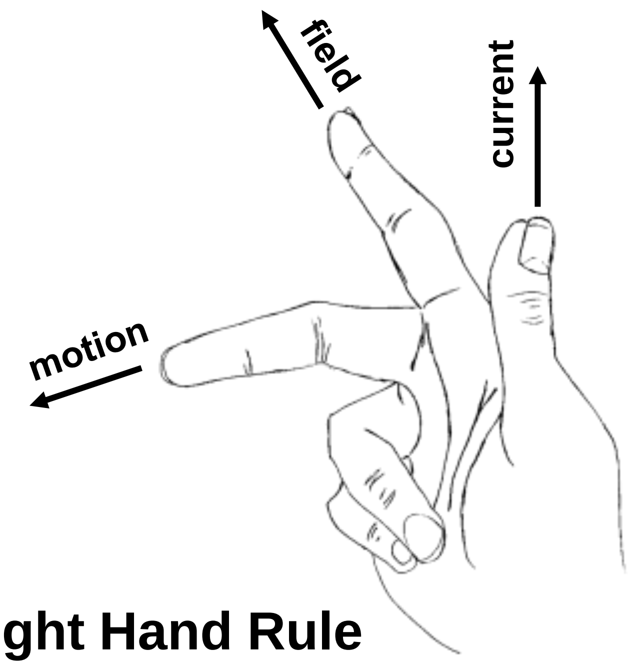
Overview

Lorentz Force

- Current flowing in a magnetic field results in a force on the conductor
- Direction of generated force is governed by “Right Hand Rule” (Lorentz Force Law)

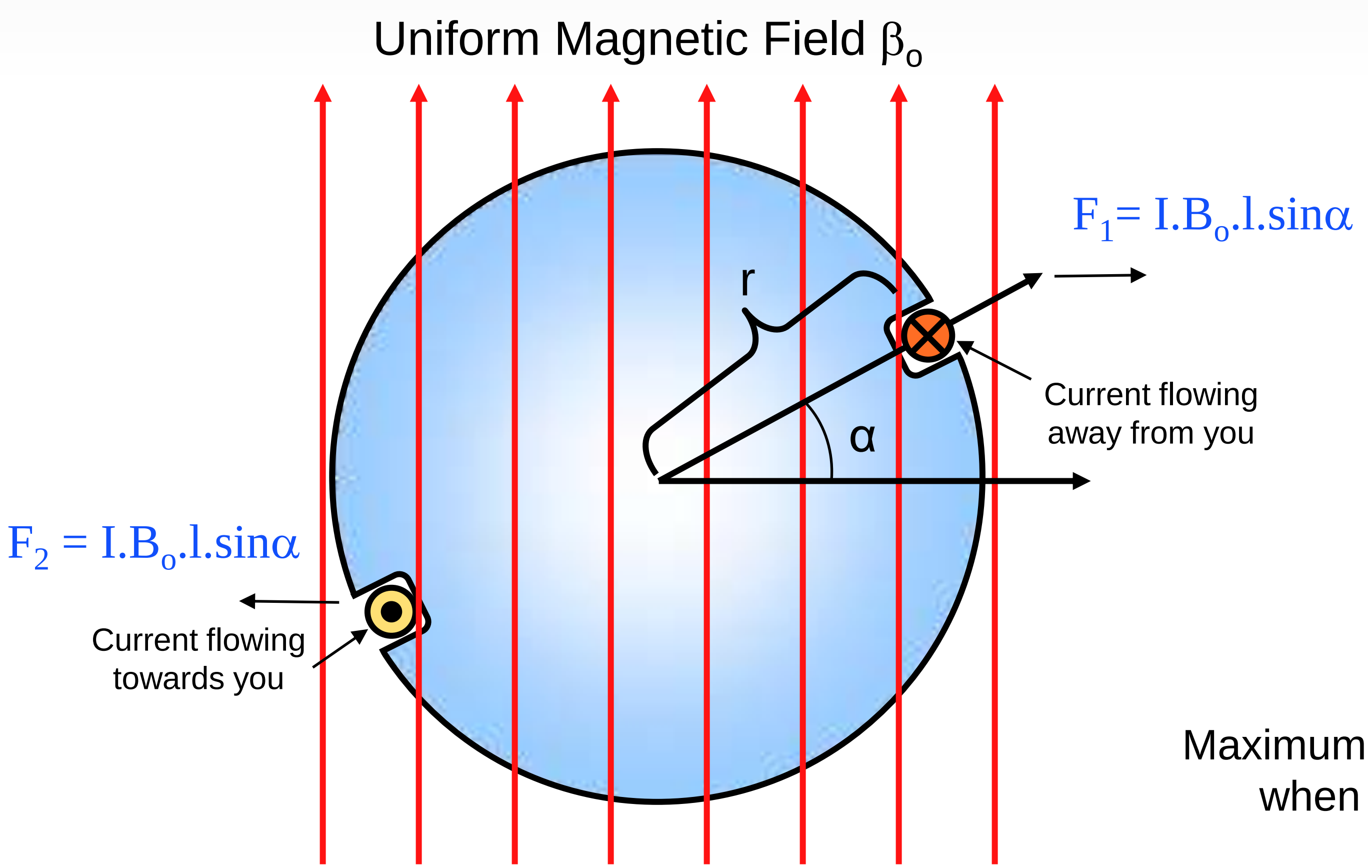


$$F = I.B.l.\sin\theta$$



Right Hand Rule

Electromagnetic Force Creates Torque



Single Coil Rotor

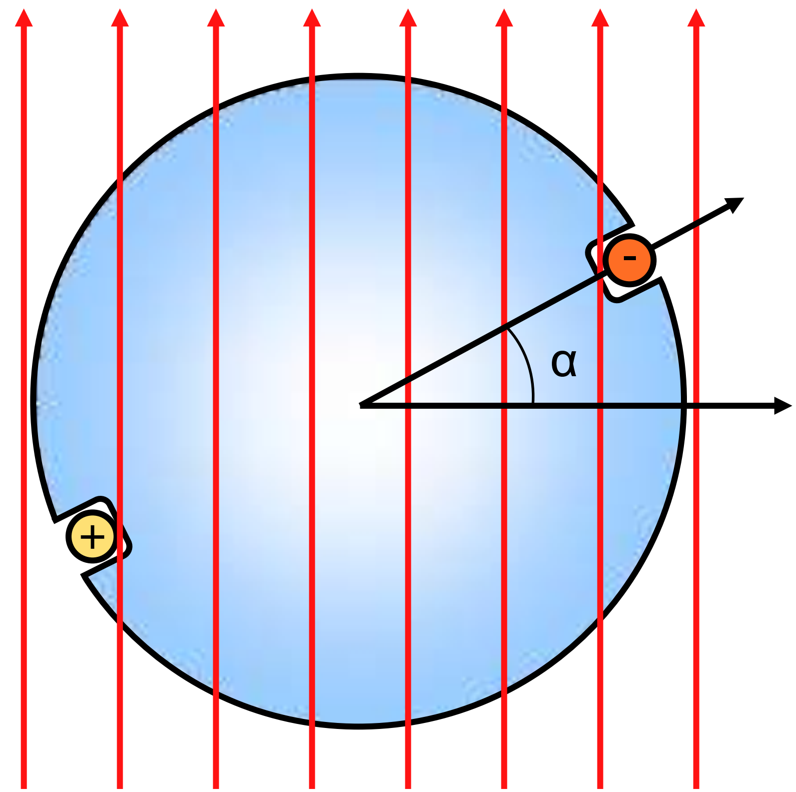
Maximum torque occurs when $\alpha = \pm 90°$

$T = 2I.B_o.l.r \sin\alpha$ (clockwise)

l = length of wire

Electromagnetic Induction

Uniform Magnetic Field B_o



Single Coil Rotor

- A motional emf is induced if a conductor moves in a magnetic field
- The induced emf is proportional to the rate of change of magnetic flux

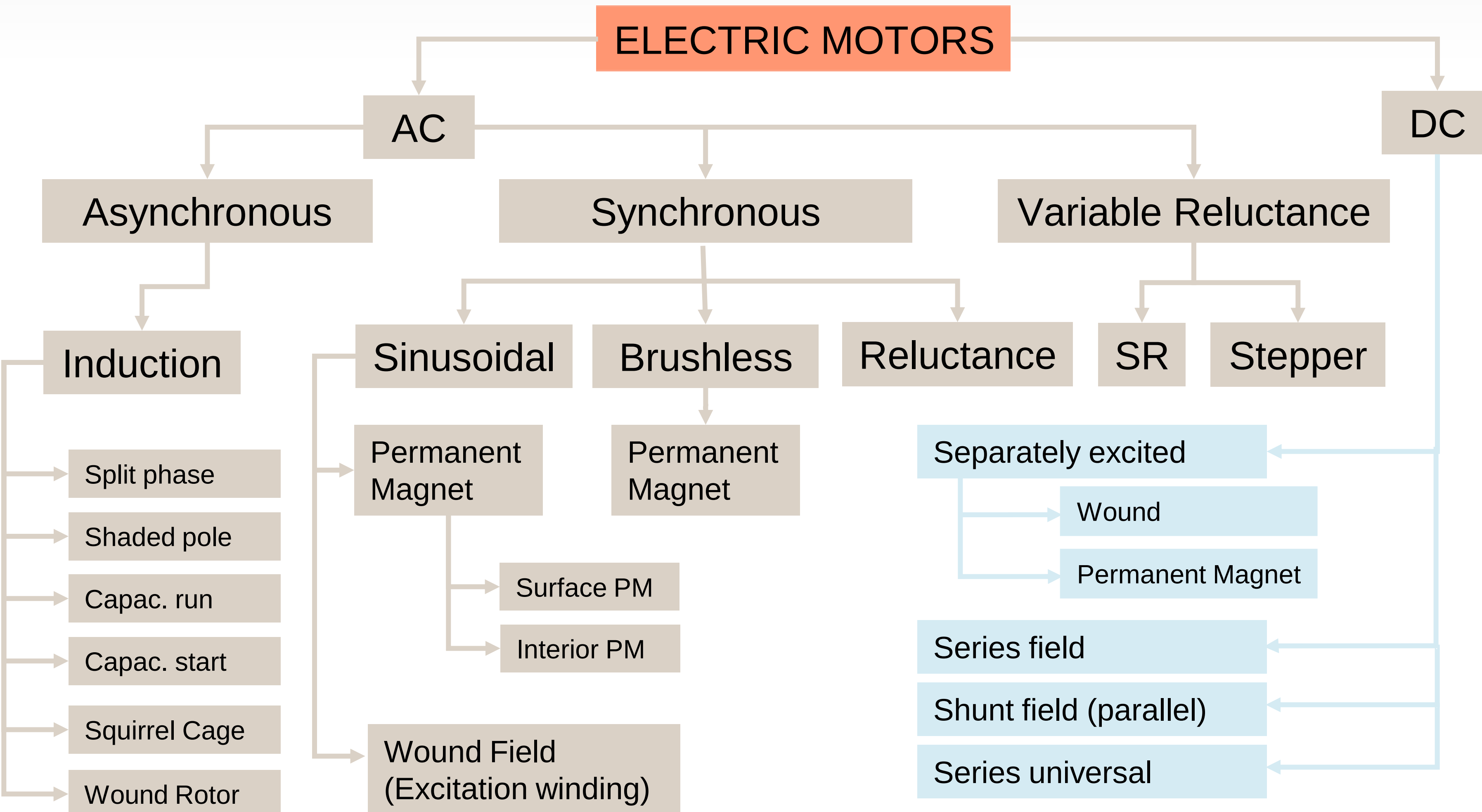
$$voltage = \frac{d\psi}{dt} = \frac{\partial \psi}{\partial \theta} \cdot \frac{d\theta}{dt} = \frac{\partial \psi}{\partial \theta} \cdot \omega_m$$

Ψ = flux *linkage* of winding

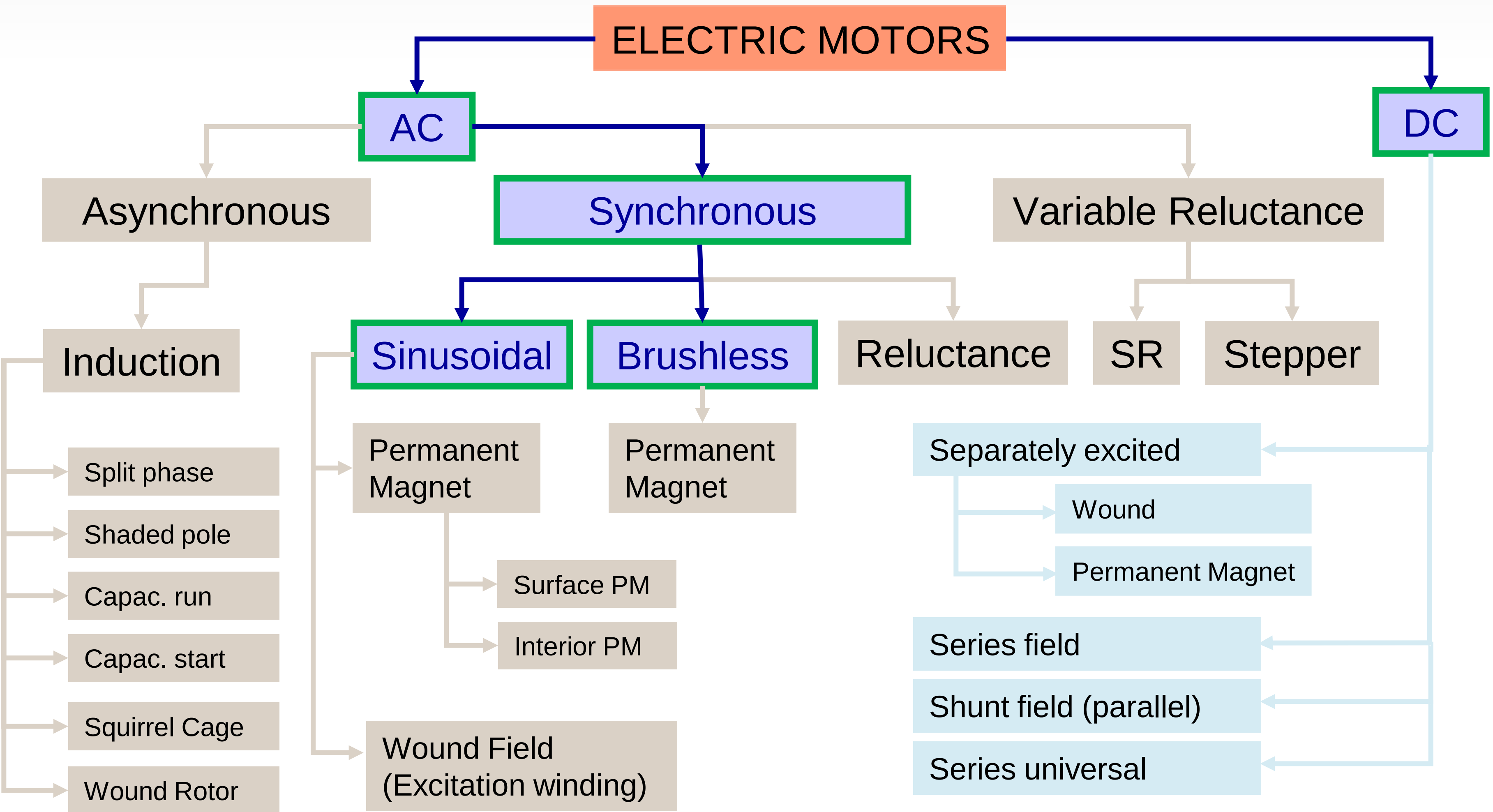
Maximum EMF occurs when $\alpha = \pm 90°$

freescale™

Electric Motor Clasification

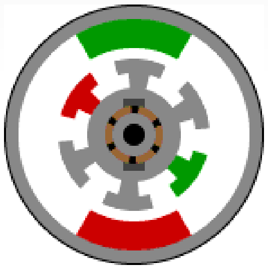


Electric Motor Clasification

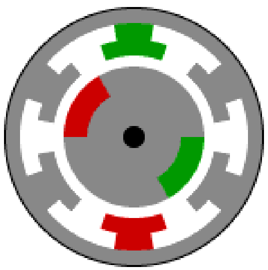


freescale™

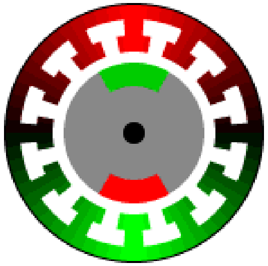
S12ZVM Motor targets



- DC Motors
  - Two or more permanent magnets in stator
  - Rotor windings connected to mechanical commutator



- BLDC Motors
  - PM in rotor, 3-phase conductors in stator
  - Trapezoidal back-EMF



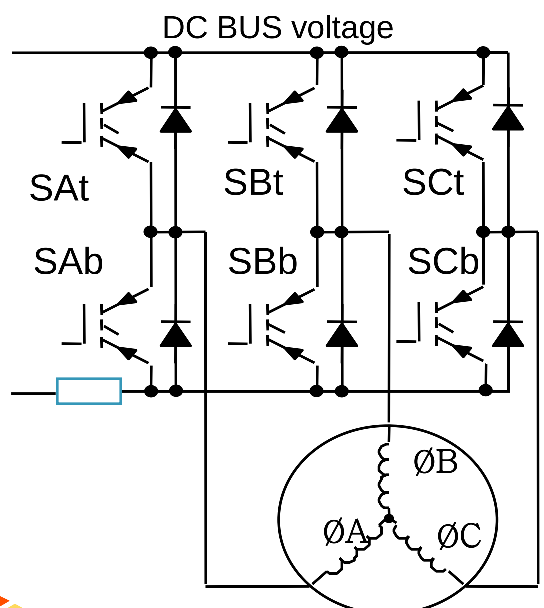
- Permanent Magnet Synchronous Motors
  - Similar to BLDC in construction
  - Sinusoidal back-EMF

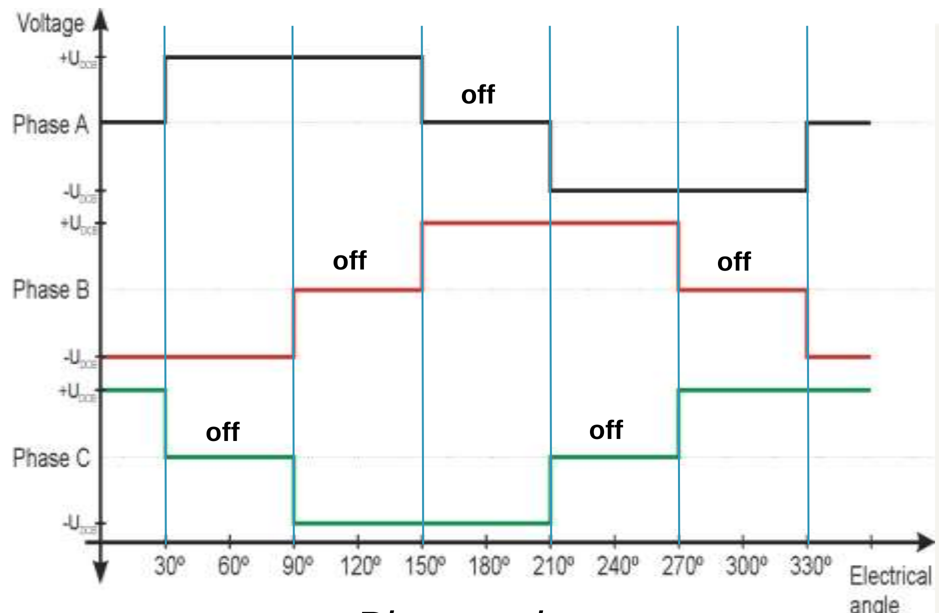
PM Machines – Trapezoidal vs. Sinusoidal

- The characteristic "Trapezoidal" or "Sinusoidal" is linked with the shape of the Back-EMF of the Permanent Magnet motor.
  - "Sinusoidal" means Synchronous (PMSM) motors
  - "Trapezoidal" means Brushless DC (BLDC) motors
- BLDC motor control (6-step control)
  - Only 2 of the 3 stator phases are excited at any time
  - 1 unexcited phase used as sensor (sensorless control)
- Synchronous motor (Field-oriented control)
  - All 3 phases are persistently excited at any time

BLDC Six-Step Commutation

- Commutation is performed by power transistors switching in synchronization with the rotor
  - Need to know the actual rotor position
- Voltage applied on two phases only, one phase remains unpowered



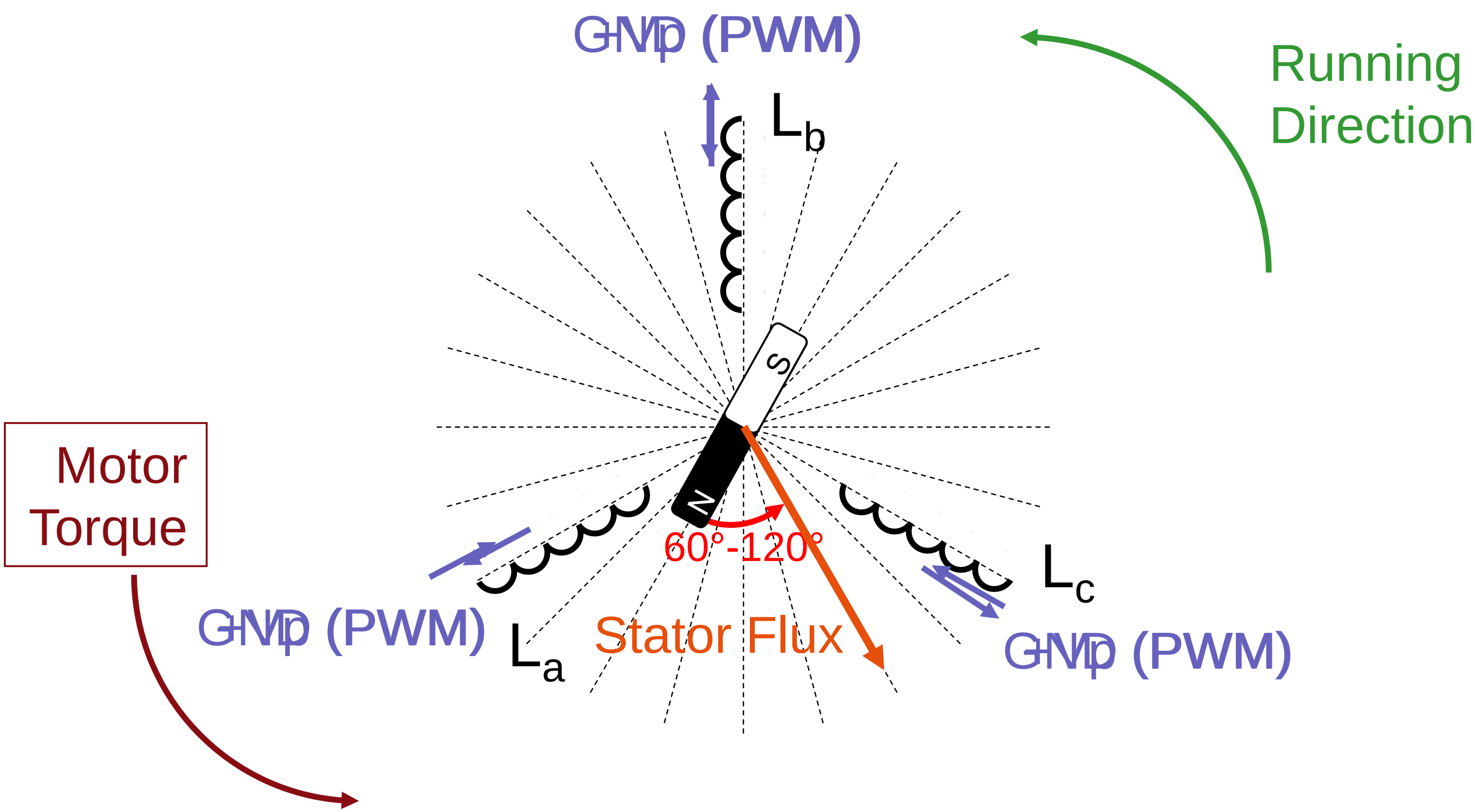


Phase voltages

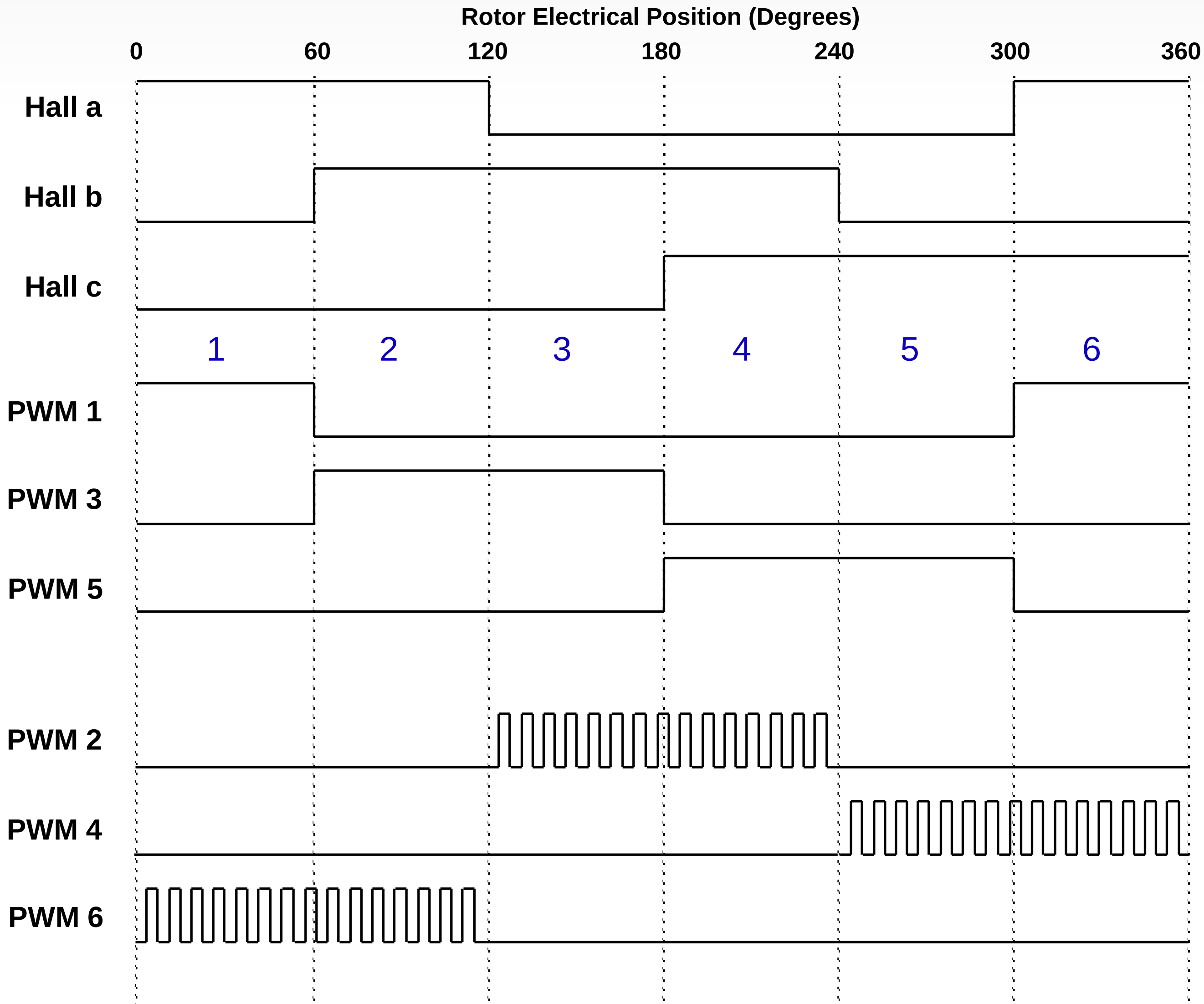
freescale™

BLDC Six-Step Commutation Principle

- Stator field is generated between 60° to 120° relative to the rotor field to get maximal torque and energy efficiency

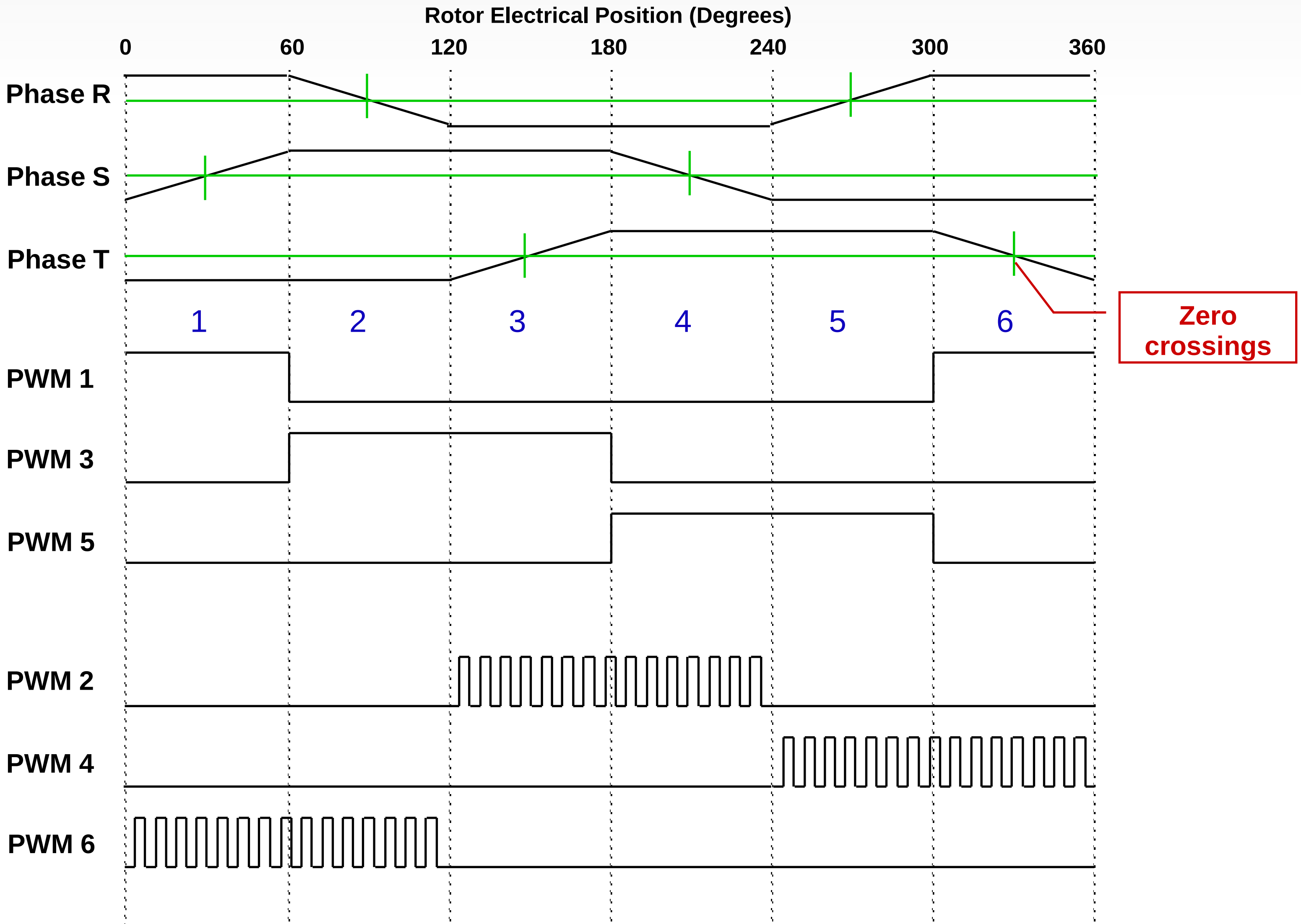


Sensor-based Commutation



freescale™

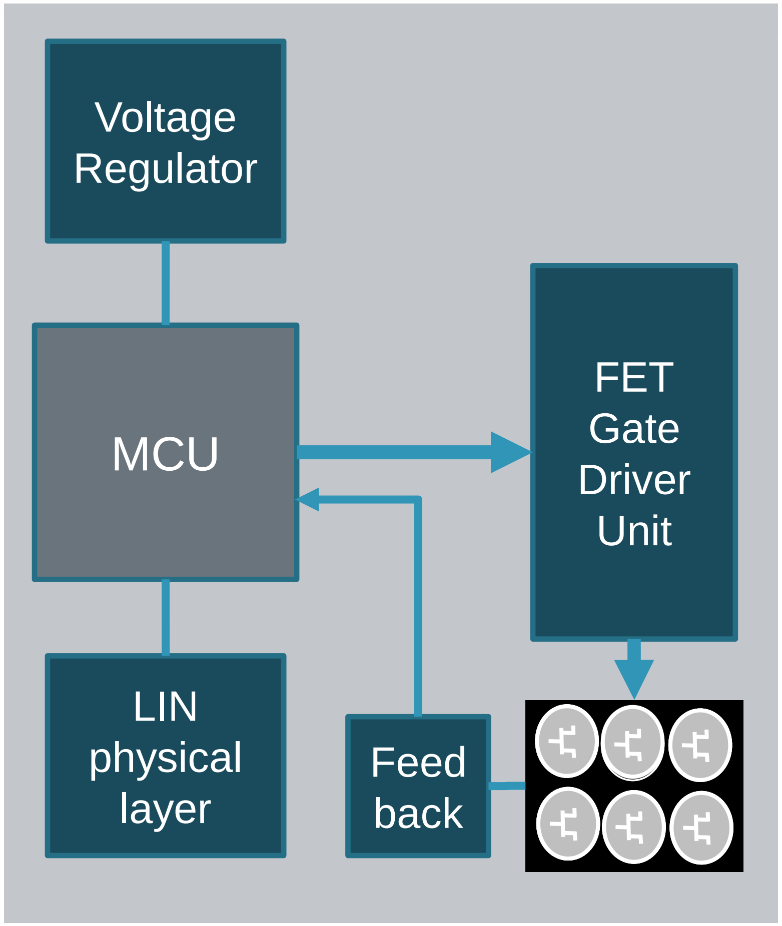
Sensor-less Commutation



MagniV S12ZVM

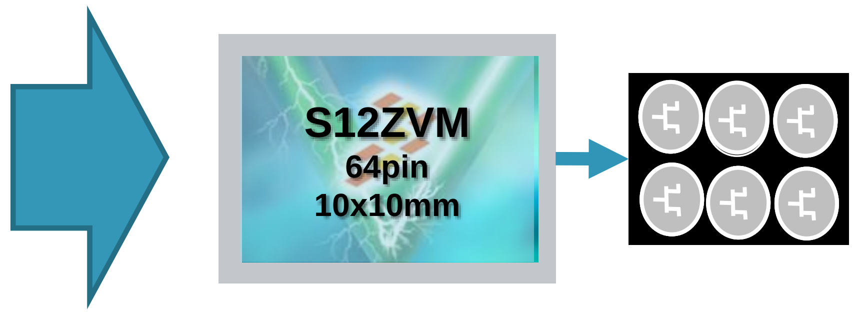
Device Overview

S12ZVM Family Concept



High level of integration:

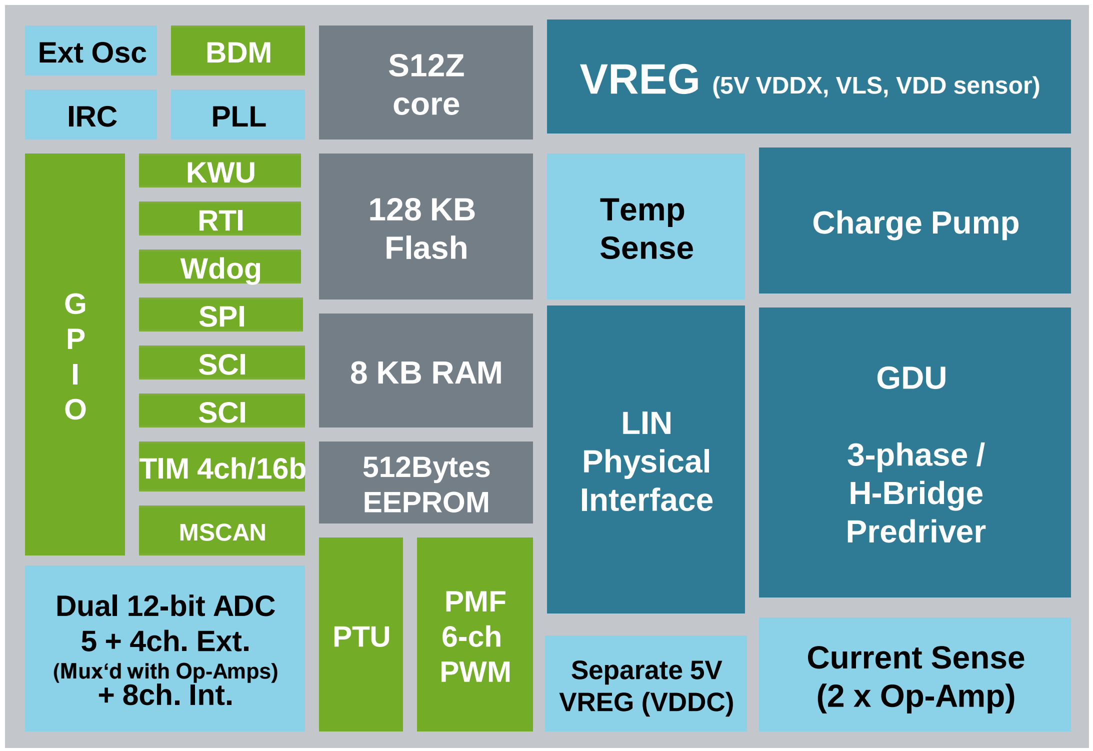
MCU, LIN physical layer, Voltage Regulator, FET pre-driver…all on the same die



Product Benefits:

- Reduced Bill of Materials
- Reduced PCB size/cost
- Fewer Points of Failure = Increased Quality

Overview of S12ZVM Feature Set

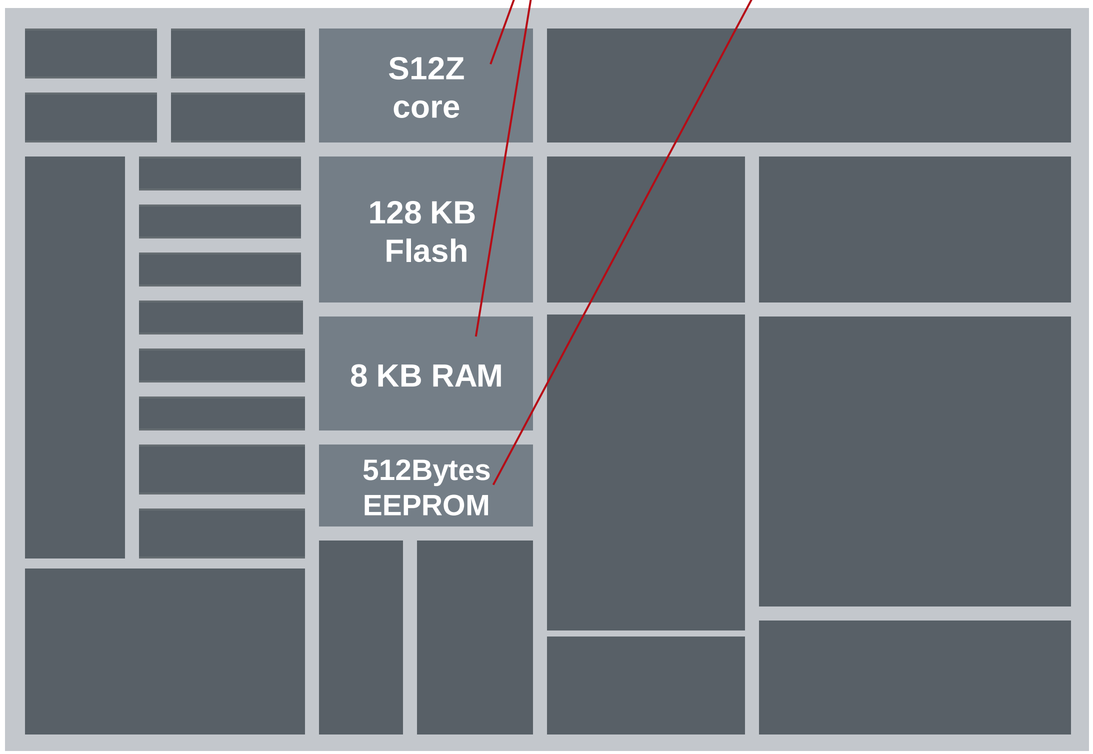


Overview of S12ZVM Feature Set

New S12Z CPU
- Up to 100MHz
- 32-bit ALU & 32-bit MAC unit,
- Optimized 32-bit math. operations

Safe RAM
- ECC

Flash& EEPROM
- ECC
- Memory Protection
- Margin Read



freescale™

Overview of S12ZVM Feature Set

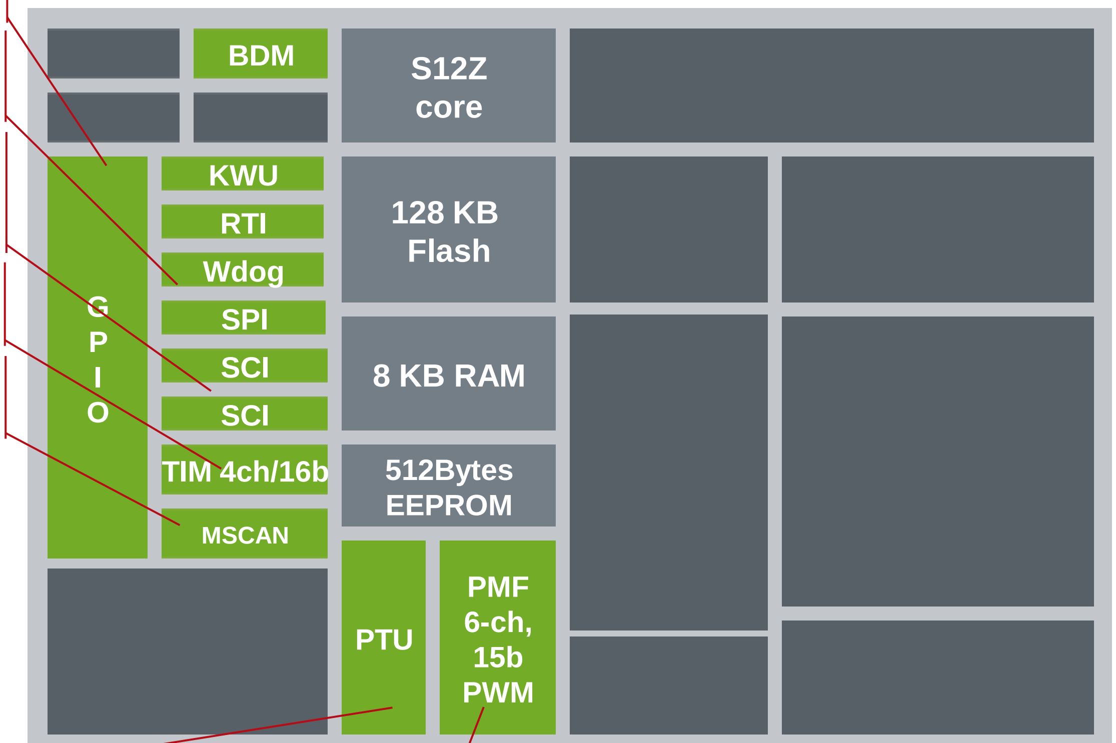
Sensor Supply
- 20mA I/O for sensor

Window Watchdog
- Independent RC osc

2 UARTs
- one linked to LIN Phy
- 2nd as test interface

Multiple timers
- IOC/periodic wakeup

CAN Option
- CAN controller

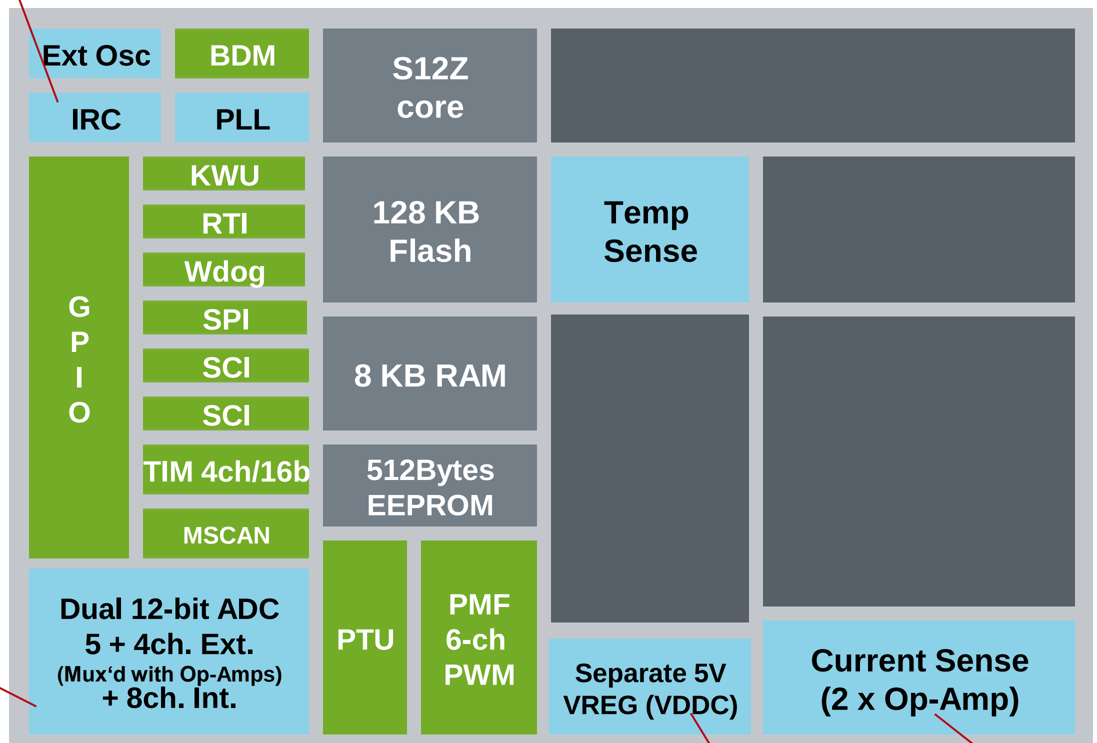


Programmable Trigger Unit
- Synchronize ADC to PWM
- Trigger Command list with up to 32 triggers per cycle

PWM Module
- Complementary mode with deadtime ctrl.
- Fault protection
- Double-Switching

Overview of S12ZVM Feature Set

Internal RC osc.
- +/-1.3% over tmp



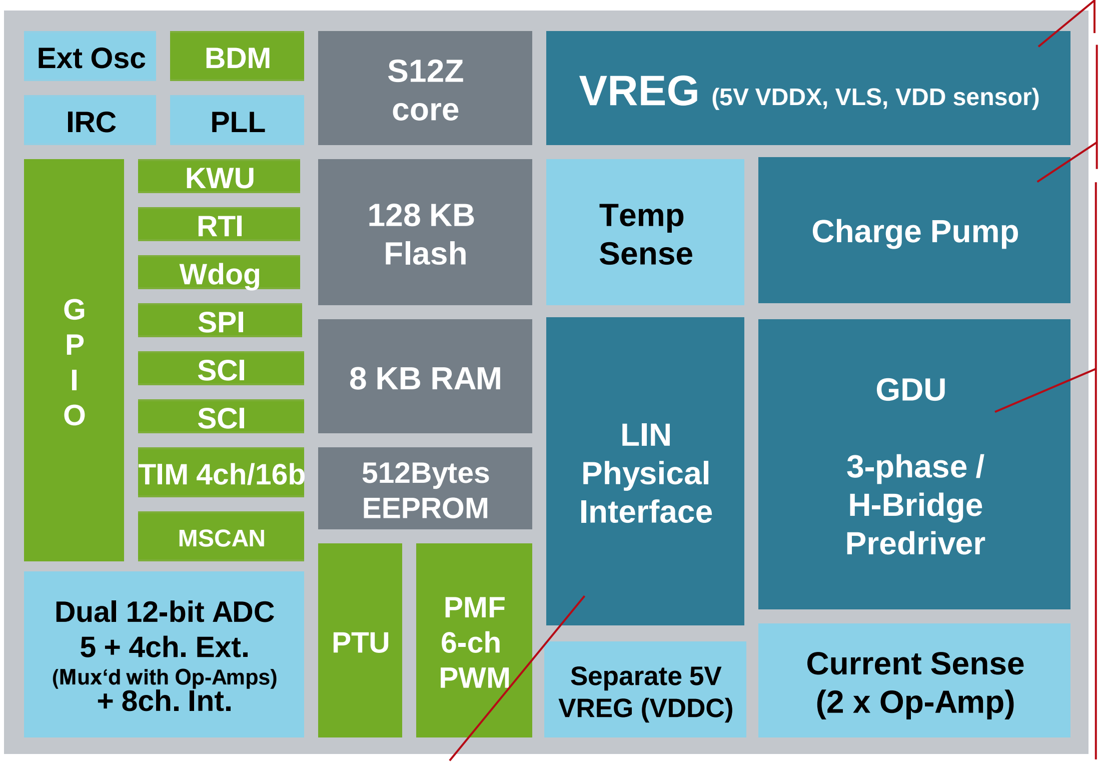
Two 12-bit ADC
- List Based Architecture allows flexible definition of order and number of conversions
- 2.5µs conv. time

CAN Support
- 5V Vreg controller for external transceiver

Current Sense Amplifiers
- 2-shunt system supported with additional selectable over-current protection comparator

freescale™

Overview of S12ZVM Feature Set



Voltage Regulator
- Boost Option
- Support for external ballast transistor
- Vbat sense

Charge Pump
- Optional to support 100% duty cycle

Gate Drive Unit
- Operational from 3.5V to 26V
- Bootstrap circuit based
- 11V Vreg
- Phase comparators
- Desaturation comp. for HS/LS protection
- Under-/Over-voltage detection
- DC-link and phase voltage internally accessible on ADC
- Selectable HS/LS slew rate

LIN Physical Interface
- 250kb/s fast mode
- +/-6kV

Overview of S12ZVM Feature Set

Internal RC osc.
- +/-1.3% over tmp

Sensor Supply
- 20mA I/O for sensor

Window Watchdog
- independent RC osc

2 UARTs
- ne linked to LIN Phy
- 2nd as test interface

Multiple timers
- IOC/periodic wakeup

CAN Option
- CAN controller

Two 12bit ADC
- List Based Architecture allows flexible definition of order and number of conversions
- 2.5µs conv. time

New S12Z CPU
- Up to 100MHz
- 32-bit ALU & 32-bit MAC unit,
- Optimized 32-bit math. operations

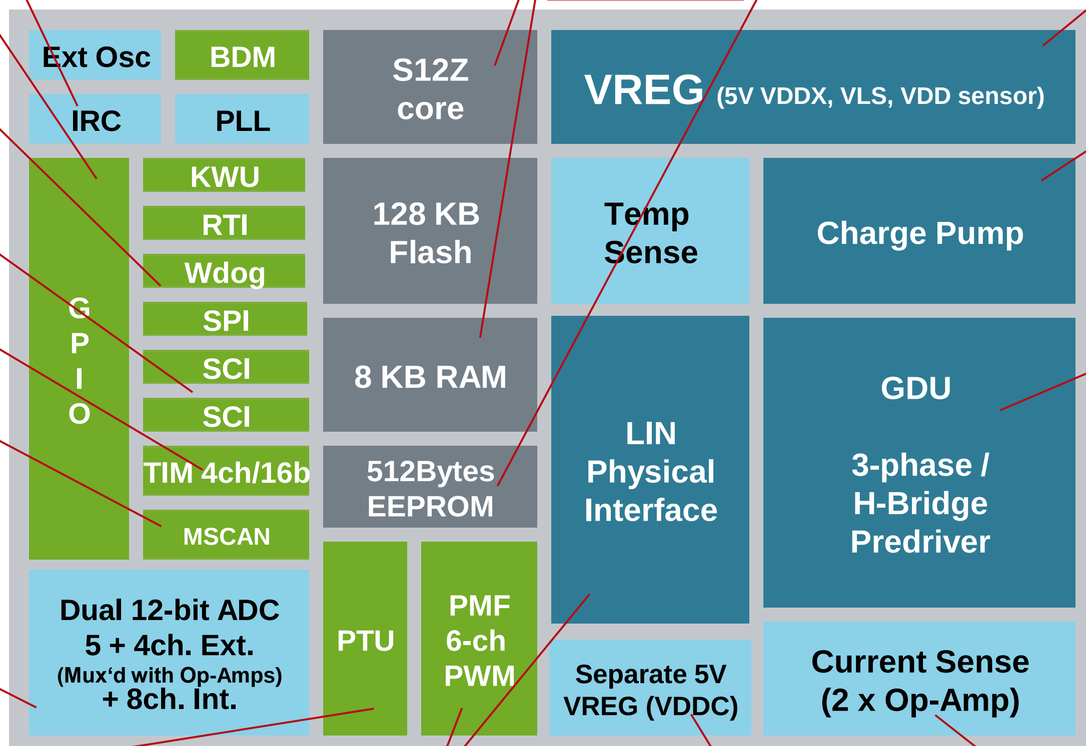
Safe RAM
ECC

Flash& EEPROM
- ECC
- Memory Protection
- Margin Read

Voltage Regulator
- Boost Option
- Support for external ballast transistor
- Vbat sense

Charge Pump
- Optional to support 100% duty cycle

Gate Drive Unit
- Operational from 3.5V to 26V
- Bootstrap circuit based
- 11V Vreg
- Phase comparators
- Desaturation comp. for HS/LS protection
- Under-/Over-voltage detection
- DC-link and phase voltage internally accessible on ADC
- Selectable HS/LS slew rate



Programmable Trigger Unit
- Synchronize ADC to PWM
- Trigger Command list with up to 32 triggers per cycle

PWM Module
- Complementary mode with deadtime ctrl.
- Fault protection
- Double-Switching

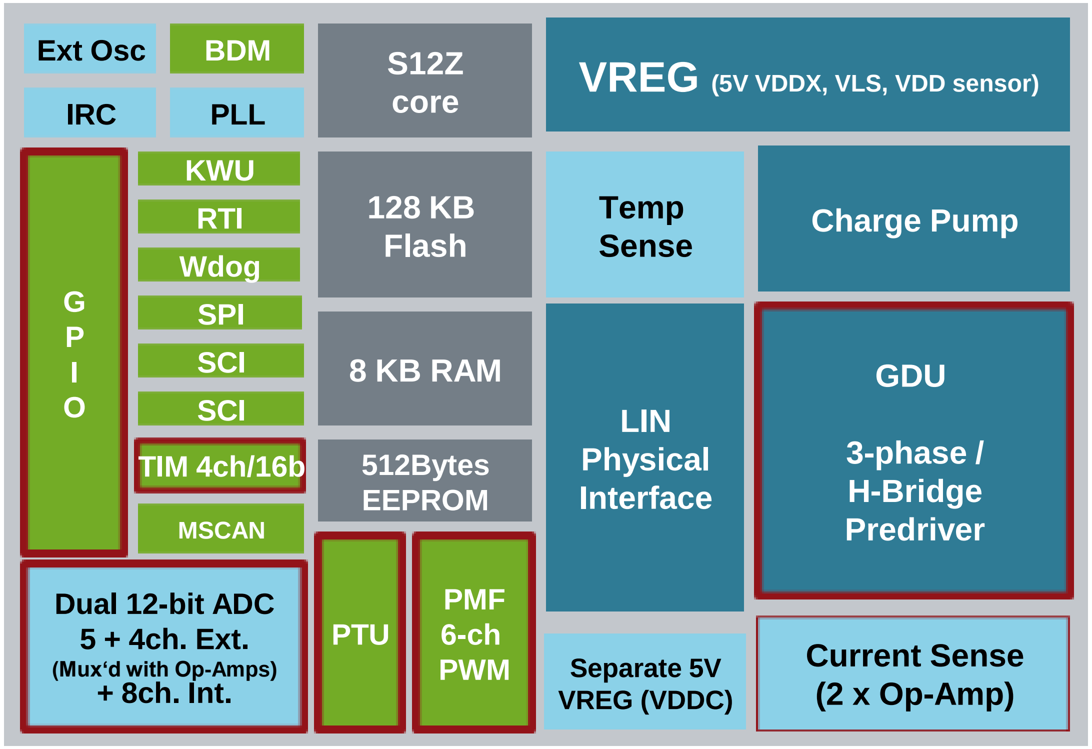
LIN Physical Interface
- 250kb/s fast mode
- +/-6kV

CAN Support
- 5V Vreg controller for external transceiver

Current Sense Amplifiers
- 2-shunt system supported with additional selectable over-current protection comparator

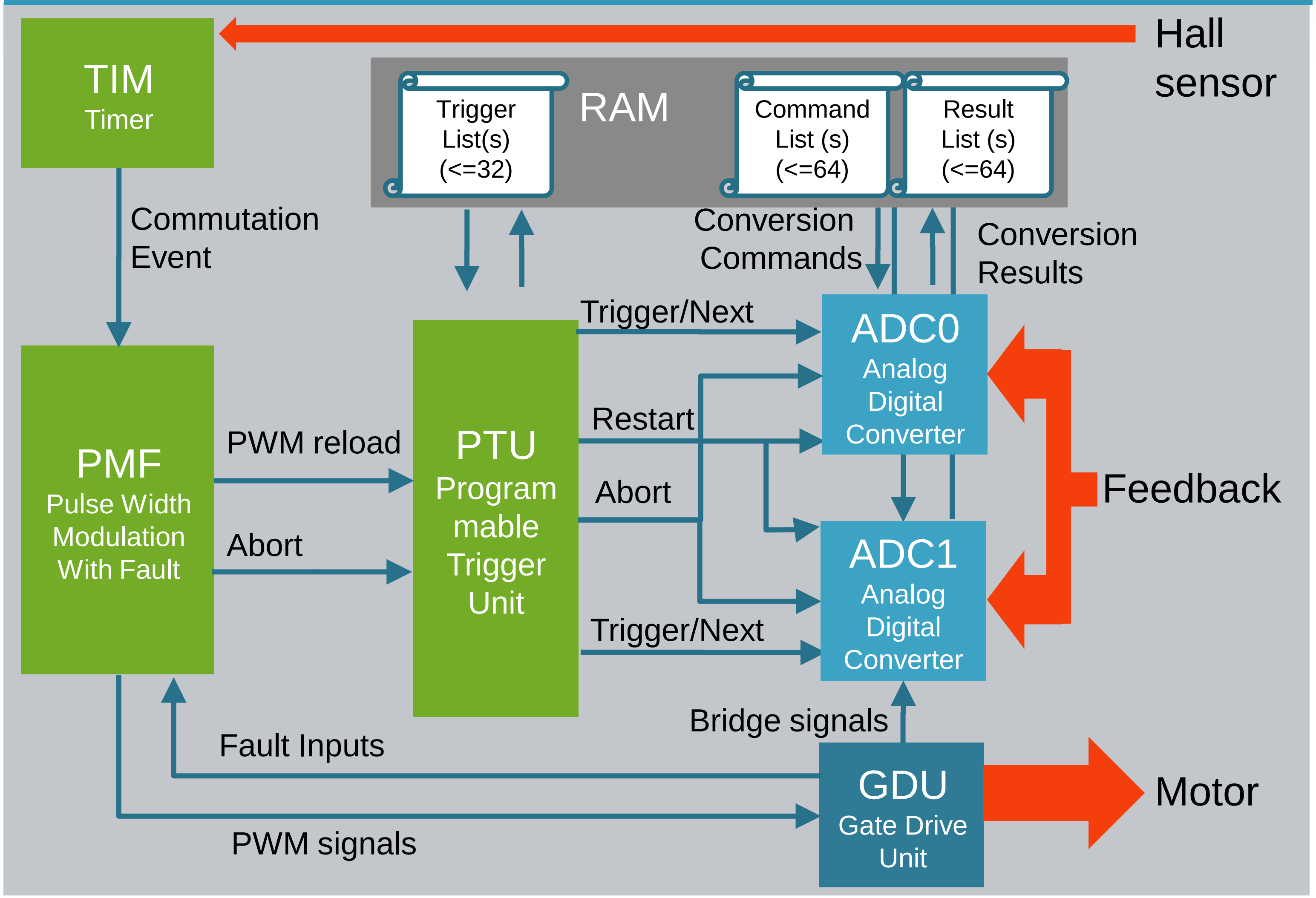
freescale™

S12ZVM Motor Control Loop Modules



Motor Control Loop Implementation

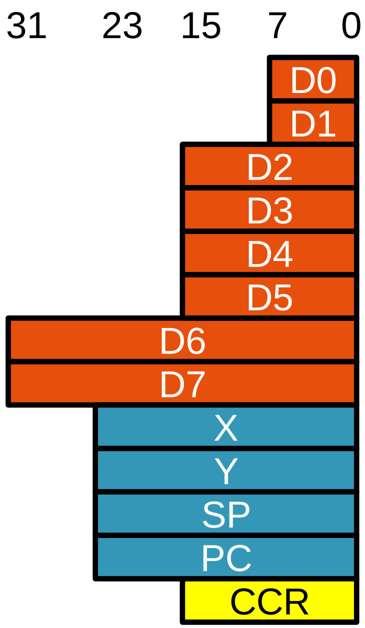
One control cycle can be a PWM cycle or a number of PWM cycles



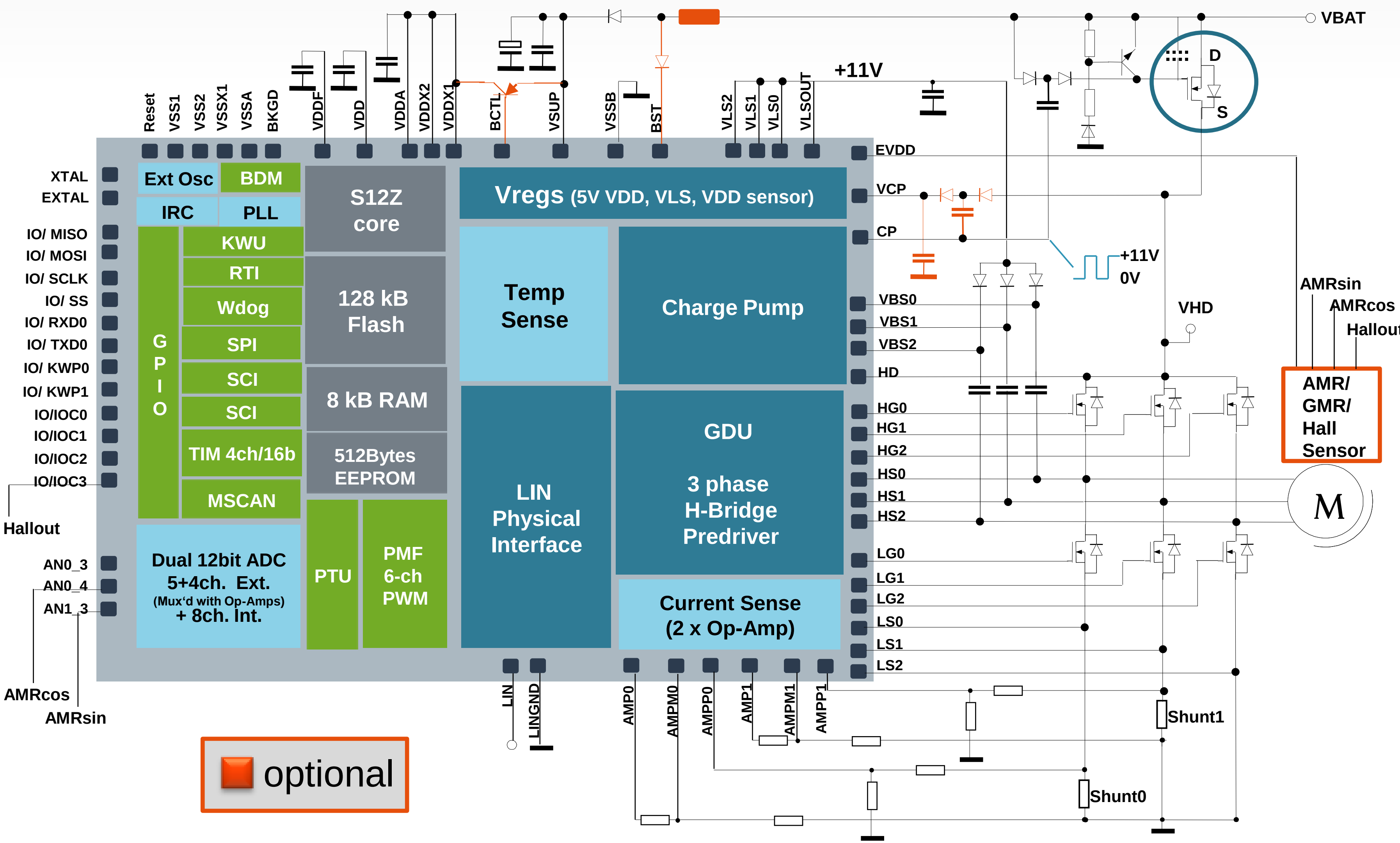
S12Z CPU Architecture… a 16-bit MCU?

- **24-bit = 16MByte linear address space (no paging)**
- **32-bit wide instruction and data bus in Harvard architecture**
- **32-bit ALU**
  - Single-cycle 16x16 multiply (2.5 cycles 32x32)
  - MAC unit 32-bit += 32-bit*32-bit (3.5 cycles)
  - Hardware divider 32-bit = 32-bit/32-bit   (18.5 cycles)
  - Single cycle multi-bit shifts (Barrel shifter)
  - Fractional Math support
- **CPU operates at 100MHz**
  - Optimized bus architecture with 100MHz load and store to RAM
  - NVM works with 1 Wait-state => effective 20ns accesses
- **Instructions and addressing modes optimized for C-Programming & Compiler**

Programmer's Model



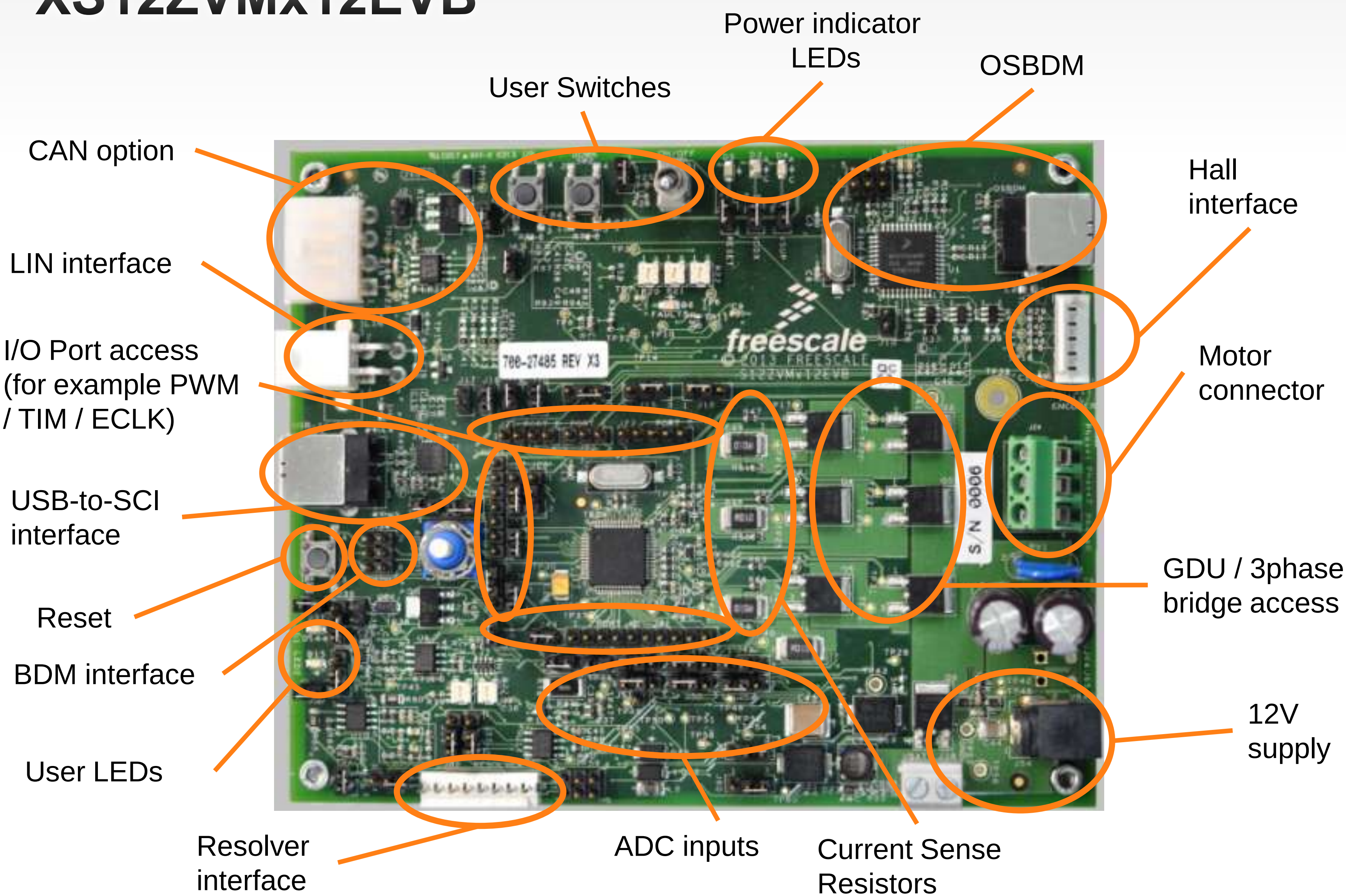
S12ZVML Application Schematic



Development Tools

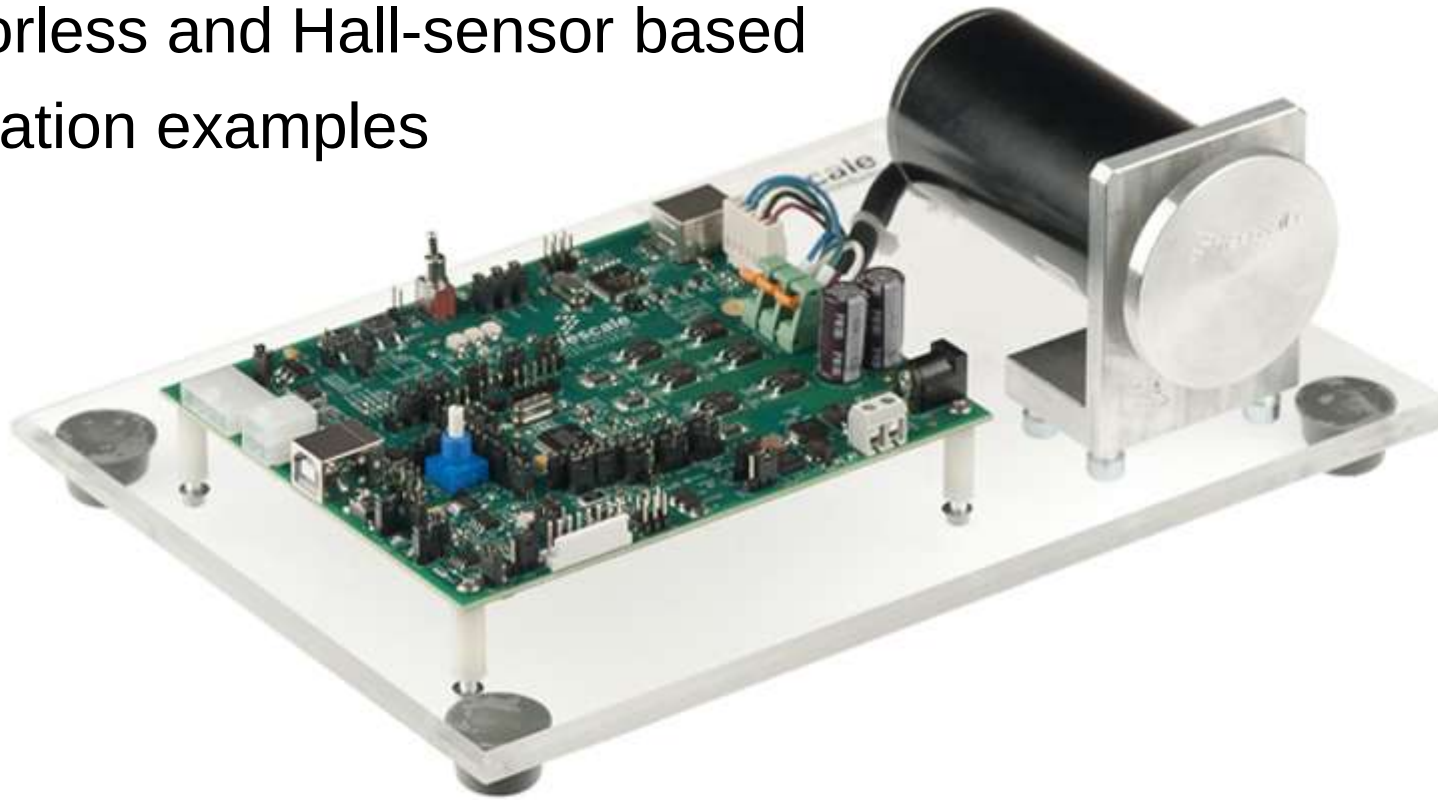
Hardware + Software

XS12ZVMx12EVB



MTRCKTSBNZVM128 BLDC Motor Control Kit

- S12ZVML12EVBLIN
- 4 pole-pair Linix 45ZWN24-90 motor
- Sensorless and Hall-sensor based application examples

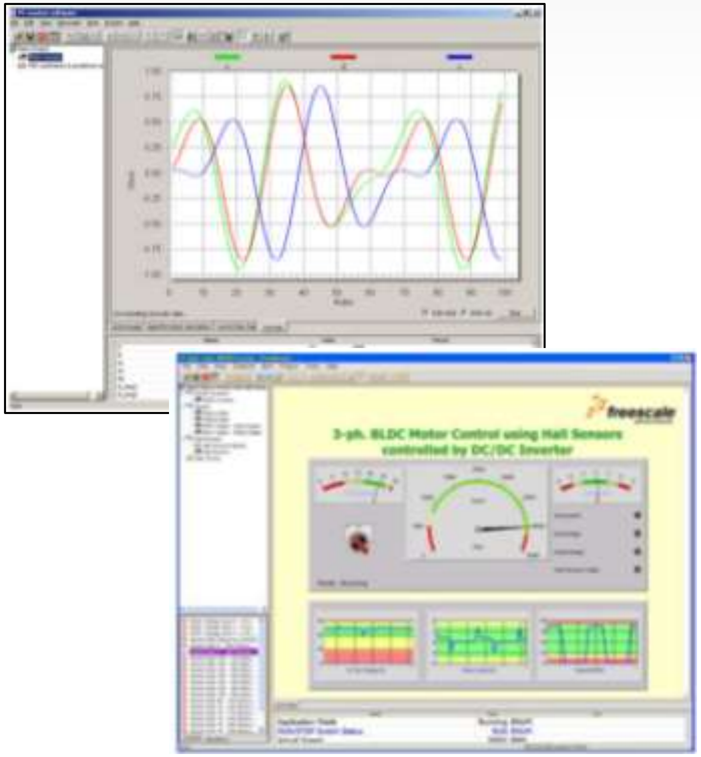


http://www.freescale.com/webapp/sps/site/overview.jsp?code=AUTOMCDEVKITS

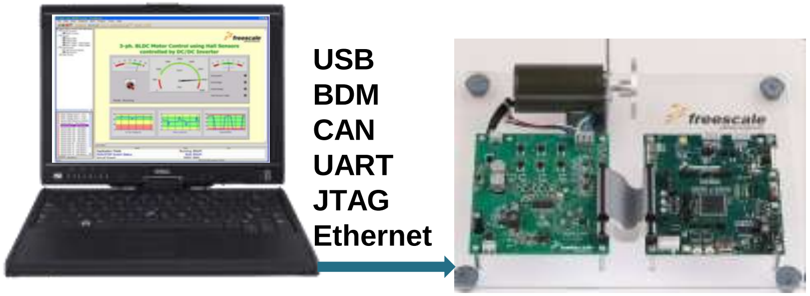
freescale™

FreeMASTER – Run Time Debugging Tool

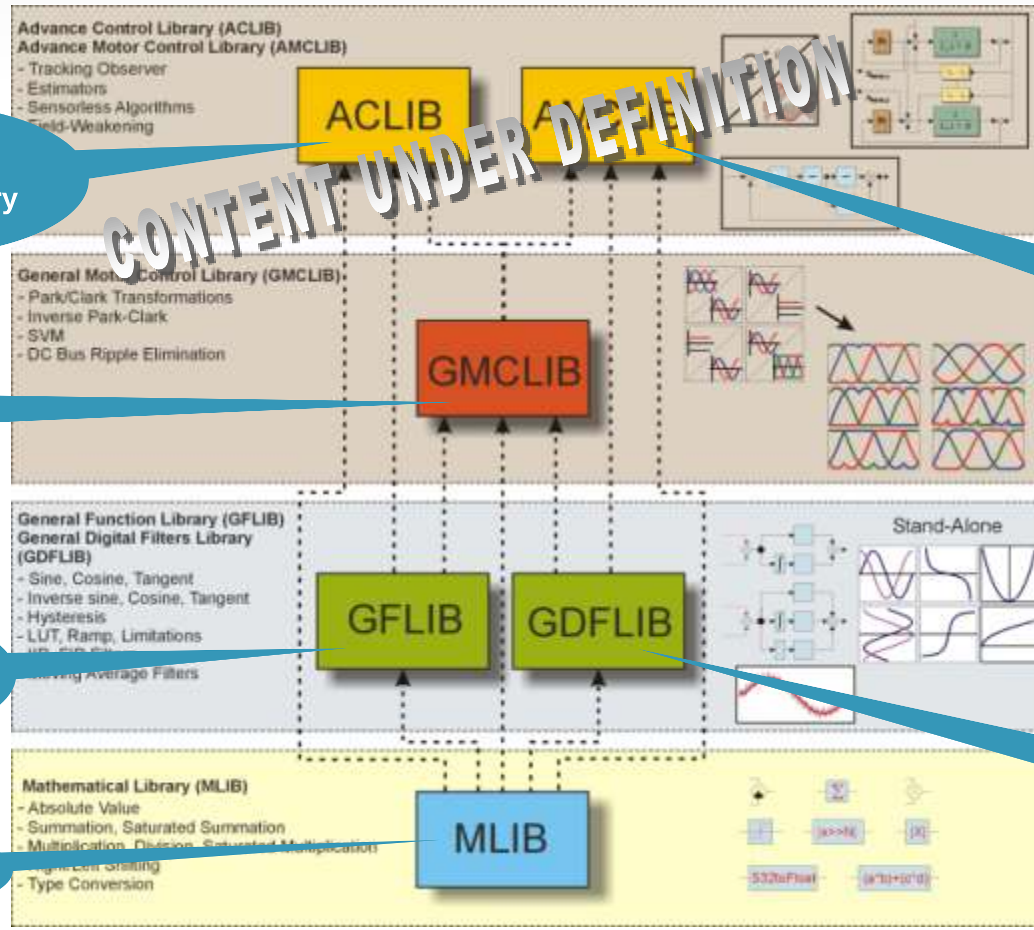
- User-friendly tool for real-time debug monitor and data visualization
  - Completely non-intrusive monitoring of variables on a running system
  - Display multiple variables changing over time on an oscilloscope-like display, or view the data in text form
  - Communicates with an on-target driver via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



http://www.freescale.com/freemaster



Automotive Math and Motor Control Library Set – Block Diagram



Advanced Control Library

Advanced Motor Control Library

General Motor Control Library

General Function Library

General Digital Filters Library

Mathematical Library

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=AUTOMATH_MCL

freescale™

Auto Math and Motor Control Library Contents

MLIB

- **Absolute Value, Negative Value**
  - MLIB_Abs, MLIB_AbsSat
  - MLIB_Neg, MLIB_NegSat
- **Add/Subtract Functions**
  - MLIB_Add, MLIB_AddSat
  - MLIB_Sub, MLIB_SubSat
- **Multiply/Divide/Add-multiply Functions**
  - MLIB_Mul, MLIB_MulSat
  - MLIB_Div, MLIB_DivSat
  - MLIB_Mac, MLIB_MacSat
  - MLIB_VMac
- **Shifting**
  - MLIB_ShL, MLIB_ShLSat
  - MLIB_ShR
  - MLIB_ShBi, MLIB_ShBiSat
- **Normalisation, Round Functions**
  - MLIB_Norm, MLIB_Round
- **Conversion Functions**
  - MLIB_ConvertPU, MLIB_Convert

GFLIB

- **Trigonometric Functions**
  - GFLIB_Sin, GFLIB_Cos, GFLIB_Tan
  - GFLIB_Asin, GFLIB_Acos, GFLIB_Atan, GFLIB_AtanYX
- **Limitation Functions**
  - GFLIB_Limit, GFLIB_VectorLimit
  - GFLIB_LowerLimit, GFLIB_UpperLimit
- **PI Controller Functions**
  - GFLIB_ControllerPIr, GFLIB_ControllerPIrAW
  - GFLIB_ControllerPIp, GFLIB_ControllerPIpAW
- **Interpolation**
  - GFLIB_Lut1D, GFLIB_Lut2D
- **Hysteresis Function**
  - GFLIB_Hyst
- **Signal Integration Function**
  - GFLIB_IntegratorTR
- **Sign Function**
  - GFLIB_Sign
- **Signal Ramp Function**
  - GFLIB_Ramp
- **Square Root Function**
  - GFLIB_Sqrt

GDFLIB

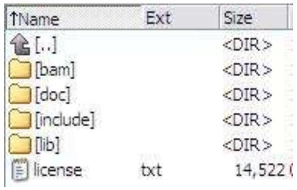
- **Finite Impulse Filter**
  - GDFLIB_FilterFIR
- **Moving Average Filter**
  - GDFLIB_FilterMA
- **1st Order Infinite Impulse Filter**
  - GDFLIB_FilterIIR1init
  - GDFLIB_FilterIIR1
- **2nd Order Infinite Impulse Filter**
  - GDFLIB_FilterIIR2init
  - GDFLIB_FilterIIR2

GMCLIB

- **Clark Transformation**
  - GMCLIB_Clark
  - GMCLIB_ClarkInv
- **Park Transformation**
  - GMCLIB_Park
  - GMCLIB_ParkInv
- **Duty Cycle Calculation**
  - GMCLIB_SvmStd
- **Elimination of DC Ripples**
  - GMCLIB_ElimDcBusRip
- **Decoupling of PMSM Motors**
  - GMCLIB_DecouplingPMSM

ACLIB/AMCLIB

- **Angle Tracking Observer**
- **Tracking Observer**
- **PMSM BEMF Observer in Alpha/Beta**
- **PMSM BEMF Observer in D/Q**
- **Content To Be Defined**

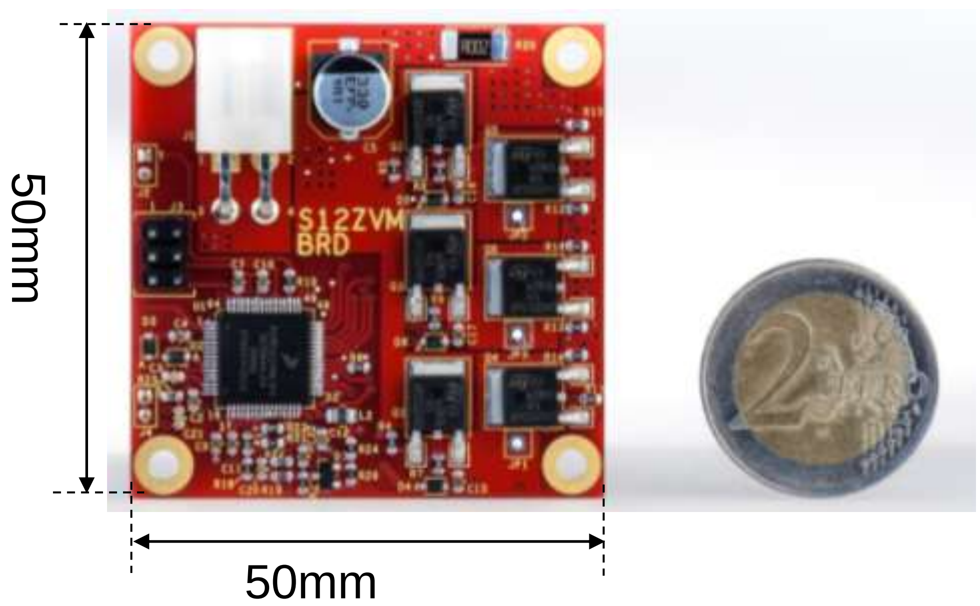


Delivery Content

→ Matlab/Simulink Bit Accurate Models
→ User Manuals
→ Header files
→ Compiled Library File
→ License File (to be accepted at install time)

Coming Soon

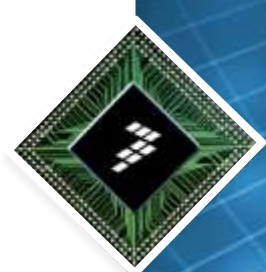
- S12ZVM Demonstration board
  - Full 3-phase BLDC system
  - 5cm x 5cm small form factor



freescale™

Starting an S12ZVM Project in CodeWarrior

Hands-on session #1



Hands-on session #1 – Objectives

- In this session you will:
  - Import an existing project into CodeWarrior for MCU 10.4
  - Configure the programming/debugging interface in CodeWarrior
  - Run a simple program on the S12ZVM EVB
  - Watch variables in the debugging interface

freescale™

Hardware Setup

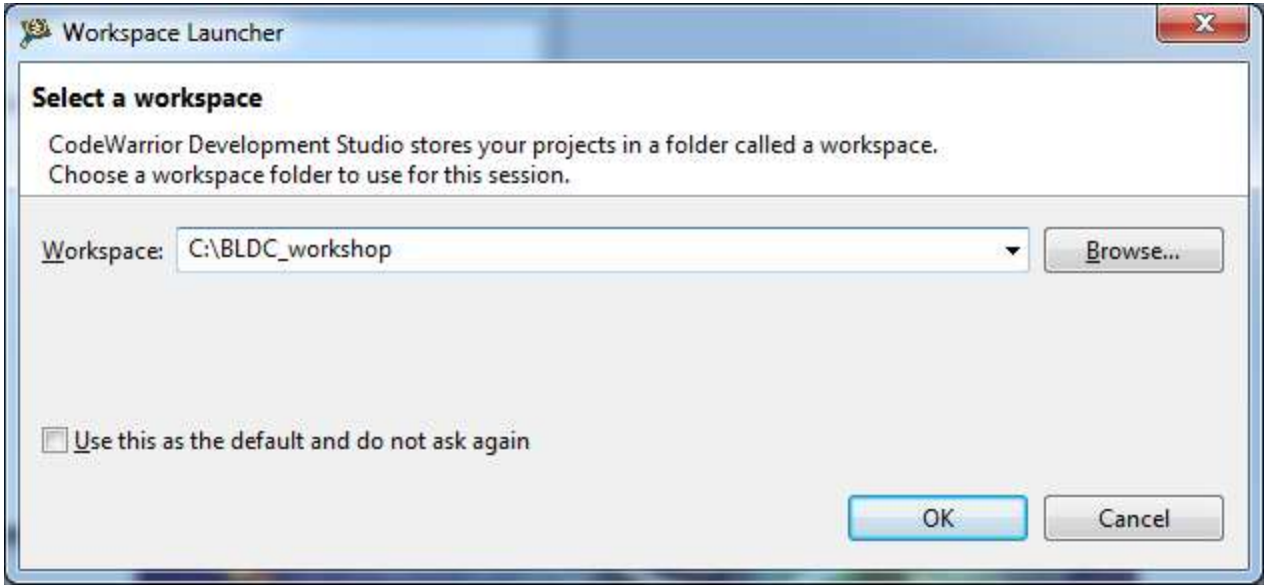
- EVB jumpers:
  - Move jumper on J47 so that it connects pins 2-3
  - Move jumpers on J27/J28 so that each one connects pins 1-2

- Connect the USB cable from the OSBDM interface, J7, to the computer. This is a single interface for:
  - Programming the MCU via the BDM and
  - Communicating with FreeMaster via SCI port

- Connect the 12V power supply to the VBAT input, J54

Initial Configuration

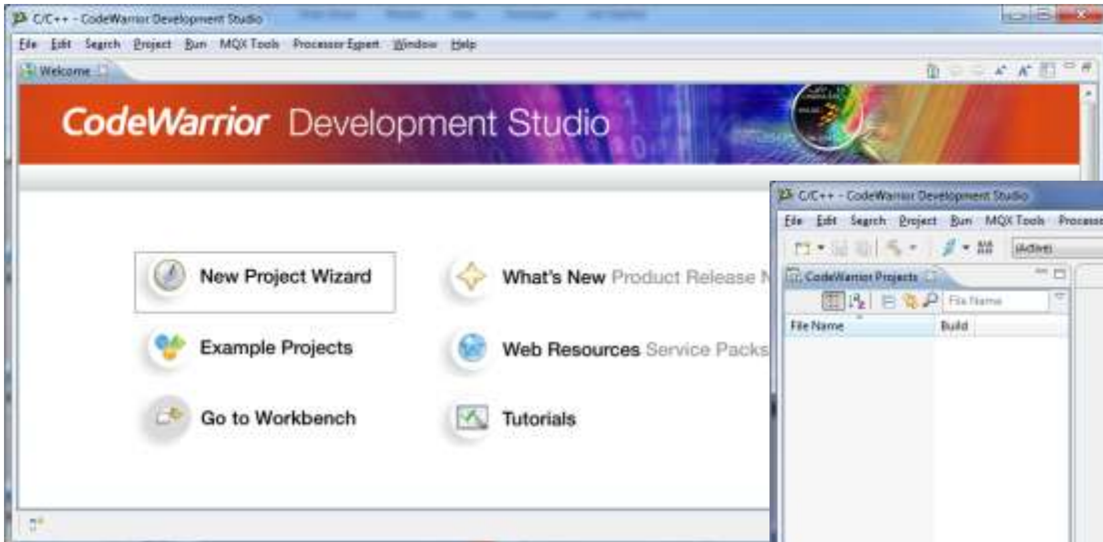
- Software:
  - From the Start menu select **Code Warrior MCU 10.4**
  - Select a workspace and click OK

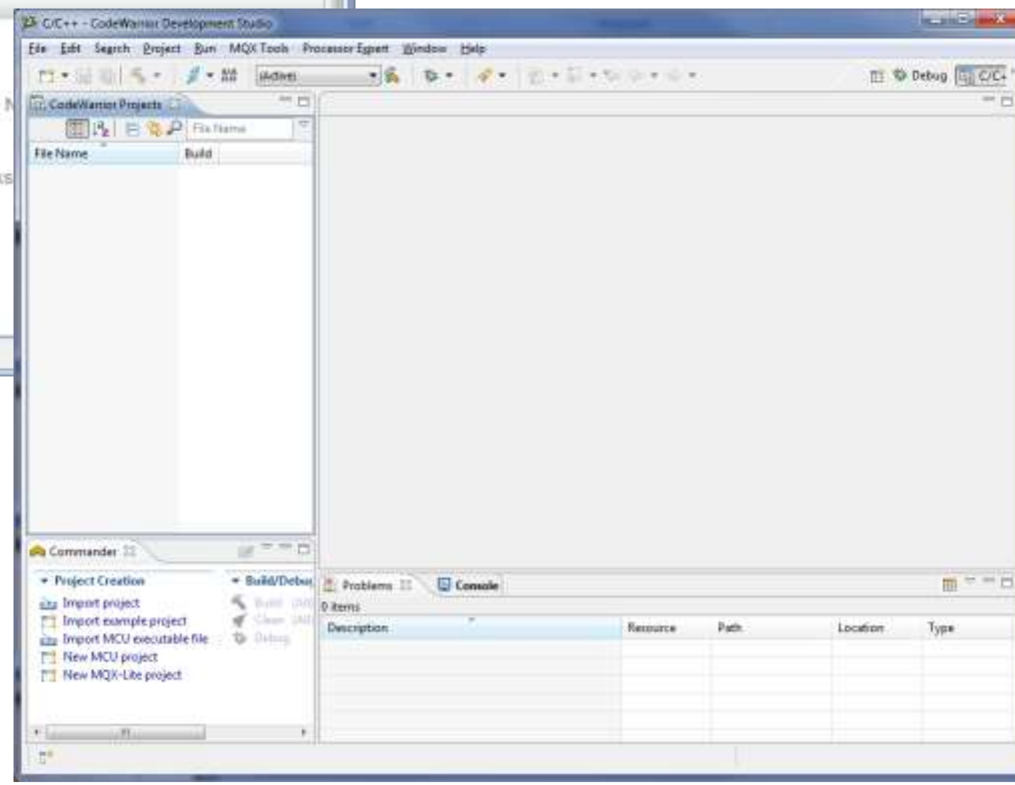


freescale™

Launching the Workbench in CodeWarrior

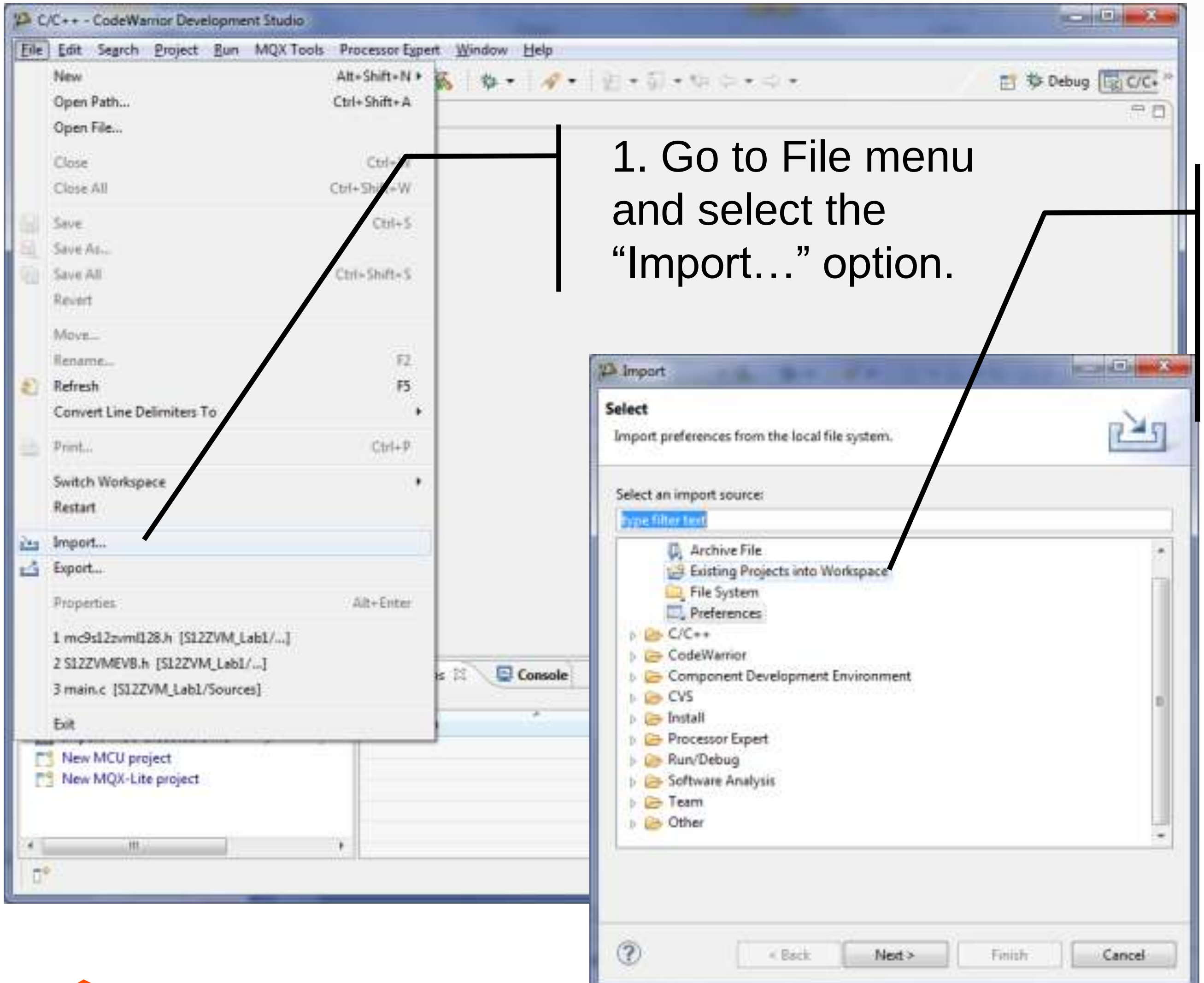
- From the main page, select "**Go to Workbench**"





The workbench will open with the CodeWarrior Projects view enabled

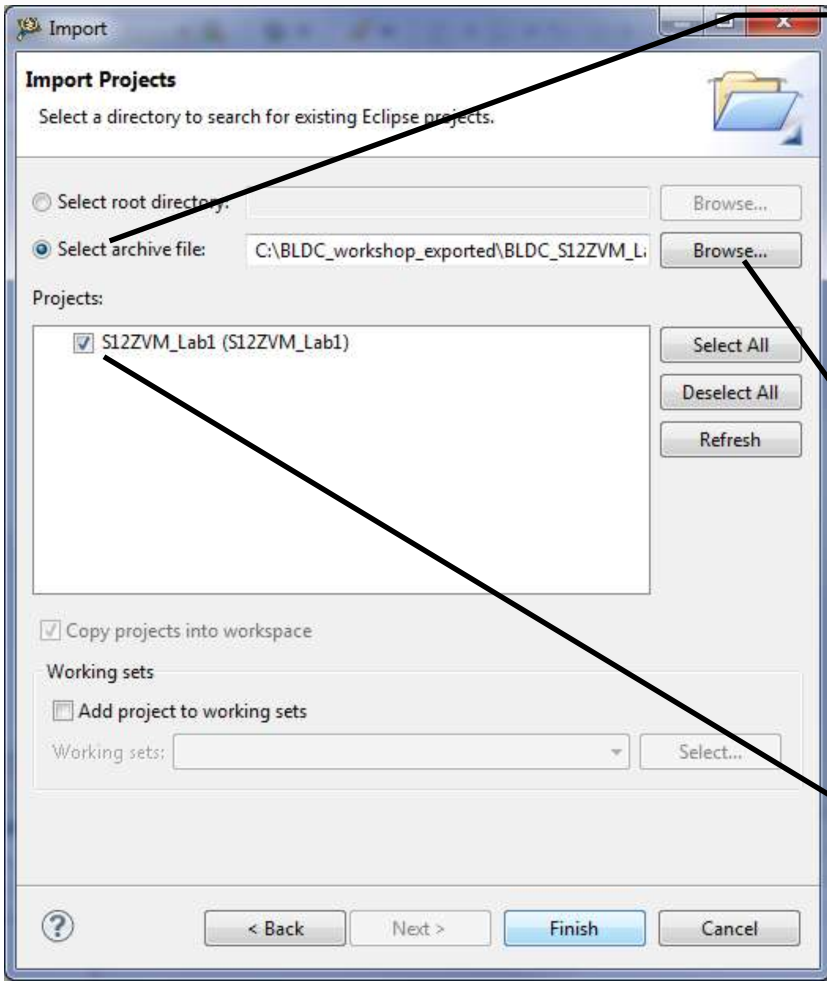
Import a project



1. Go to File menu and select the “Import…” option.

2. In the Import dialog window, select option: “Existing Projects into Workspace”

Import a project (continued)



3. Click on “Select archive file:”

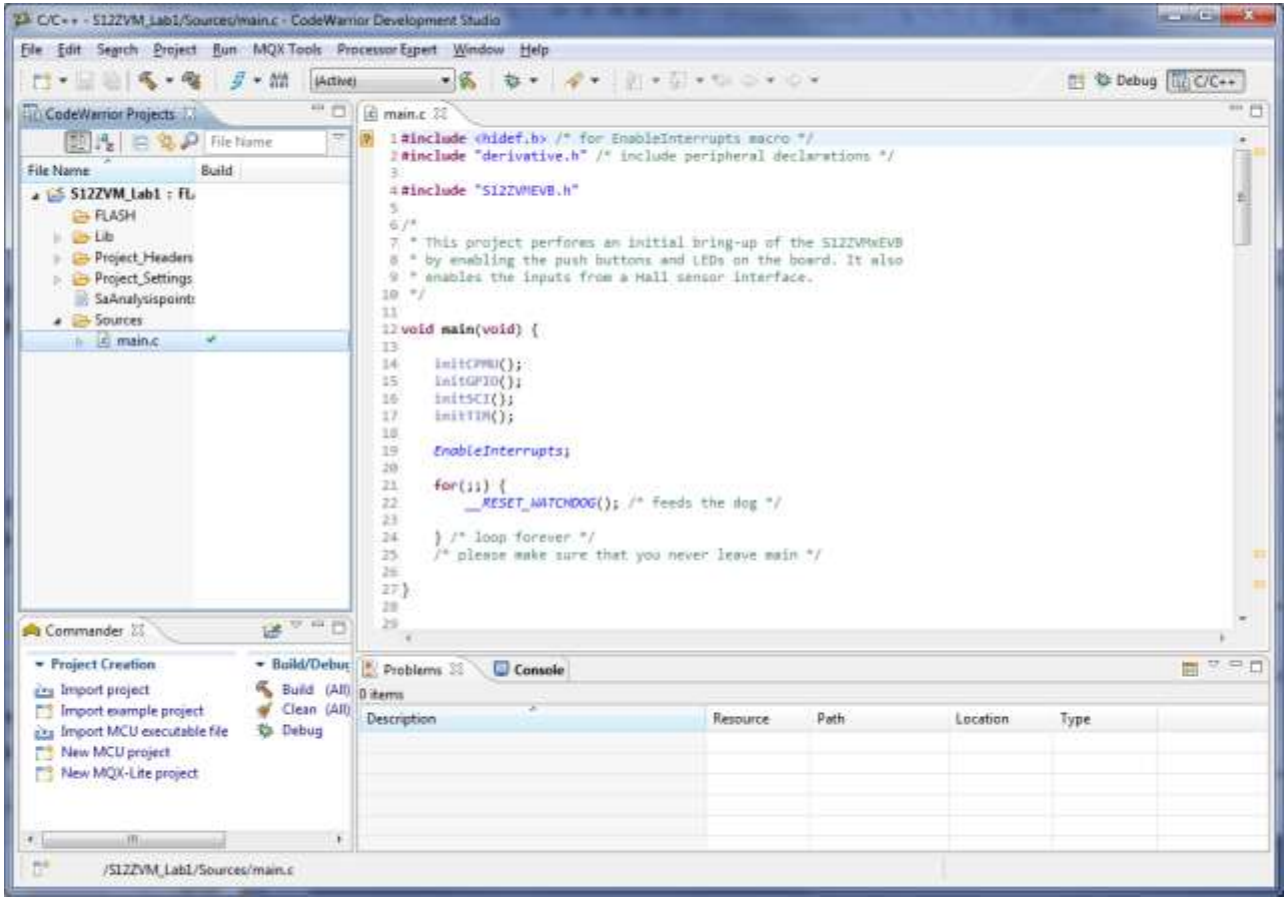
4. Click on “Browse” and look for the zip project file in C:\BLDC_workshop_exported \BLDC_S12ZVM_Lab1.zip

5. Make sure the checkbox is marked, then click on “Finish”

freescale™

The project window

- The example project source file can be found under “Sources”, and it is called “main.c”



Import project into CodeWarrior

- This simple project will:
  - Configure the CPMU to 25MHz core clock / 12.5 MHz bus clock
  - Configure channel 0 of the TIM timer module to a 1millisecond time base
  - Configure channel 1 of the TIM timer module as an XORed input of the hall sensor interface
  - Configure the SCI to 9600 bps for future utilization
  - Toggle LED 1 every 1 millisecond based on TIM channel 0
  - Toggle LED 2 with every edge of the Hall sensor inputs

Example project – header files

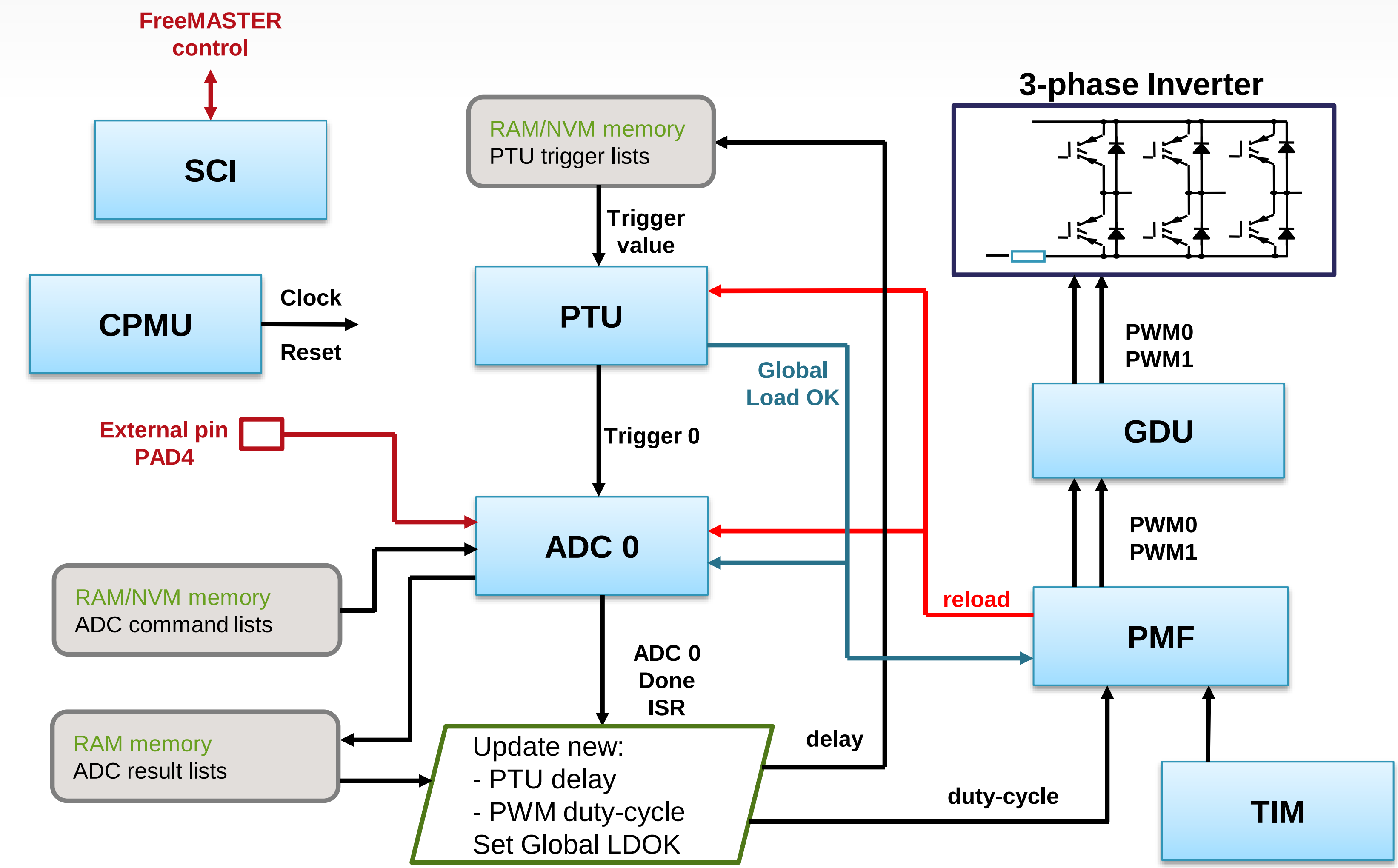
```
#include <hidef.h>         /* for EnableInterrupts macro */
#include "derivative.h"    /* include peripheral declarations */

#include "S12ZVMEVB.h"      /* EVB definitions */

/* Global variables to store each Hall sensor input */
unsigned char hall_pattern = 0;
unsigned char hall_a_input = 0;
unsigned char hall_b_input = 0;
unsigned char hall_c_input = 0;
```

Example project – Main()

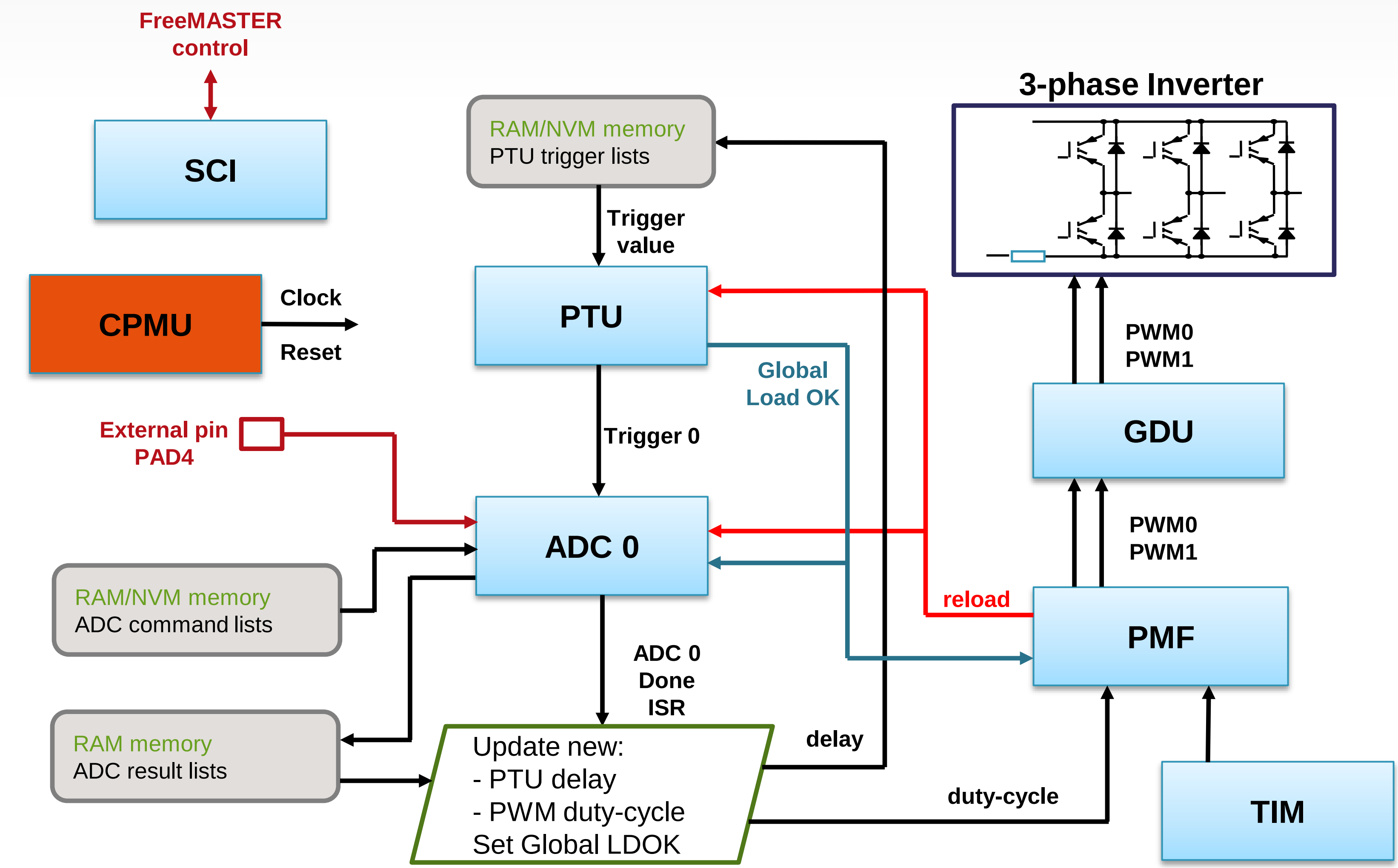
```
void main(void) {

   initCPMU();     /* configuration for 25MHz core clock */
   initGPIO();     /* configure pins for LED and Hall interface on board */
   initSCI();      /* initialize SCI port 1 at 9600bps (for future use) */
   initTIM();      /* initialize timer channels */


   EnableInterrupts;

   for(;;) {
     __RESET_WATCHDOG();/* feeds the dog */

     hall_a_input = 0x01 & (hall_pattern>>0);
     hall_b_input = 0x01 & (hall_pattern>>1);
     hall_c_input = 0x01 & (hall_pattern>>2);

   } /* loop forever */
}        /* please make sure that you never leave main */
```

freescale™

Application Block Diagram

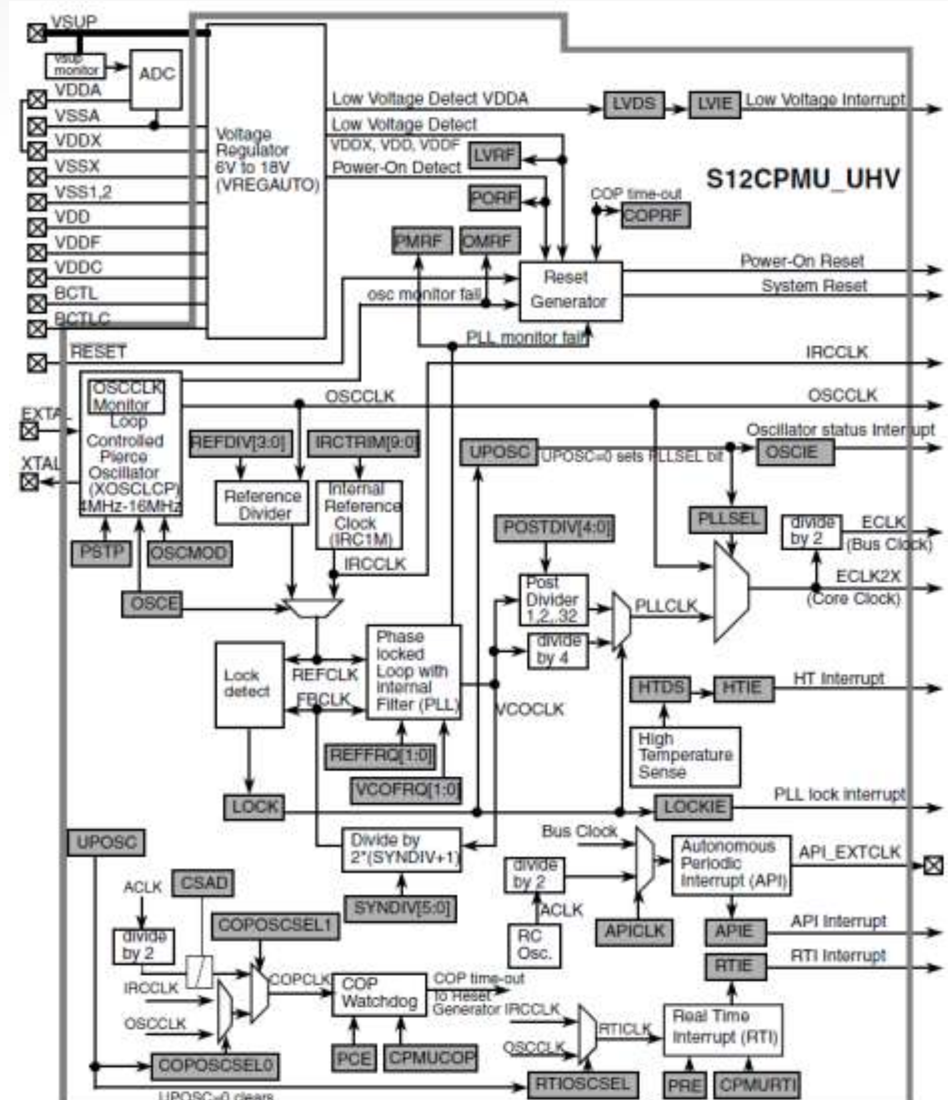


Application Block Diagram - CPMU



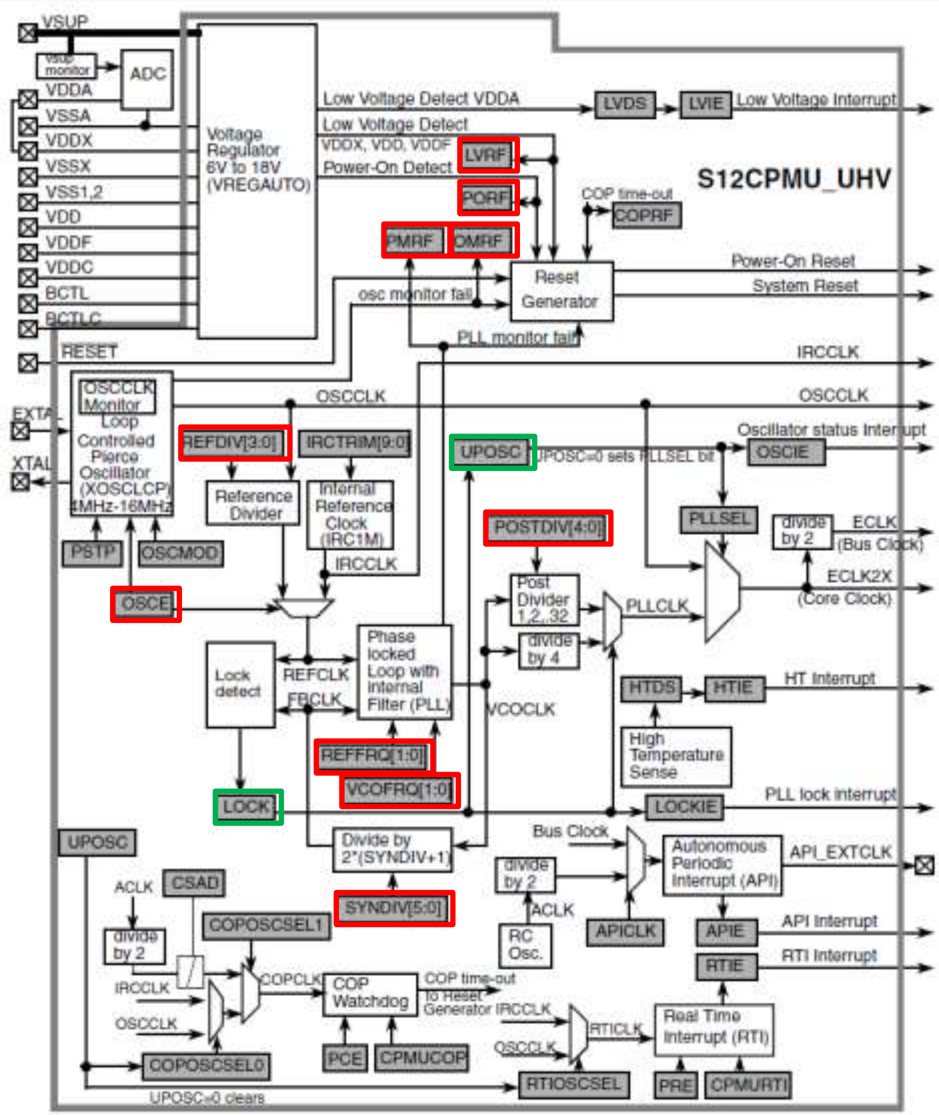
CPMU - Clock, Reset and Power Management

- Supports crystals or resonators from 4 to 16MHz.
- High noise immunity due to input hysteresis and spike filtering.
- Dynamic gain control eliminates the need for external current limiting resistor
- Integrated resistor eliminates the need for external bias resistor
- Low-voltage detect with low-voltage interrupt/reset
- Power-on reset
- On Chip Temperature Sensor and Bandgap Voltage measurement via internal ADC channel
- External ballast device support to reduce internal power dissipation
- Capable of supplying both the MCU internally plus external components
- Over-temperature protection and interrupt
- Internal Reference Clock 1MHz (sufficient for LIN communication in slave mode) with
  - Frequency trimming
  - Temperature Coefficient trimming
- Autonomous periodical interrupt (API)
- COP time-out
- Loss of oscillation/PLL monitor reset



CPMU - Clock, Reset and Power Management

- Configuration S/W routine:
  - Set external 4MHz oscillator as clock source
  - Core clock set to 25 MHz
  - Bus clock set to 12.5 MHz
  - Wait for stable PLL operation
  - Clear fail-monitor flags



Example project – CPMU

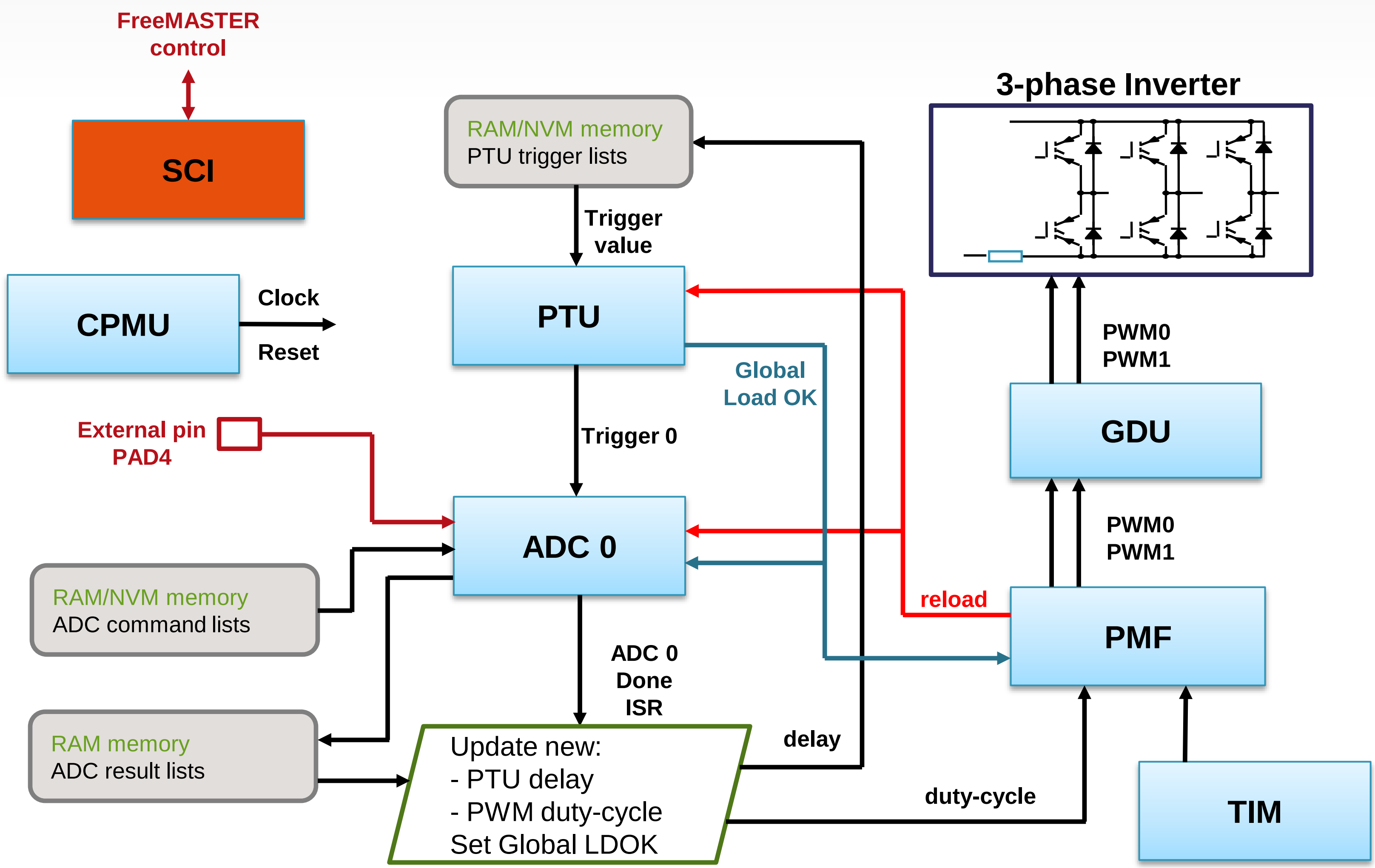
```
void initCPMU(void){

  while (GDUF_GLVLSF)        /* Wait for stable supply after power up */
    GDUF_GLVLSF = 1;

  /* Settings for 25MHz/12.5MHz core/bus clocks, out of 4MHz ext. osc. */
  CPMUREFDIV_REFDIV = 3;    /* fREF = 4MHz / (3 + 1) = 1MHz */
  CPMUREFDIV_REFFRQ = 0;    /* 0 if fREF is from 0 to 1MHz */
  CPMUSYNR_SYNDIV = 24;     /* fVCO = fREF * 2 *(24 + 1) = 50MHz */
  CPMUSYNR_VCOFRQ = 1;      /* 1 if fVCO is from 48 to 80MHz */
  CPMUPOSTDIV_POSTDIV = 1; /* fPLL = fVCO (1 + 1) = 25MHz */

  CPMUOSC_OSCE = 1;         /* Enable external oscillator */
  while (CPMUIFLG_UPOSC == 0) {}; /* Wait for external oscillator */
  while (CPMUIFLG_LOCK == 0) {};  /* Wait for PLL to lock */

  CPMURFLG = 0x60;          /* Clear PORF and LVRF flags */
}
```

PIM – Port Integration Module

- The PIM establishes the interface between the peripheral modules and the device I/O pins

- Routing options:
  - SPI0 to alternative pins
  - SCI1 to alternative pins
  - Various SCI0-LINPHY0 routing options
  - PWM channels to GDU and/or pins
  - TIM0 routing to ACLK, RXD0 or RXD1
  - 3 pin input mux to one TIM0 IC channel (logically XORed)

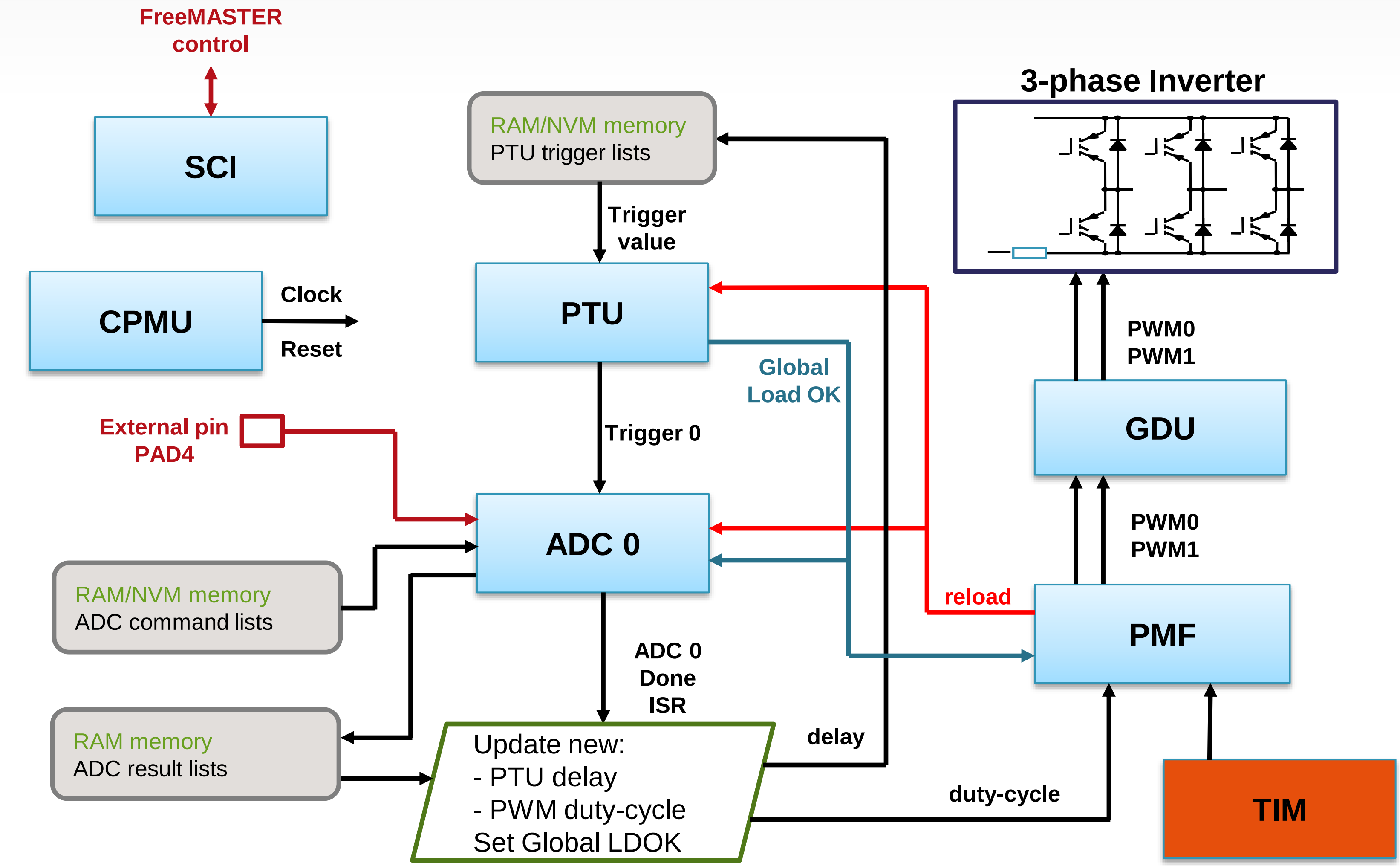
Application Block Diagram – PIM & SCI



Example project – GPIO (PIM) & SCI

```c
void initGPIO(void){

   MODRR0 = MODRR0_CONFIG;    /* Serial Port routing*/
   DDRS = DDRS_CONFIG;        /* User LEDs */
   DDRP = DDRP_CONFIG;        /* EVDD output */

   HALL_SUPPLY = 1;   /* Turn on the hall sensor supply (EVDD) */
   LED1 = 0;          /* Turn off LEDs by default */
   LED2 = 0;          /* Turn off LEDs by default */

   MODRR2 = MODRR2_CONFIG;/* Serial Port routing*/
}



void initSCI(void){

   SCI1BD = SCI_BAUDRATE;  /* Set target baud rate = fbus / SCIBD */
   SCI1CR2_TE = 1;         /* Enable Transmitter */
   SCI1CR2_RE = 1;         /* Enable Receiver */
}
```

Application Block Diagram – TIM



Example project – TIM

```
void initTIM(void)
{

   TIM0TIOS_IOS0 = 1;    /* Channel 0 configured as an output compare */
   TIM0TCTL2 = 0;        /* No action on pin for channel 0 OC event */

   TIM0TIOS_IOS1 = 0;    /* Channel 1 configured as an input capture */
   TIM0TCTL4 = 0xC;      /* Input Capture on both edges of channel 1 */

   TIM0TIE_C0I = 1;      /* Channel 0 interrupt enabled */
   TIM0TIE_C1I = 1;      /* Channel 1 interrupt enabled */

   TIM0TC0 = TIM_CH0_PERIOD;
   TIM0TSCR2_PR = TIM_PRESCALER;

   TIM0TSCR1_TEN = 1;    /* Enable Timer counter */
}
```

freescale™

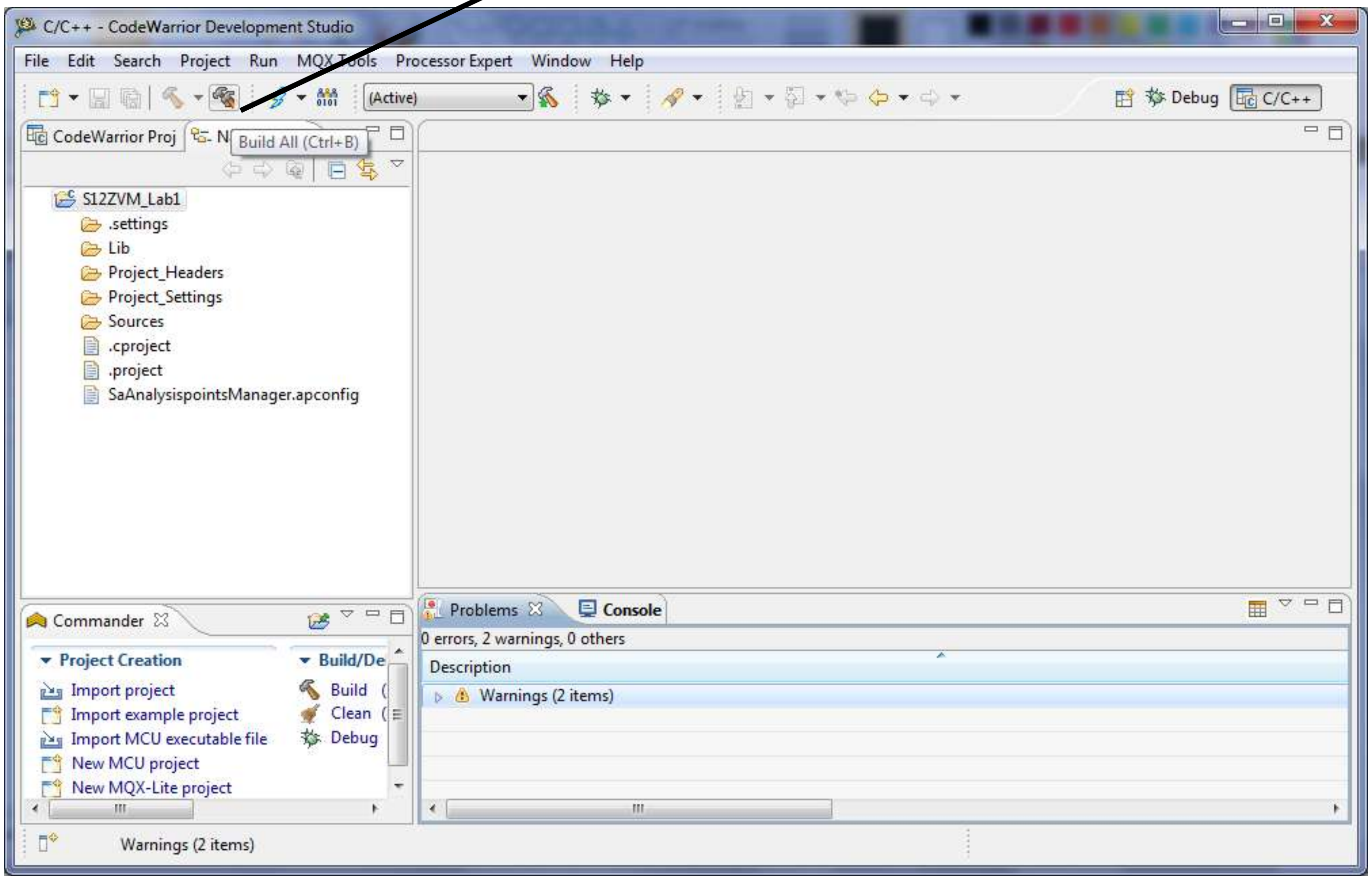
Example project – TIM Interrupt Service Routines

```
interrupt VectorNumber_Vtim0ch0 void TIM_CH0_ISR(void)
{
  LED1 = ~(LED1);        /* Toggle LED 1 */
  TIM0TC0 = TIM0TCNT + TIM_CH0_PERIOD;/* Set new output compare value */
  TIM0TFLG1 = TIM0TFLG1_C0F_MASK;/* Clear the flag */
}
```

```
interrupt VectorNumber_Vtim0ch1 void TIM_CH1_ISR(void)
{
  LED2 = ~(LED2);        /* Toggle LED 2 */
  hall_pattern = 0x07 & (PTIT >> 1);/* Capture Hall sensor pattern */
  TIM0TFLG1 = TIM0TFLG1_C1F_MASK;/* Clear the flag */
}
```

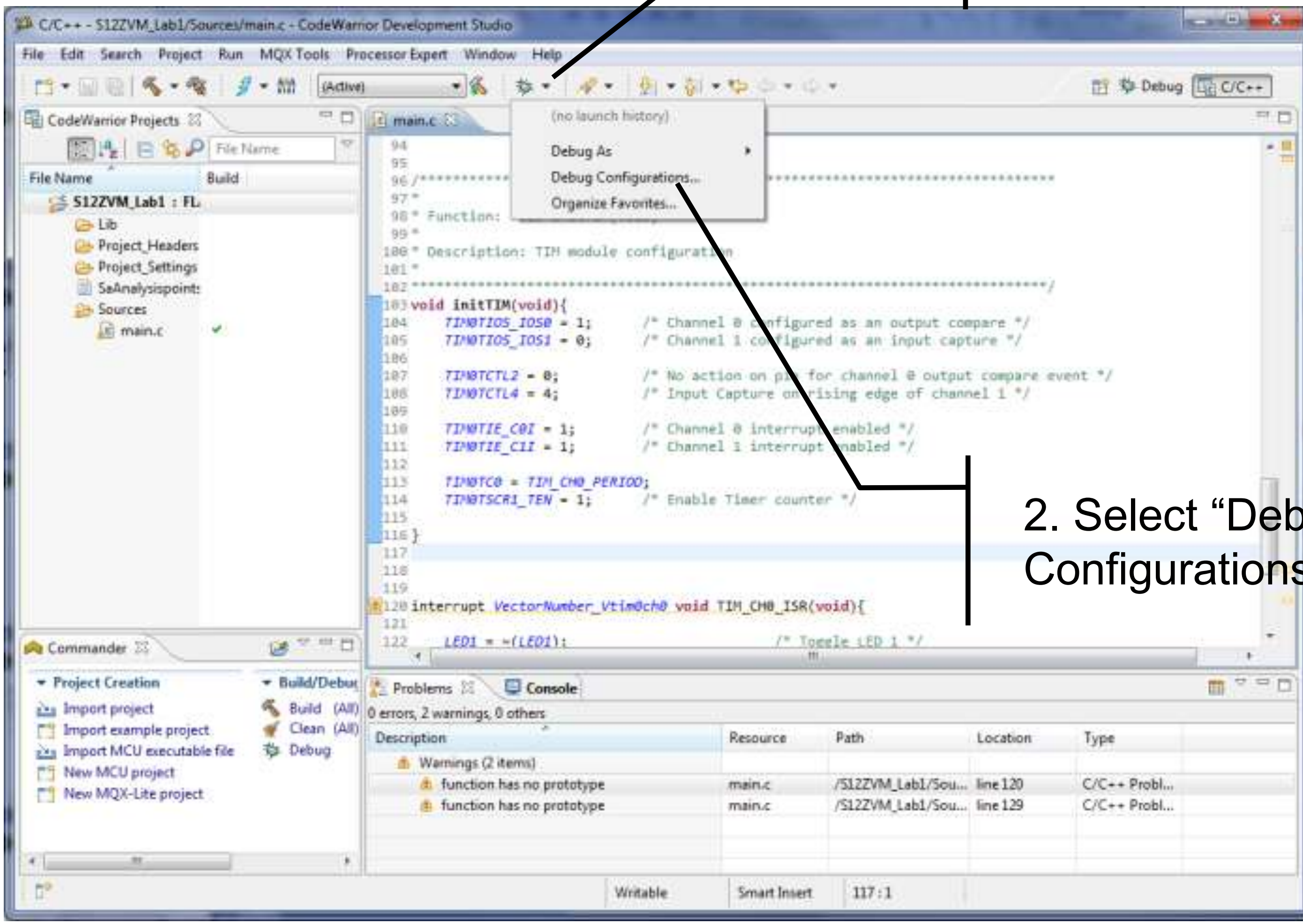
Build the project

1. Click on the “Build All” button



Start the Debugger

1. Click on the drop-down menu at the bug icon

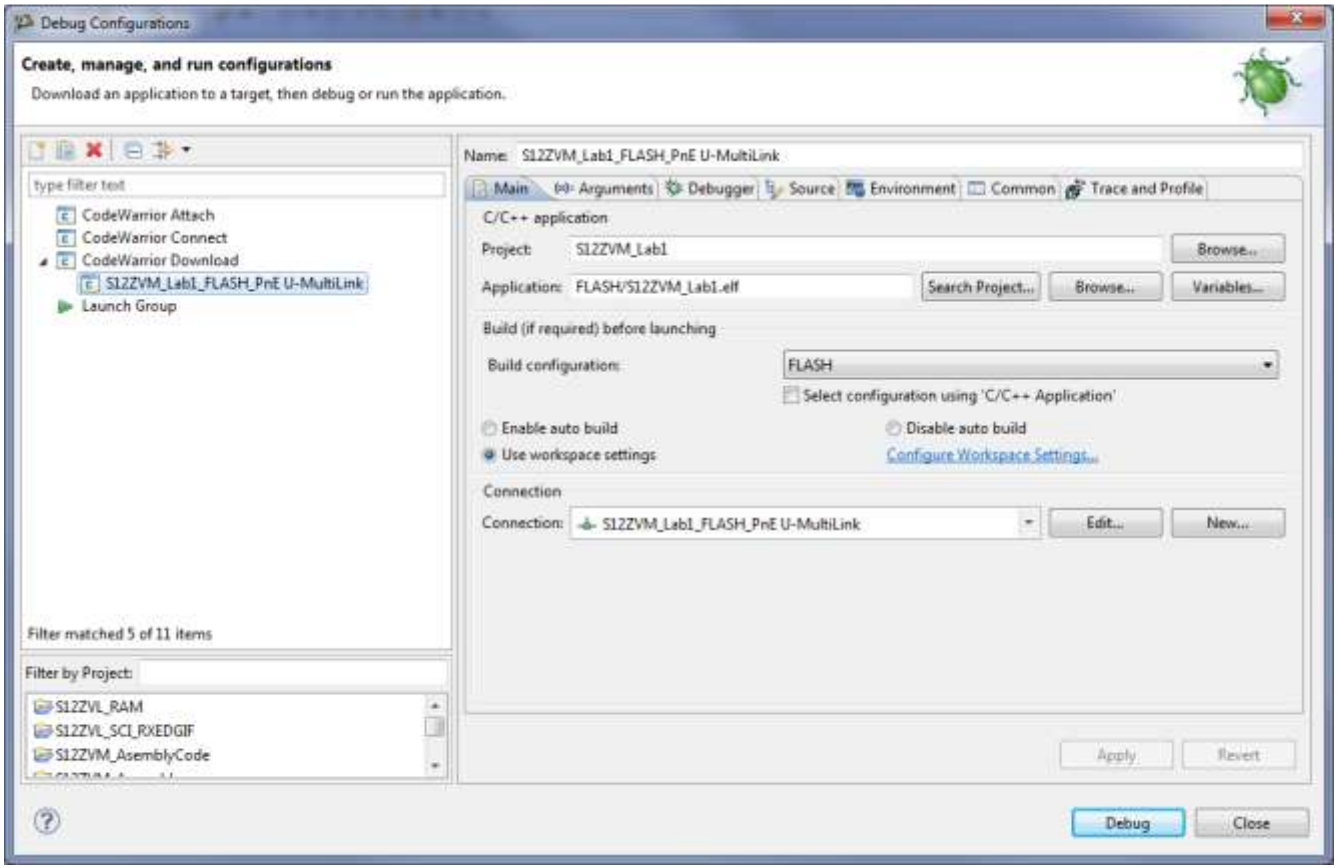


2. Select "Debug Configurations…"

freescale™

Debug Configurations

- The following image shows the settings of the debugger in the “CodeWarrior Download” configuration



Debug Configuration Settings

- In the Main tab:
  - Project: S12ZVM_Lab1
  - Application: FLASH/S12ZVM_Lab1.elf
  - Connection: S12ZVM_Lab1_FLASH_PnE U-MultiLink

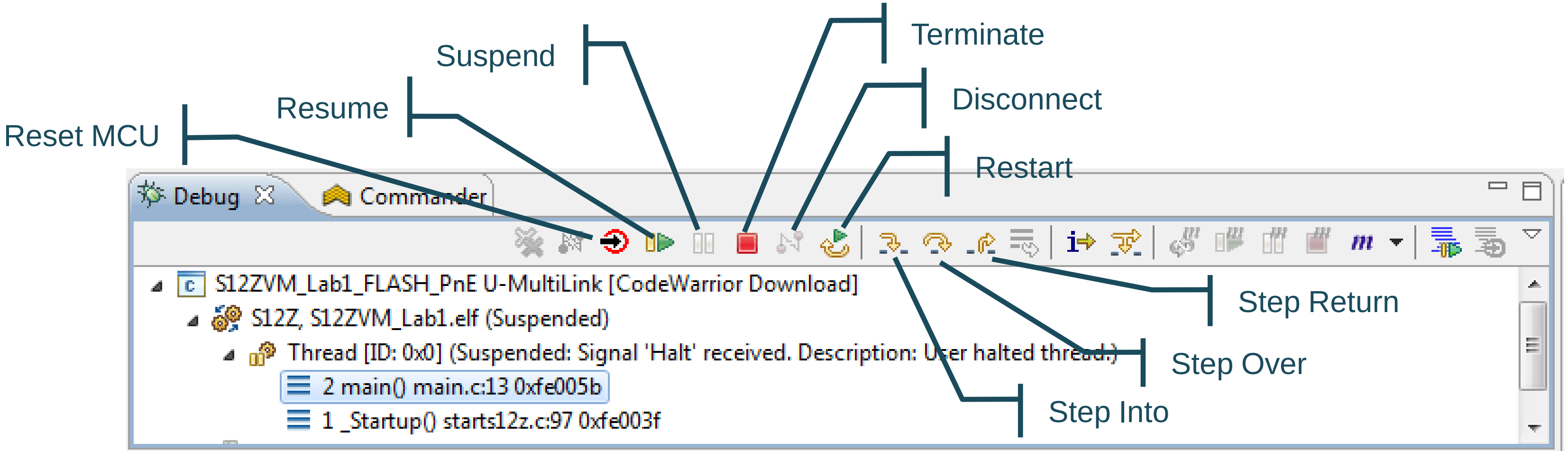
- In the Edit menu of the Connection section:
  - Target: S12ZVM_Lab1_FLASH_PnE U-MultiLink Target
  - Connection type: P&E S12Z Multilink\Multilink Universal\Cyclone Pro\OSBDM
  - The checkbox for “Use Bus Clock as Debug Controller (SIBDC) Clock Source {Default=Checked}” must be un-checked

Starting the debugger

- From the Debug Configurations interface, click on Debug
  - Or, from the workspace, click on the Bug icon.

- CodeWarrior will compile the project and program the device.

- You can start the execution of the code with the provided buttons:



Running the code

- When the code is running, you will see that LED1 is ON and slightly dimmed (1ms ON / 1 ms OFF)
- Rotate the motor manually and see how LED2 toggles at different positions of the rotor
- Stopping the execution at any time will show you the current values of the variables being “watched”

freescale™

What we have learned until now?

- Import existing project into Eclipse IDE workbench
- Setup Eclipse IDE to compile the source code and program it to a target
- Connect with debugging interface to target, and run the programmed device

Using FreeMASTER Interface

Hands-on session # 2

Hands-on session #2 – Objectives

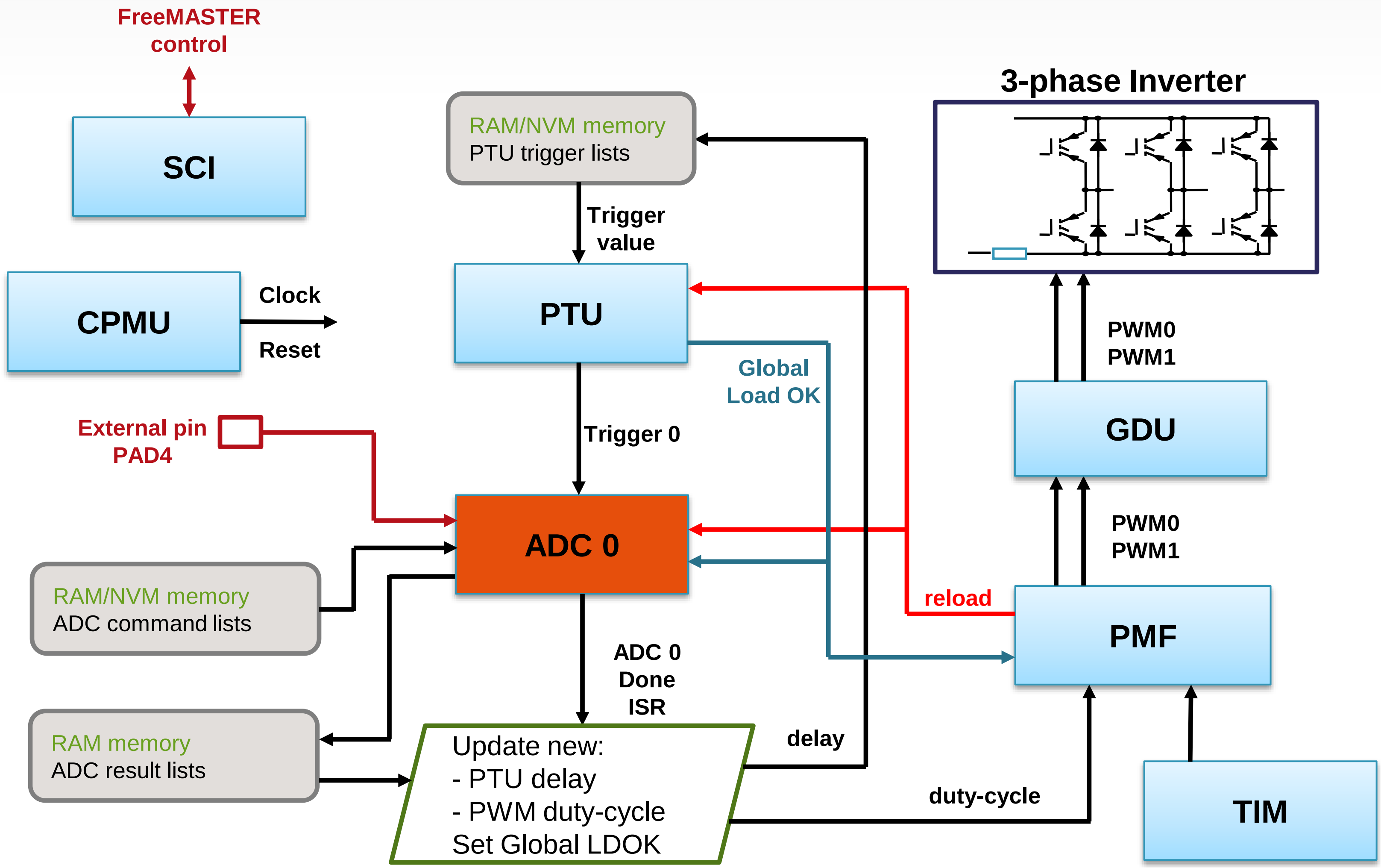
- In this session you will:
  - Add a routine to initialize the ADC and read the potentiometer value
  - Import the FreeMaster Serial Communication Driver into a CW project
  - Start a FreeMaster project and visualize the variable data as text
  - Add a scope view in freemaster to visualize the variable data

freescale™

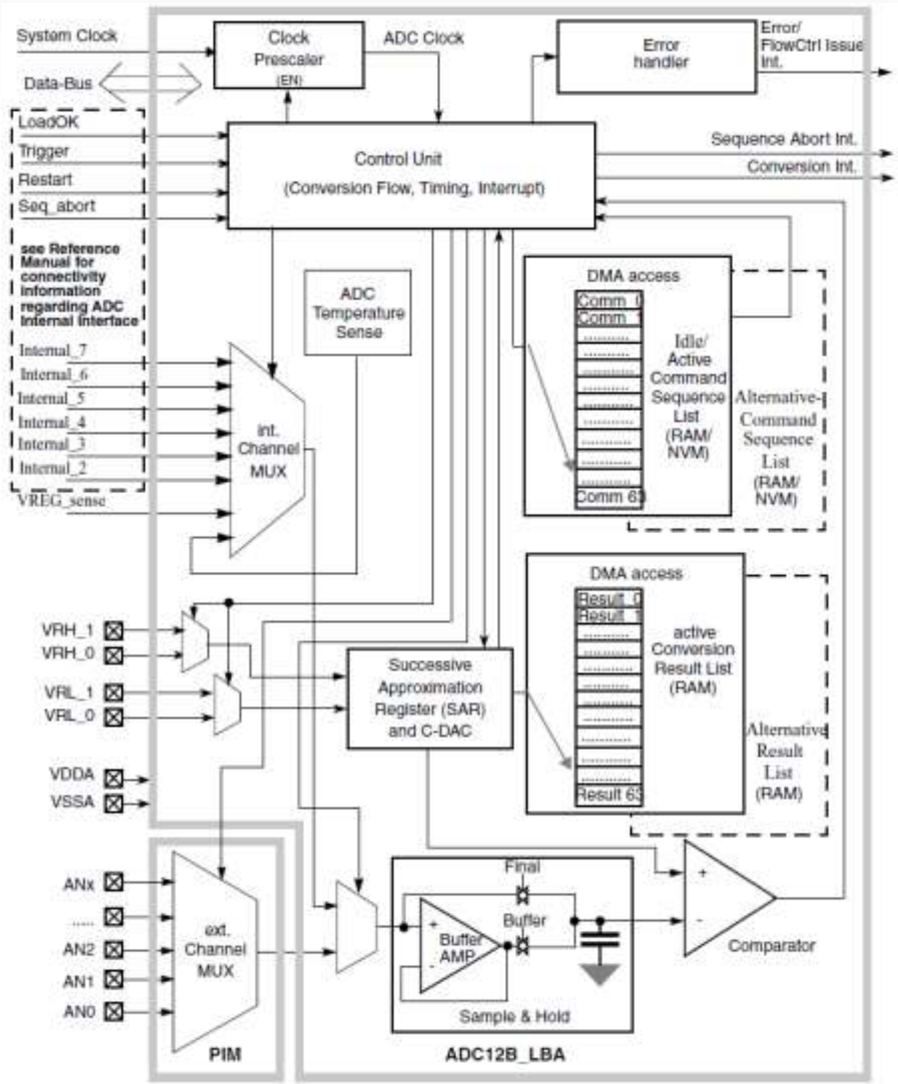
Lab session #2 – Details

- Import the 2nd lab session “S12ZVM_Lab2”
- This project has two additions:
  - FreeMaster serial interface driver that “reports” the values of the project variables to FreeMaster, using the serial port SCI1
  - The ADC is configured to cyclically read the potentiometer value

Application Block Diagram – ADC

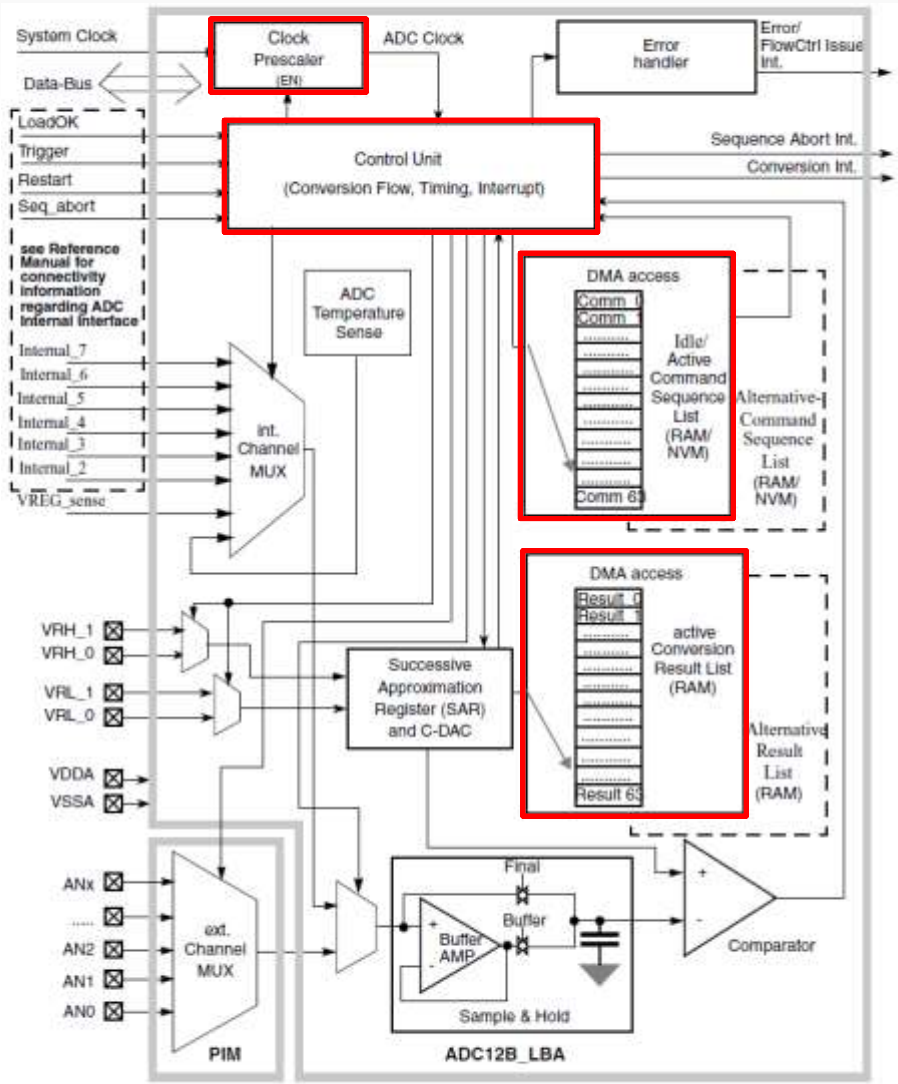


ADC – Analog-to-Digital Converter



- Programmers model with List Based Architecture for conversion command and result value organization
- 8-, 10-, or 12-bit selectable resolution
- Channel select control for n external analog input channels
- Eight additional internal channels
- Programmable sample time
- Providing a sample buffer amplifier for channel sampling (improved performance in view to influence of channel input path resistance versus conversion accuracy)
- Left/right justified result data
- Individual selectable VRH_0/1 and VRL_0/1 Ref. inputs
- Special conversions for selected VRH_0/1, VRL_0/1, (VRL_0/1 + VRH_0/1) / 2
- Providing 15 conversion interrupts with flexible interrupt organization per conversion result
- One dedicated interrupt for “End Of List” type commands
- Provides conversion sequence abort
- The Command Sequence List and Result Value List are implemented in double buffered manner (two lists in parallel for each function)
- Conversion Command loading possible from System RAM or NVM

ADC – Analog-to-Digital Converter



- Configuration S/W routine:
  - Set ADC clock to 6.25 MHz
  - Set output format to 12-bit resolution, right justified data
  - Set ADC in “Trigger” mode
  - Set pointer to ADC Command Sequence Lists
  - Set pointer to ADC Result Value Lists
  - Enable End of list ISR
  - Enable ADC error ISRs

ADC Configuration

- The following global variables have been defined to store the Command List and Results List:

```
volatile char ADC0CommandList[4][4] @0x001000 = {
   {0xC1,0xD4,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00}
};

volatile unsigned short ADC0ResultList[4] @0x001020 = {0, 0, 0, 0};
```

- In this case, there is a single list with a single command for ADC0. The command specifies:

{ 0xC1, 0xD4, 0x00, 0x00 }

Reserved

Sample time (0 = 4 ADC clock cycles)

Channel and reference Voltage select (D4 = AN4, VRH_1/VRL_1)

Conversion command & Interrupt select (C0 = End of list, interrupt enable for converted channel)

ADC Initialization

```
void initADC(void){

  ADC0CTL_0_ACC_CFG = 3;  /* Dual access mode */
  ADC0CTL_0_MOD_CFG = 1;  /* Trigger mode */
  ADC0TIM = 0;    /* ADC clock = fbus / (2x(ADC0TIM + 1)) [0.25 - 8MHz] */

  ADC0FMT_DJM = 1;        /* Right justified result data */
  ADC0FMT_SRES = 4;       /* 12-bit resolution */

  /* ADC0 Command & Result Base Pointers */
  ADC0CBP = ADC0CommandList;
  ADC0RBP = ADC0ResultList;
  ADC0CROFF1 = 0;

  ADC0CONIE_1_EOL_IE = 1; /* Enable End-of-list interrupt */
  ADC0EIE = 0xEE;         /* Enable all errors interrupts */
  ADC0CTL_0_ADC_EN = 1;   /* Enable ADC0 */

  ADC0FLWCTL_RSTA = 1;    /* Issue a restart event */
  while (1 == ADC0FLWCTL_RSTA) ;  /* Wait until restart flag is cleared */
}
```

Adding instructions to trigger ADC

- In this project we will trigger the ADC by software, using Trigger Mode: conversion flow is controlled only by triggers; restart is automatic when EOL conversion is complete.

- There is a new global variable to store the potentiometer value

```
unsigned short pot_value = 0;
```

- The following lines in the infinite for loop will execute a Trigger every time the first conversion flag (defined in the ADC command) is clear:

```
if (1 == ADC0CONIF_CON_IF){
  ADC0CONIF_CON_IF = 1;/* Clear flag */
  ADC0FLWCTL_TRIG = 1;/* Trigger next conversion */
}
```

ADC Interrupt Service Routine

- The potentiometer value is obtained in the ADC conversion complete interrupt routine

```
interrupt VectorNumber_Vadc0conv_compl void ADC0done_ISR(void)
{

  pot_value = ADC0ResultList[0];   /* Update Adc Result */
  ADC0CONIF = 1;                   /* Clear ISR flag */

}
```

FreeMaster Communication Driver

- Go to www.freescale.com/freemaster
  - Go to the “downloads” tab and look for “FreeMASTER Communication Driver”
  - In the CodeWarrior project window, paste the FreeMASTER folder into the “Project_Headers” folder of your project
  - Once the package is installed, there are several options to interface with the target device, using CAN, SCI, or JTAG

For additional information, refer to Freescale’s Application Note AN4752

Adding the FreeMaster Communication Driver

- The corresponding header and C files from the unpacked folder are added to the project header files.

- The paths containing the FreeMaster files must be incorporated into the project:
  - From the CW menu bar, go to Project > Properties
  - Go to “C/C++ Build”> “Settings”
  - Look for the item “Access Paths” under S12Z Compiler
  - Add the following paths under: ”Search User Paths”:

    "${ProjDirPath}\Project_Headers\FreeMASTER“

    "${ProjDirPath}\Project_Headers\FreeMASTER\S12ZVM“

    "${ProjDirPath}\Project_Headers\FreeMASTER\src_common"

freescale™

Using the FreeMaster Serial Driver

- At the top of your project, we have included the freemaster header file:

```
#include "freemaster.h"
```

- The "main" routine now includes a FreeMaster initialization (must be always after the comms initialization; in this case, the SCI):

```
FMSTR_Init();
```

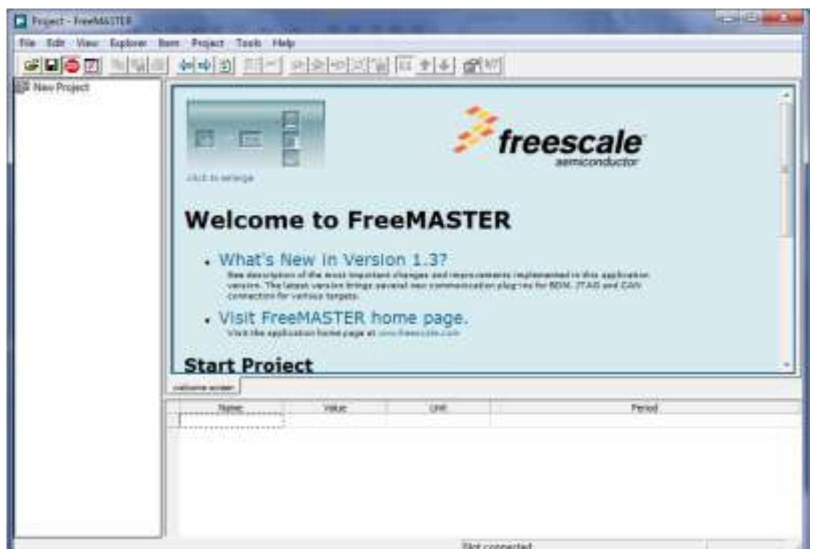
- The infinite for loop now includes a function that continuously sends the variable values to FreeMaster

```
FMSTR_Poll();
```

Start FreeMASTER Interface

- From the Start Menu in Windows, go to
  - Start > All Programs > FreeMaster 1.3

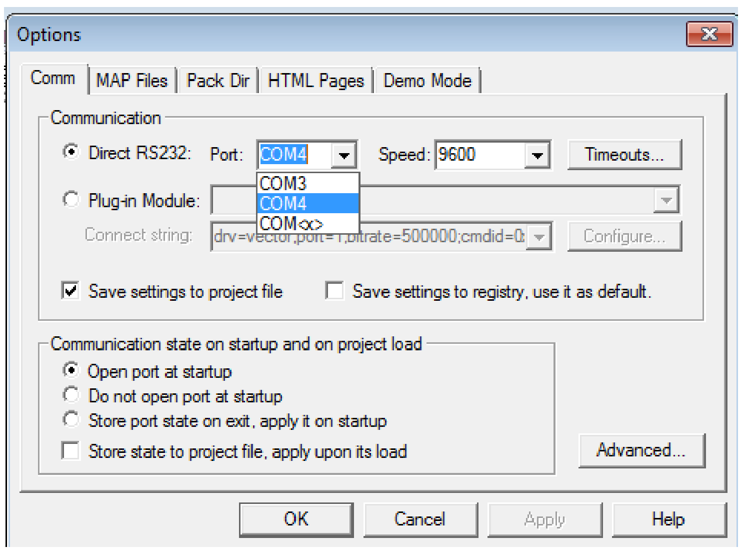
- The FreeMASTER tool will start
  - ignore all the warnings and error messages, they are most probably caused by incorrectly assigned serial port)



FreeMaster – Configuring the Serial Port

- On the menu bar, go to Project > Options

- Select the correct COMM port, with a speed setting of 9600 (this is the value we used in the SCI initialization)

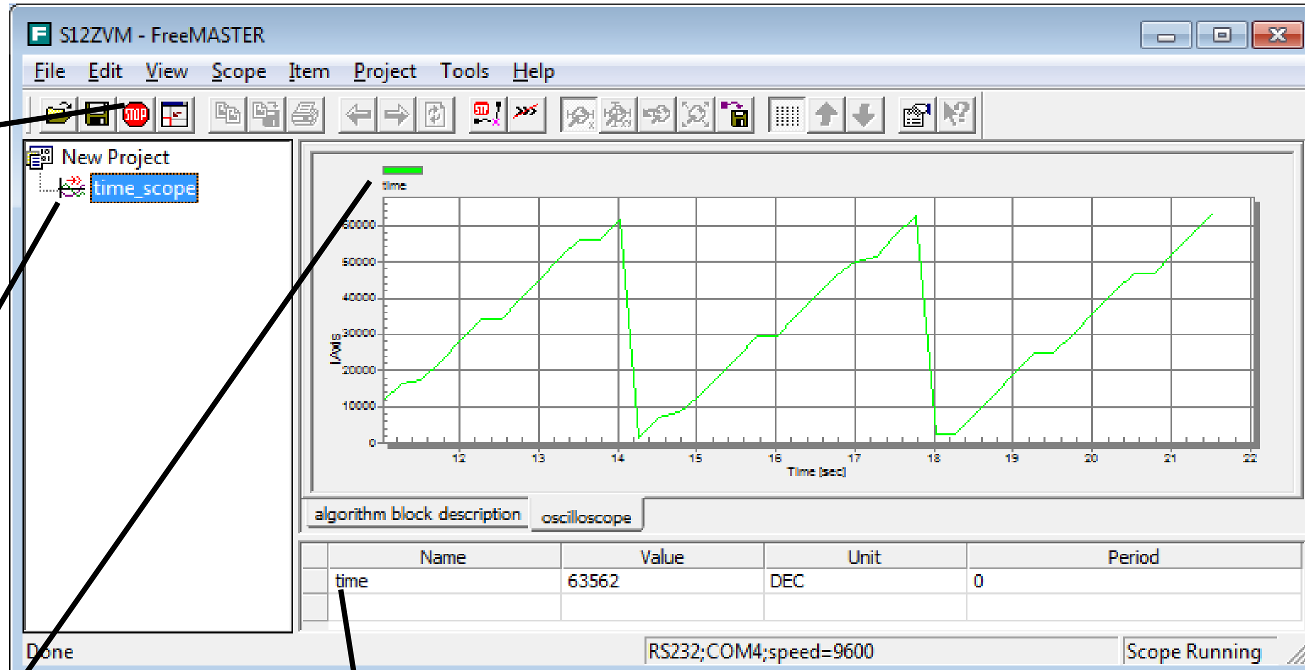


FreeMaster – Loading the MAP file

- From the options window, go to tab “MAP Files”
- Select the default symbol file:
  - Click on “…” and browse to the location where the ELF file is stored (C:\BLDC_workshop\ S12ZVM_Lab2\FLASH\)
  - Select the file “S12ZVM_Lab2.elf”
- Select the file format:
  - Binary ELF with DWARF1 or DWARF2 dbg format
- Click OK

FreeMASTER Interface

- In the FreeMASTER interface for “Empty Project” variable time is watched. This variable is also added to scope interface in order to be monitored in graphical representation.



Start/Stop serial communication with target

Variable “time” in watch window

Visualization of variable “time” in scope

Scope selector

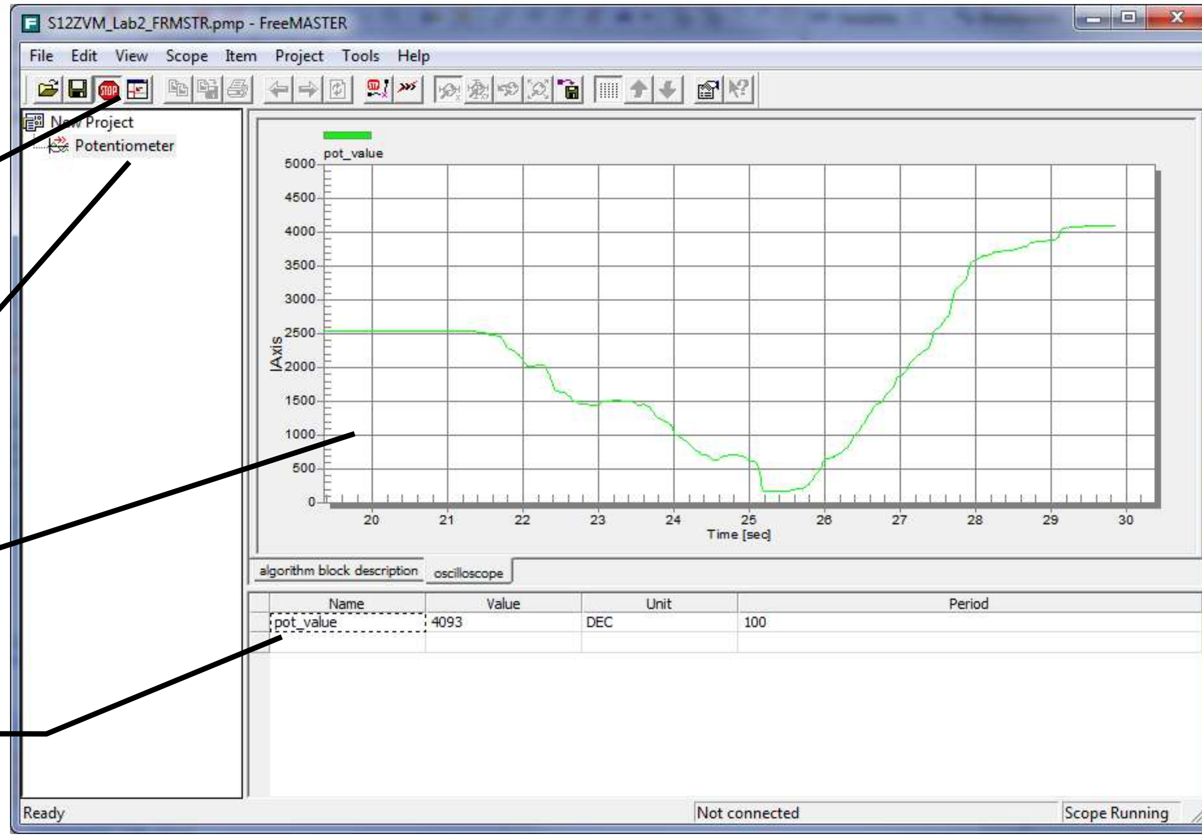
FreeMASTER Interface

Start/Stop serial communication with target

Scope selector

Visualization of variable in scope

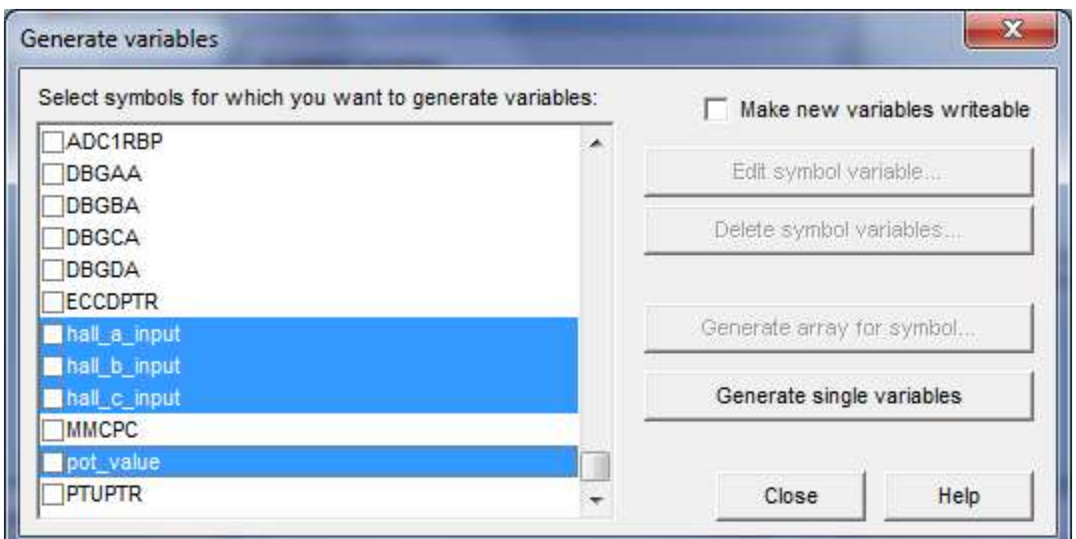
Variable in watch window

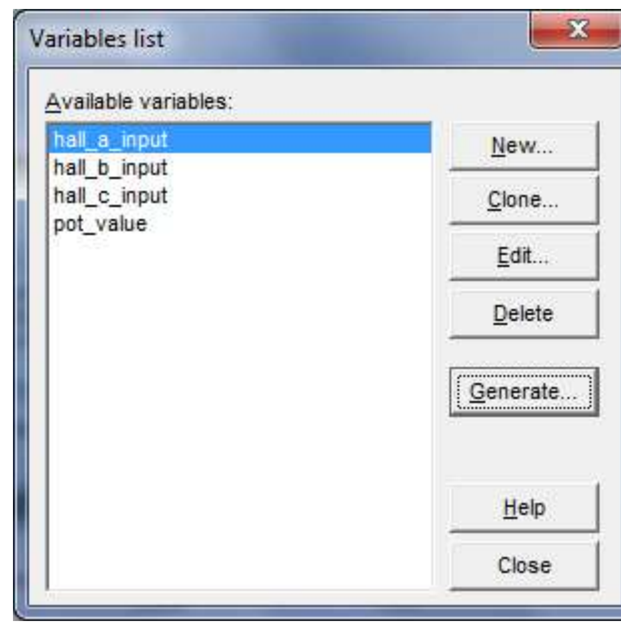


freescale™

Adding Variables

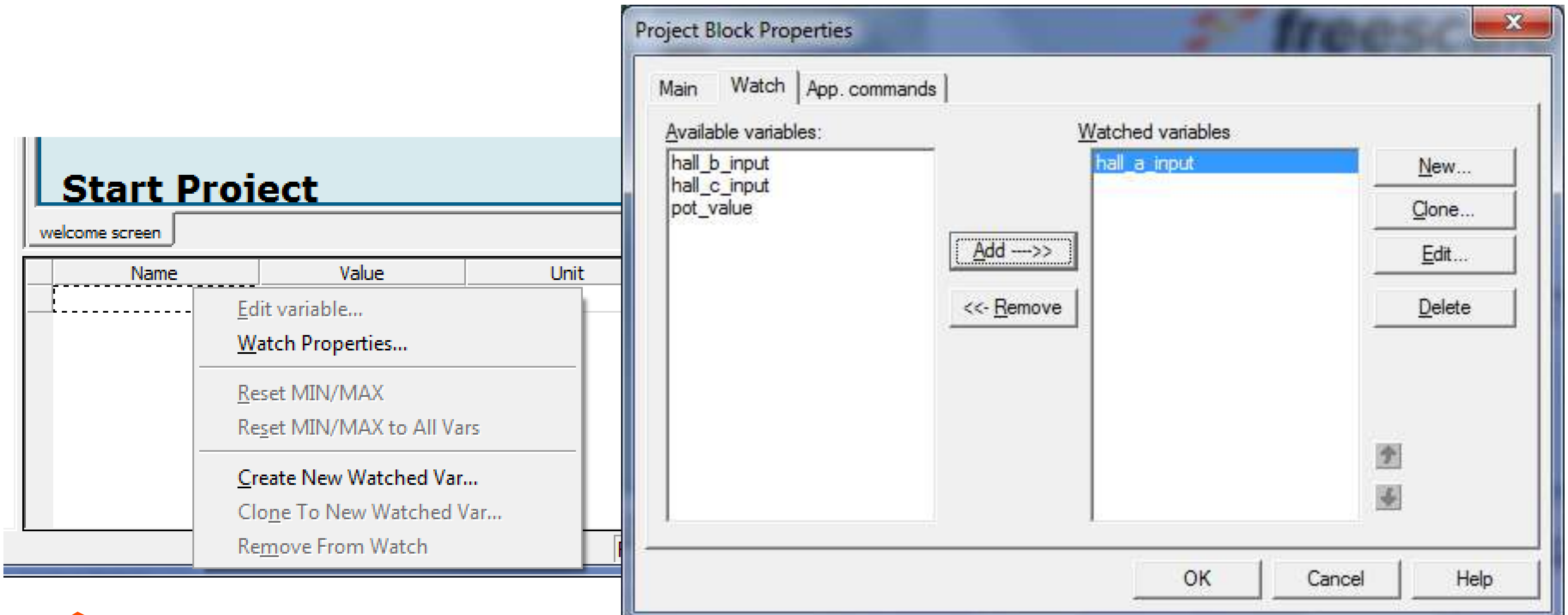
- On the menu bar, go to Project > Variables
- When the window appears, select “Generate”
- Scroll down the list of variables and find the global variables that will be monitored
- Click on “Generate single variables”, then “Close”
- Close the “Variables list” window





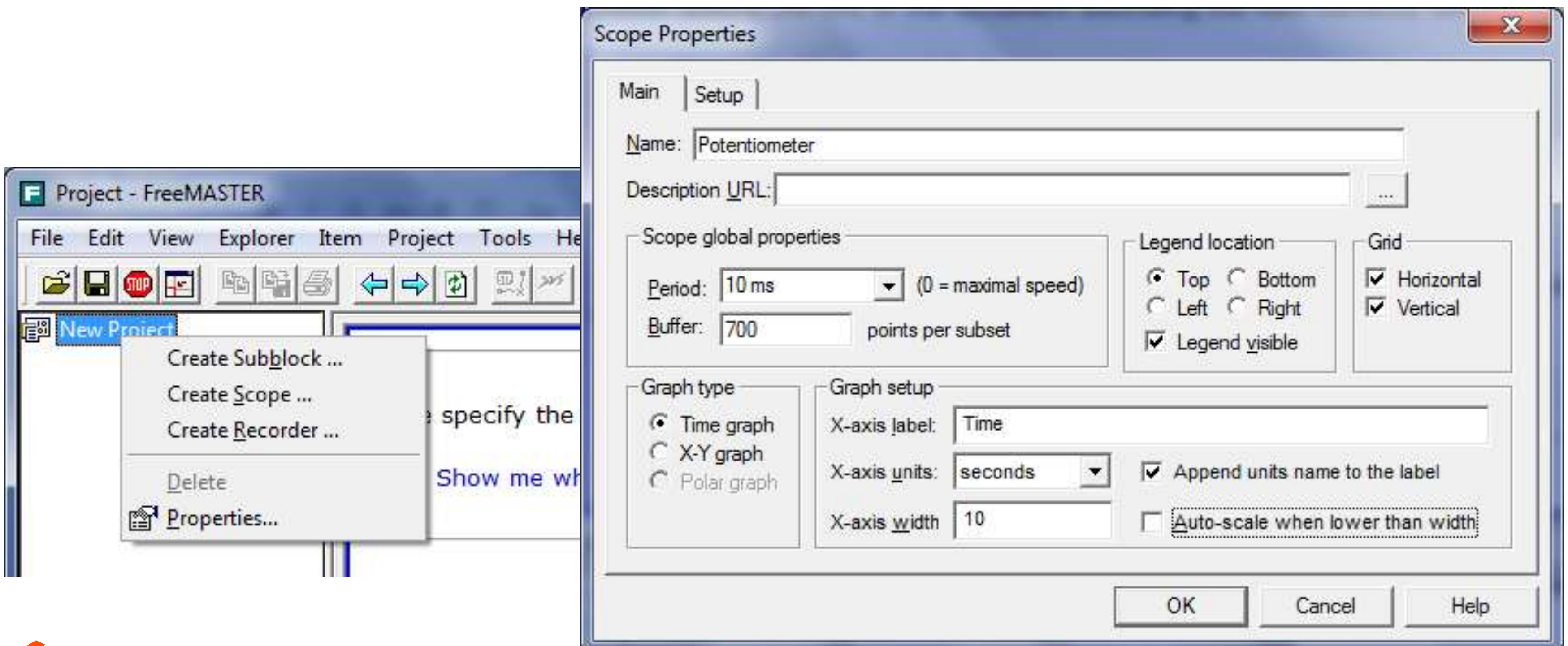
Adding variables to the Watch List

- Right click into “watch” area and select “Watch Properties”
- Switch to tab “Watch” in Project Block Properties
- Select the variables to watch and click on “Add”



Adding a Scope

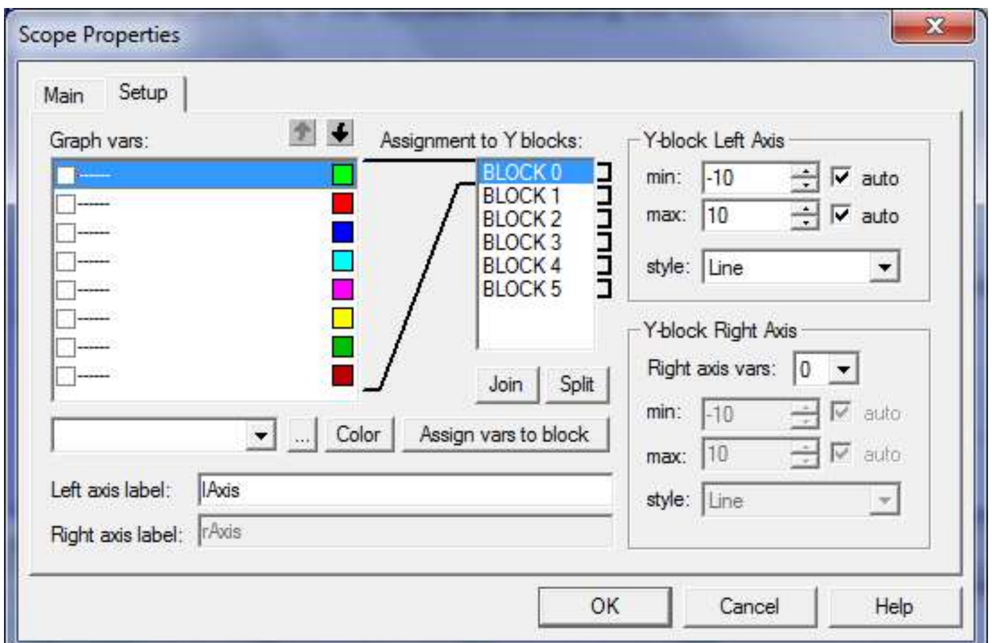
- Right-click on New Project and select the option “Create Scope”
- Define a name for the scope
- Change Period to 10ms, and Buffer to 700 points per subset



freescale™

Setup a variable in the scope

1. Select the first unassigned variable slot
2. Select the variable pot_value from the dropdown list
3. With BLOCK 0 selected, click on “Assign vars to block”
4. Set the Y-block left axis min value to 0, max value to 5000.



freescale™

Running the code

- When the code is running, you will see that LED1 is ON and slightly dimmed (1ms ON / 1 ms OFF)
- Rotate the motor manually and see how LED2 toggles at different positions of the rotor
- The value of each hall sensor line can be visualized in the watch list or in the scope view in real time, without interrupting the code execution
- The ADC converted value from the potentiometer is available in another scope view

What we have learned until now?

- How to initialize the ADC
- How to add the FreeMaster serial driver to a project
- How to setup serial interface in FreeMaster
- Adding variables to the “watch” window in FreeMaster
- Adding variables to a scope view in FreeMaster

Math and Motor Control Lib

Hands-on session # 3

Hands-on session #3 – Objectives

- In this session you will:
  - Incorporate the Math and Motor Control Library into a CW project
  - Add a function to generate a sinusoidal waveform

Setting up the Math and Motor Control Library

- From the CW menu bar, go to Project > Properties
  - Go to “C/C++ General”> “Paths and Symbols”
  - In tab “Includes” click “Add…” then look for the following path in the file system

    “C:\Freescale\AMMCLIB\MC9S12ZVM_AMMCLIB_v1.0.0\include”

  - Switch to tab “Libraries”
    - Add the following paths under: ”Search User Paths”:

      "C:\Freescale\AMMCLIB\MC9S12ZVM_AMMCLIB_v1.0.0\lib\cw10x\MC9S12ZVM_AMMCLIB_v1.0.0.UC.a “

Setting up the Math and Motor Control Library

– Go to “C/C++ Build”> “Settings”

– Look for the item “Access Paths” under S12Z Compiler

- Add the following paths under: ”Search User Paths”:

  "C:\Freescale\AMMCLIB\MC9S12ZVM_AMMCLIB_v1.0.0\include“

  "C:\Freescale\AMMCLIB\MC9S12ZVM_AMMCLIB_v1.0.0\lib\cw10x”

– Look for the item “General” under S12Z Compiler

- Add the following text in the “Other Flags” field:

“-DMCLIB_DEFAULT_IMPLEMENTATION=MCLIB_DEFAULT_IMPLEMENTATION_F16”

freescale™

Adding variables for a sine wave generator

- At the top of your project, include the following libraries:

```
#include "mlib.h"
#include "gflib.h"
#include "SWLIBS_Config.h"
```

- In the global variables section, add the following lines:

```
volatile tFrac16 angle1 = 0;
volatile tFrac16 ampl1 = 0;
volatile tFrac16 sin1 = 0;
```

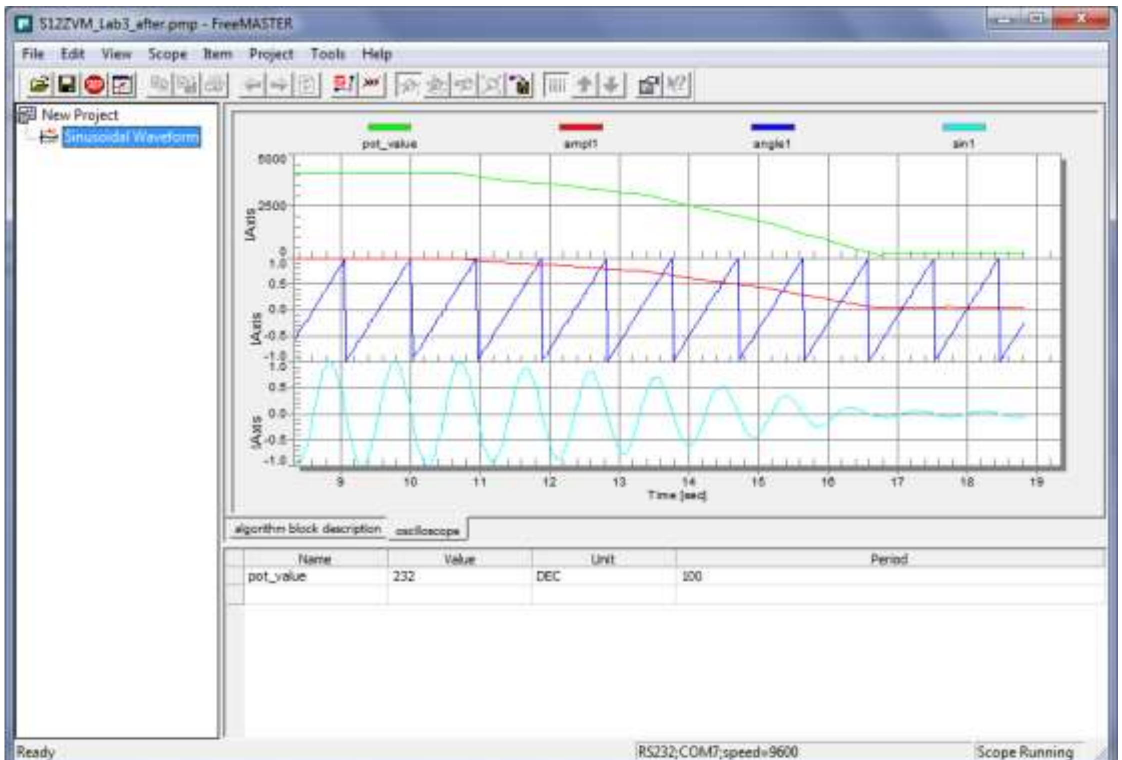
- In the Timer 0 Channel 0 ISR, add the following lines:

```
angle1 += 70;
ampl1 = (tFrac16) (8* pot_value);
sin1 = MLIB_Mul(GFLIB_Sin(angle1), ampl1);
```

freescale™

Visualizing the sine wave in FreeMaster

- Run the FreeMaster application
  - Move the potentiometer and see how the amplitude of the sine wave is affected.



What we have learned until now?

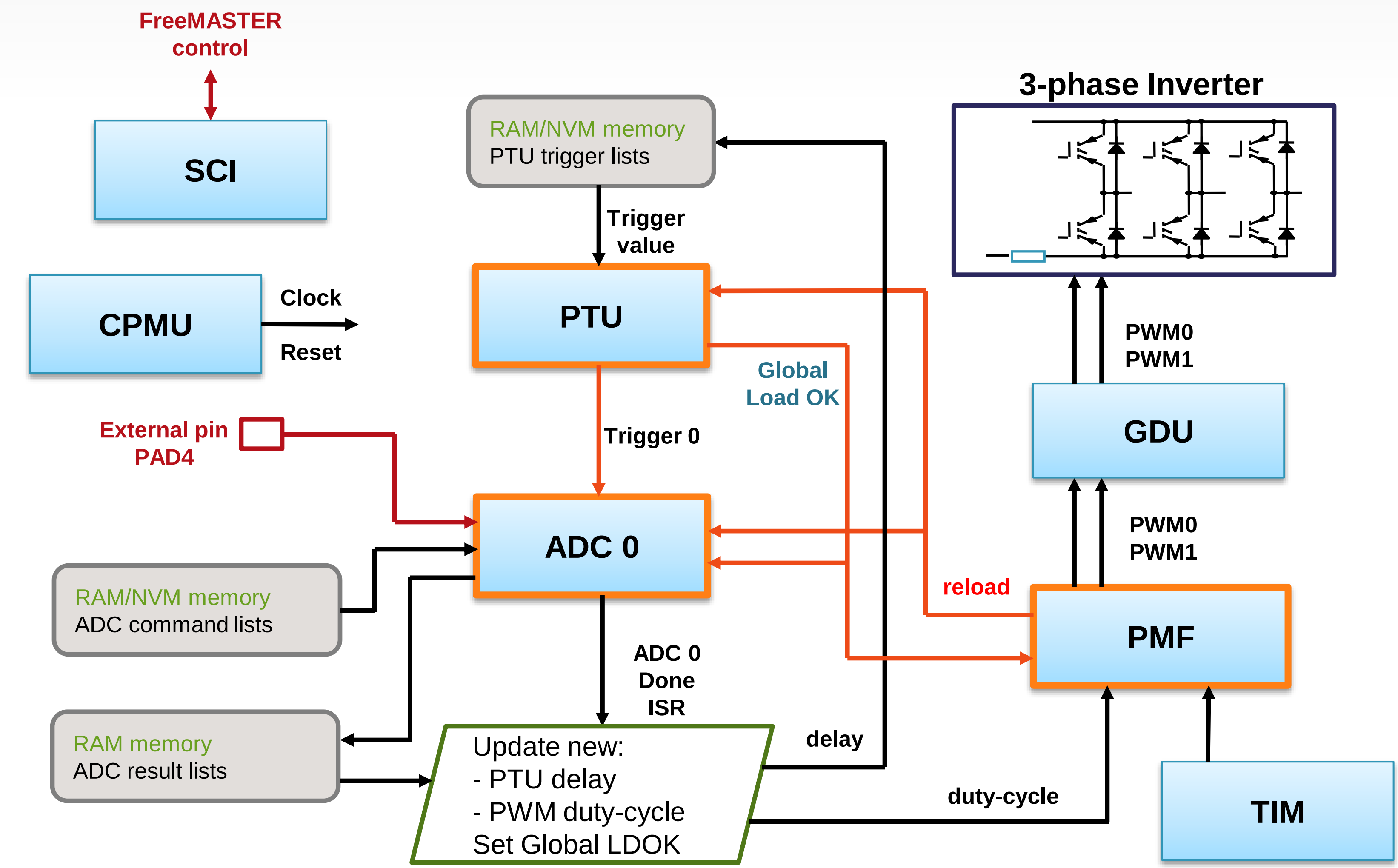
- Adding the Math and Motor Control Library into a project in CodeWarrior.

- Calling a sine function from the math library

Motor Control Loop

Events

Application Block Diagram



Control Loop Events

- Global Load OK
  - Issued by the PTU to communicate to the ADC and PMF that the buffered values must be in effect.
    - PMF: Load new modulus, prescaler, and pulse width settings at the next reload event
    - ADC: The alternative list is ready to swap upon a PMF reload

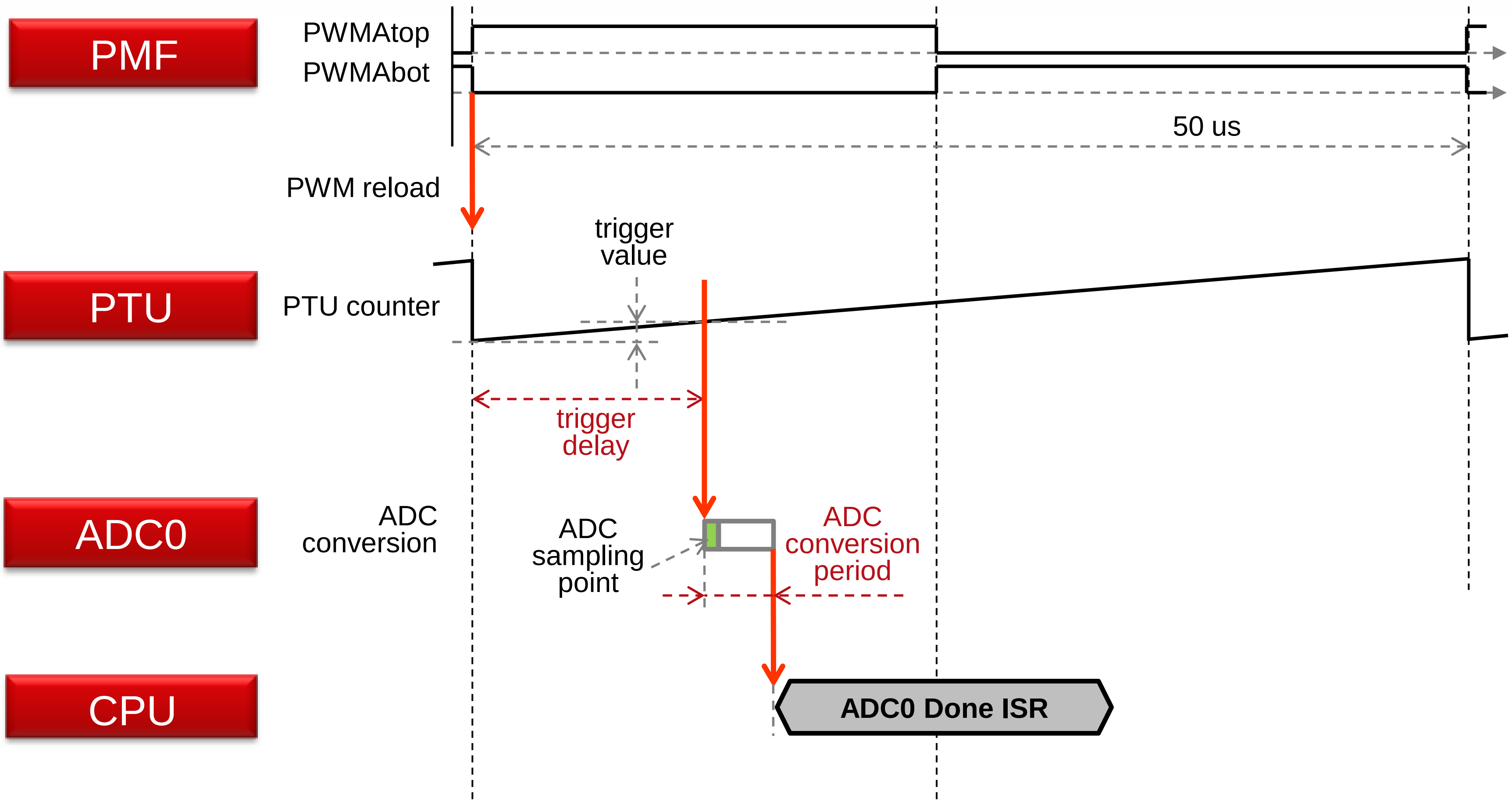
- Reload (from phase A)
  - Issued by the PMF module to communicate that a new PWM cycle is starting
    - PTU: The counter resets and restarts; the first trigger value is loaded
    - ADC: The conversion command from the top of the list is loaded

Control Loop Events (2)

- Commutation Event
  - Started by a TIM channel 0 Output Compare event to inform the PMF of a need for an asynchronous reload
    - PMF: If the commutation event is enabled, the buffered values of the PMFOUTC, PMFOUT, and MSKx bits take effect. The PMF generates the reload signal and a “reload_is_async_signal” to the PTU and ADC
    - PTU: Same as in a synchronous reload event
    - ADC: A conversion abort request is started

- ADC Conversion Trigger
  - Each trigger generator on the PTU issues a trigger to its corresponding ADC to start a new conversion

Application Timing Diagram



freescale™

PMF / PTU / ADC Sync

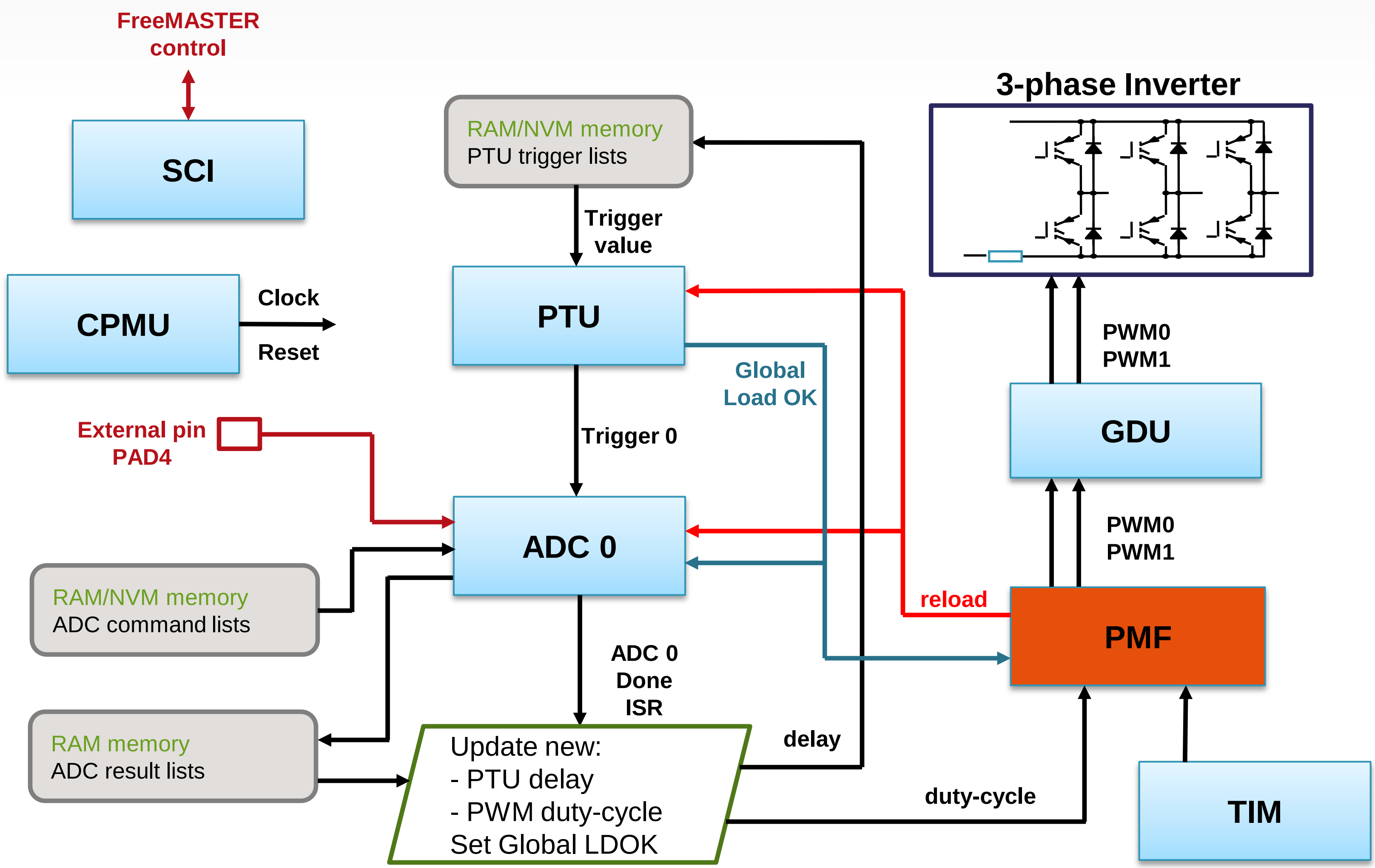
Hands-on session # 4

Hands-on session #4 – Objectives

- In this session you will:
  - Generate edge-aligned 20KHz PWM signal on phase A
  - Enable the PTU to synchronize ADC conversion trigger with PWM
  - Measure the potentiometer and phase A voltages
  - Adjust the ADC sampling point with the potentiometer

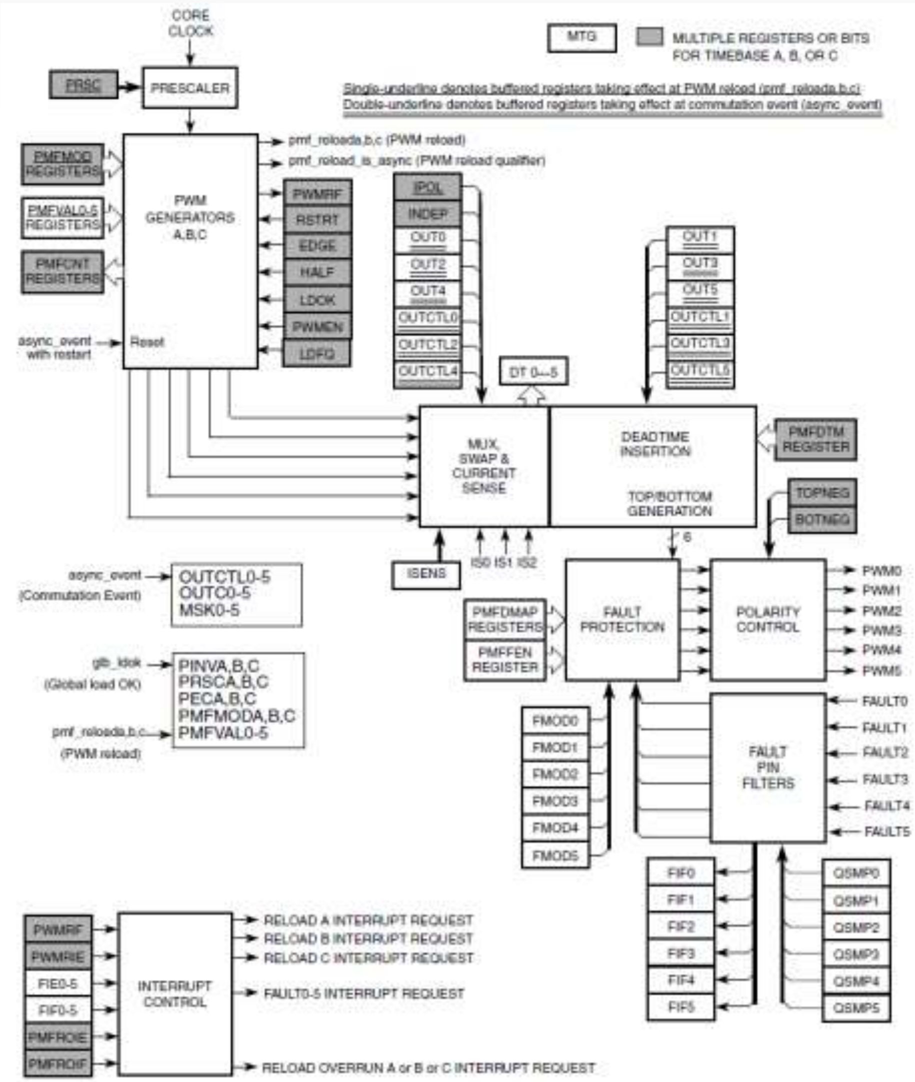
freescale™

Application Block Diagram – PMF



freescale™

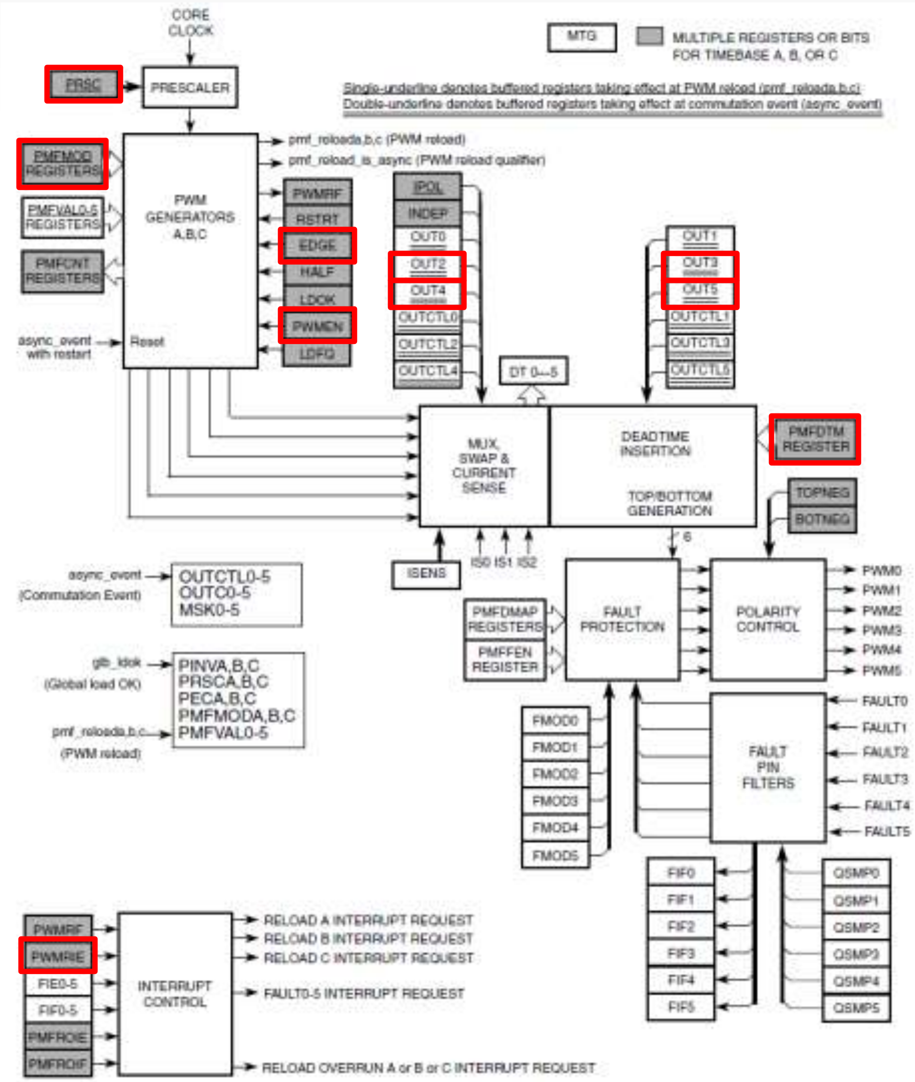
PMF – Pulse Width Modulator



- Three complementary PWM signal pairs, or six independent PWM signals
- Three 15-bit counters based on core clock
- Features of complementary channel operation:
  - Deadtime insertion
  - Separate top and bottom pulse width correction via current status inputs or software
  - Asymmetric PWM output in center-aligned mode (phase shift)
  - Double switching
  - Separate top and bottom polarity control
- Edge-aligned or center-aligned PWM signals
- Half-cycle reload capability
- Integral reload rates from 1 to 16
- Link to timer output compare input for 6-step BLDC commutation support with optional counter restart
- Reload overrun interrupt
- Individual software-controlled PWM output
- Programmable fault protection

freescale™

PMF – Pulse Width Modulator



- Configuration S/W routine:
  - Set PMF clock to core clock (25MHz)
  - Set PWM A to edge-aligned PWM
  - Set output PWM frequency to 20 kHz
  - Set dead time to 0.5 us
  - Disable PWM outputs 2 to 5 (phase B and C)
  - Enable PWM reload event/ISR

PMF – Pulse Width Modulator

```c
void initPMF(void){
  PMFCFG0_EDGEA = 1;

  PMFCFG2_REV0 = 1;
  PMFCFG2_REV1 = 0;

  PMFFQCA = 0;
  PMFMODA = PWM_MODULO;
  PMFDTMA = 10;

  PMFCFG2 |= 0x3c;
  PMFENCA_LDOKA = 1;

  PMFENCA_PWMENA = 1;
  PMFENCA_PWMRIEA = 1;
  PMFENCA_GLDOKA = 1;
}
```

Edge-aligned PWM mode

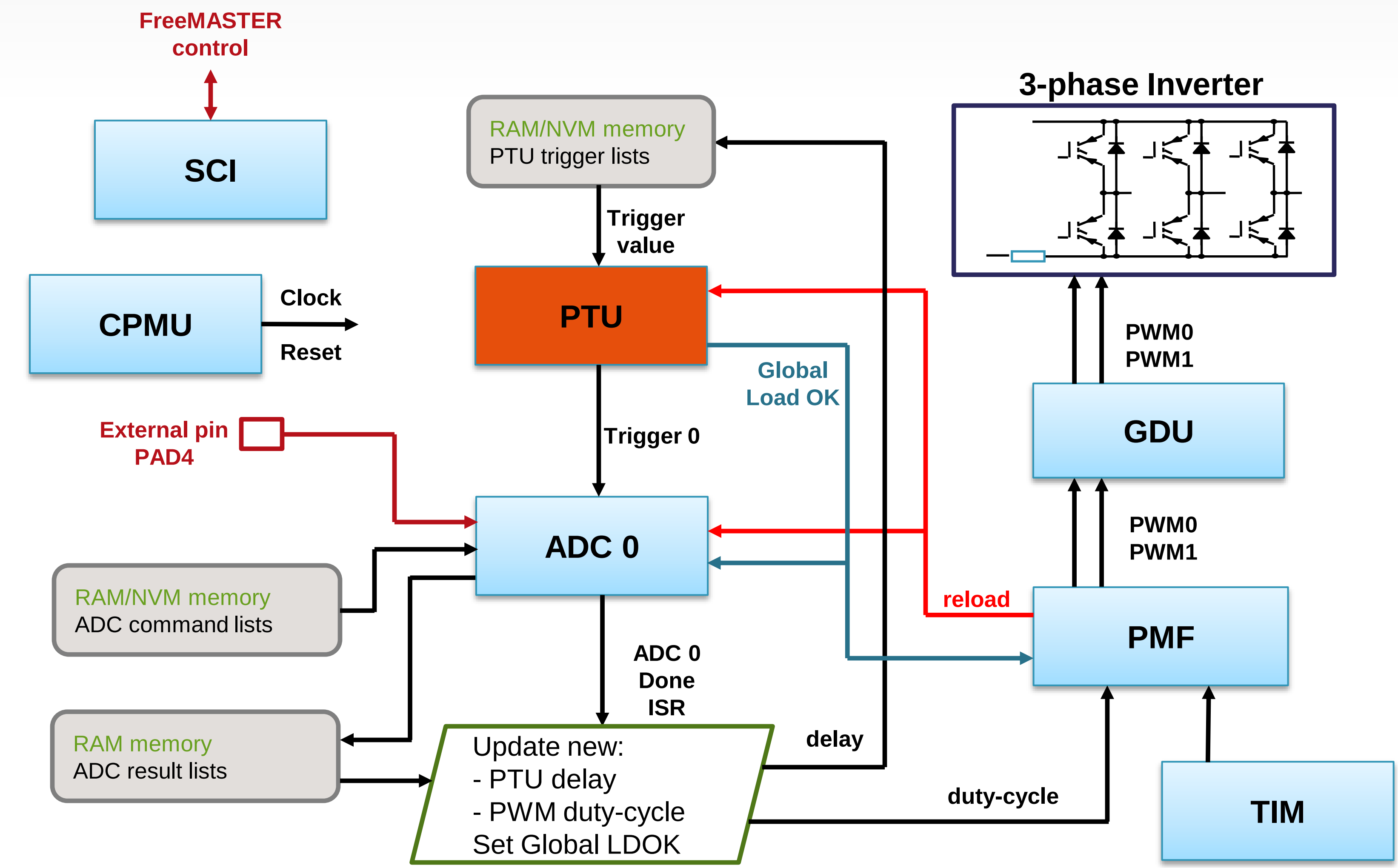
Enable generation of PWM reload event

Configure PWM frequency and dead time settings

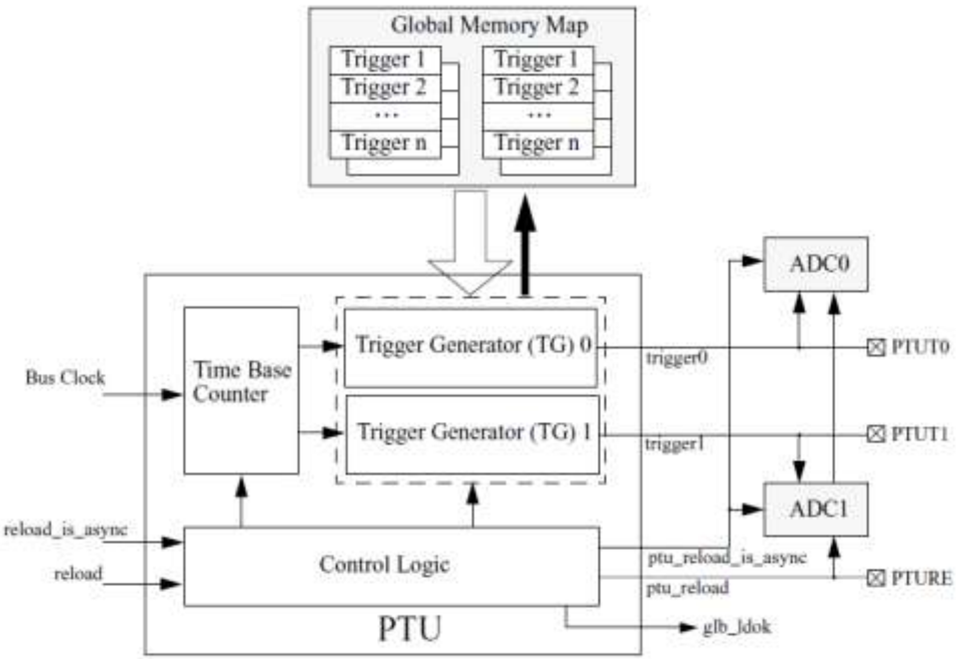
Mask PWM outputs 2 to 5 and apply settings with local Load OK

Enable PWM generator A
Enable Reload Interrupt A
Switch to Global Load OK for future updates

Application Block Diagram – PTU



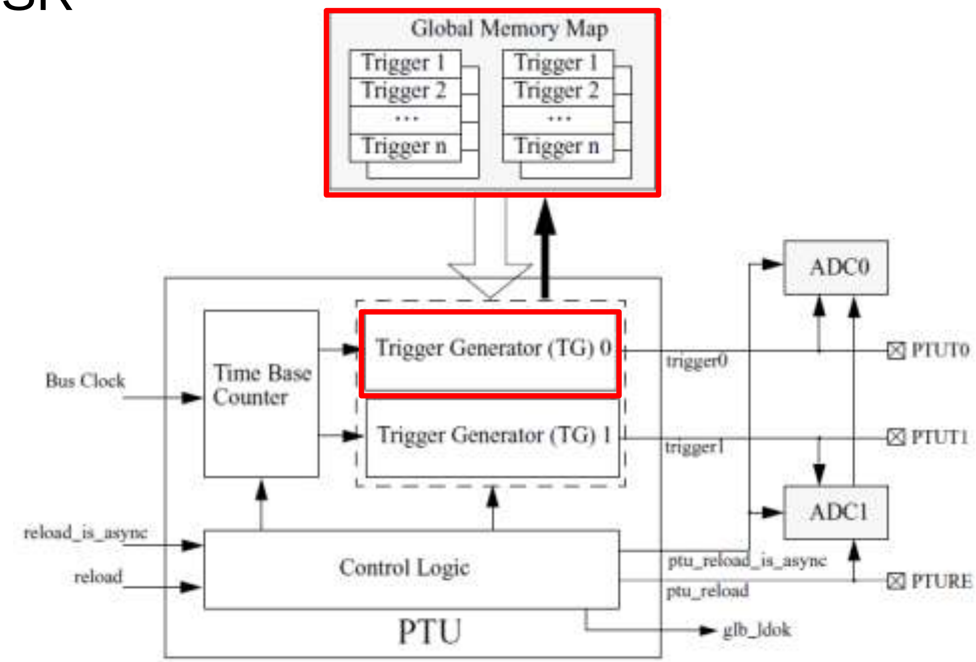
PTU – Programmable Trigger Unit



- one 16-bit counter as time base
- two independent trigger generators (TG)
- up to 32 trigger events per trigger generator
- Global Load OK support, to guarantee coherent update of all control loop modules
- trigger values stored inside the global memory map

PTU – Programmable Trigger Unit

- Configuration S/W routine:
  - Set PTU trigger list addresses
  - Enable trigger generator 0
  - Enable Trigger Done ISR



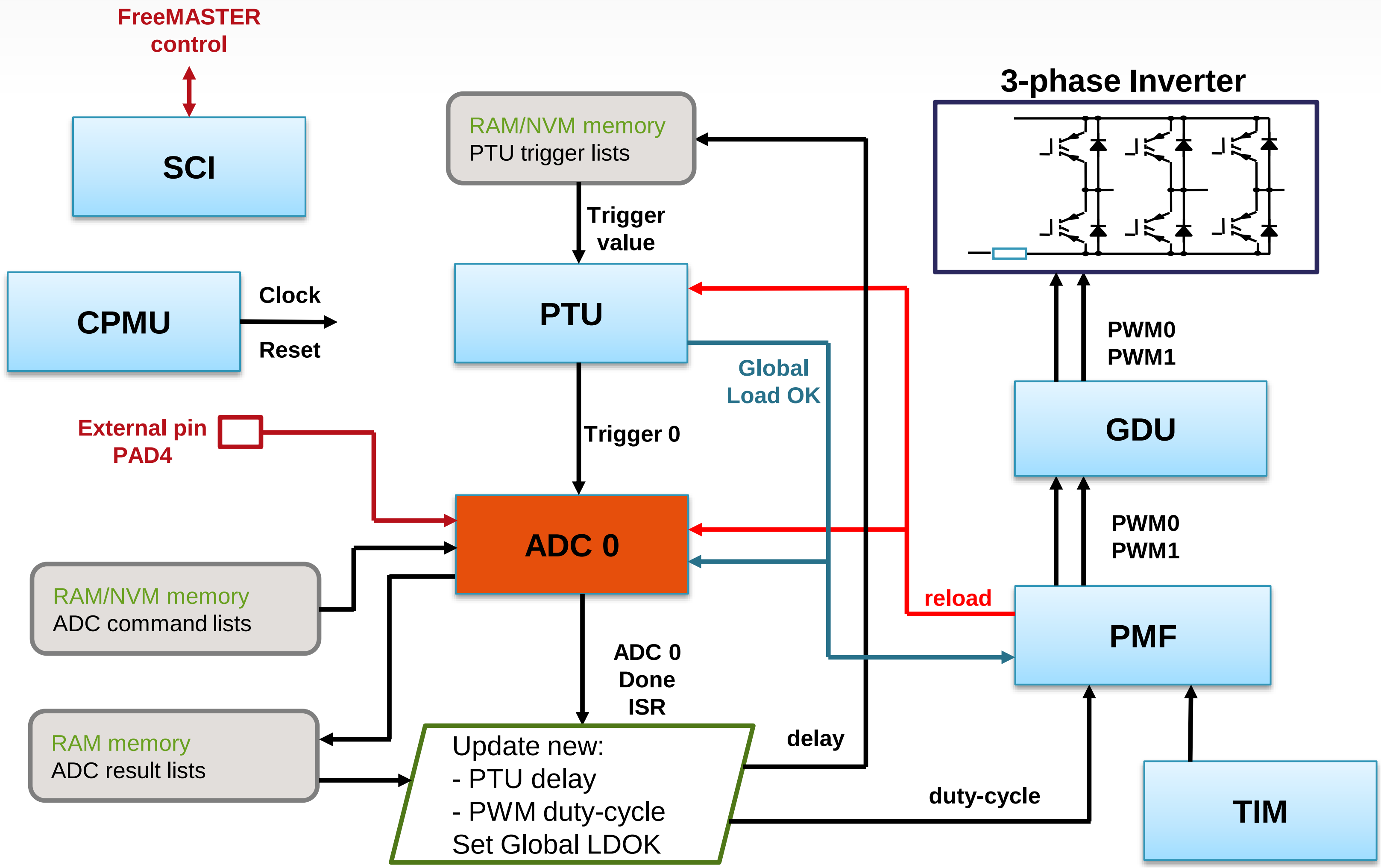
freescale™

PTU – Programmable Trigger Unit

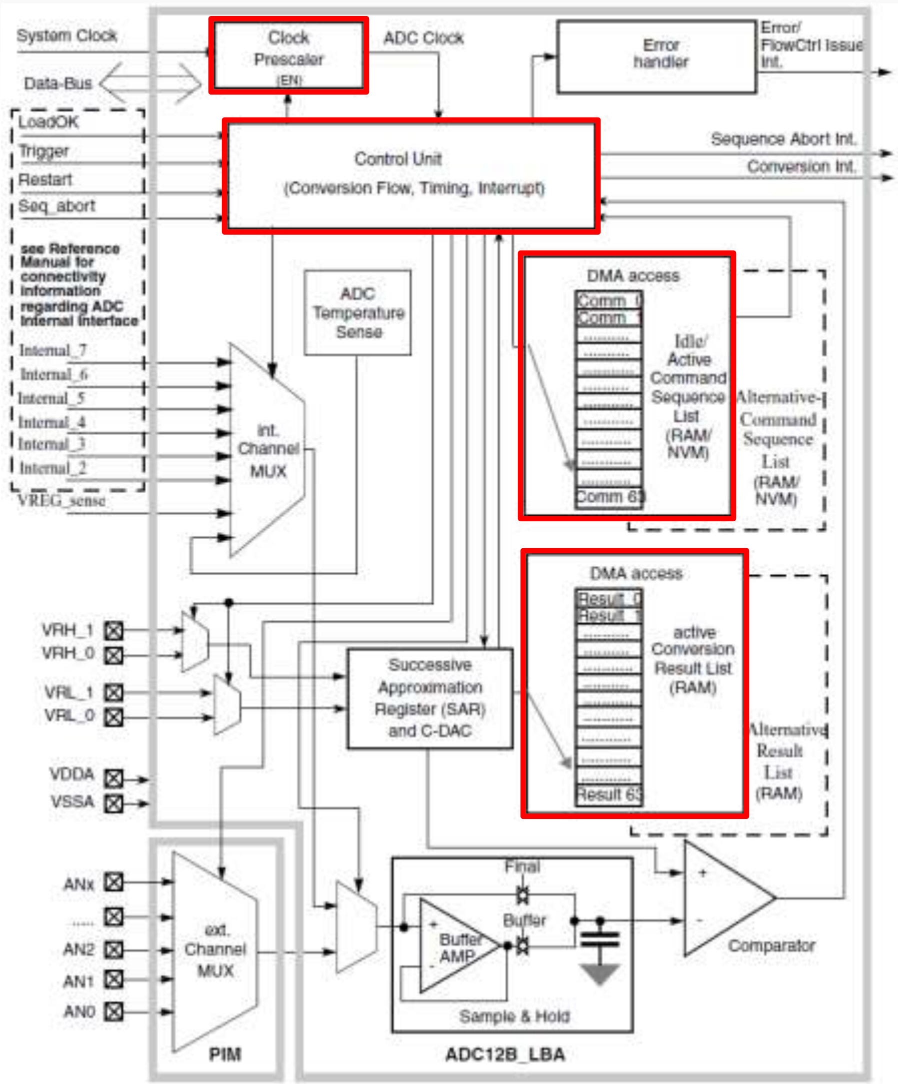
```
volatile short PTUTriggerEventList[2][3] @0x1030 =
{{0x0100,0x0000,0x0000},{0x0000,0x0000,0x0000}};

void initPTU(void)
{
   PTUIEL_TG0DIE = 1;       /* Enable Trigger Generator 0 Done Interrupt */

   /* Map start address & offset for 2nd (currently not used) list
   PTUPTR = PTUTriggerEventList;
   TG0L1IDX = 0;            /* Same as TG0L0IDX */

   PTUE_TG0EN = 1;          /* Enable Trigger Generator 0 */
}

interrupt VectorNumber_Vptutg0dn void PTUTrigger0Done_ISR(void)
{
   LED2 = 1;                /* debug pin ON */
   PTUIFL = (1 << 0);       /* Clear flag */
   LED2 = 0;                /* debug pin OFF */
}
```

freescale™

Application Block Diagram – ADC



ADC – Analog-to-Digital Converter



- Configuration S/W routine:
  - Changed to “Restart” Mode for more accurate timing of the triggers based on PTU trigger generator.
  - Changed to left alignment to use the Multiply & Saturate function to scale converted ADC values to PWM duty cycle.

freescale™

ADC Configuration

- The following changes were done to the Command List:

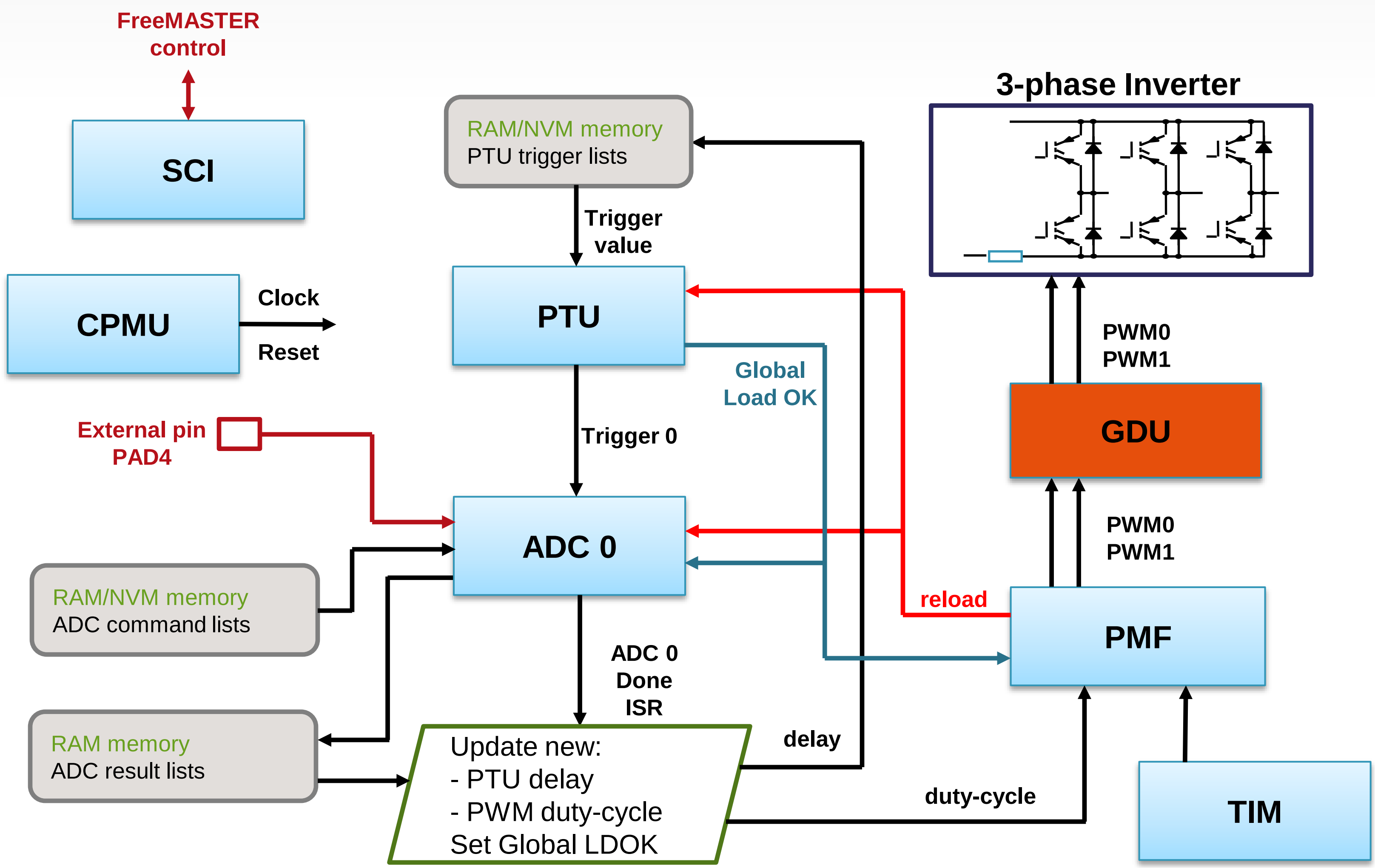
```
volatile char ADC0CommandList[4][4] @0x001000 = {
   {0x00,0xD4,0x00,0x00},            //changed from end of list to normal
   {0xC0,0xDA,0x00,0x00},            //added channel from GDU Phase MUX */
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00}
};
```

- In this case, there are no software commands to restart or trigger the ADC. The flow control is handled entirely by the PMF and PTU.

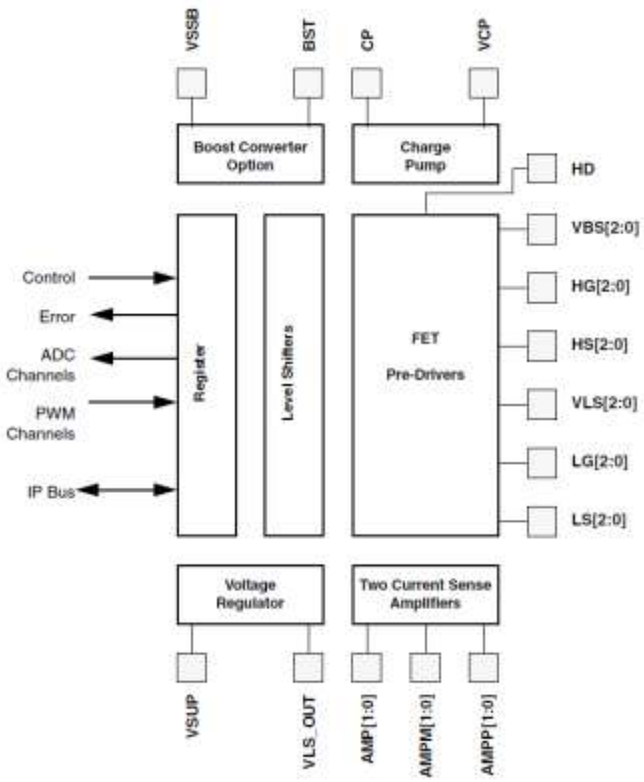
ADC Initialization

```
void initADC(void){

  ADC0CTL_0_ACC_CFG = 3;  /* Dual access mode */
  ADC0CTL_0_MOD_CFG = 0;  /* Restart mode */
  ADC0TIM = 0;    /* ADC clock = fbus / (2x(ADC0TIM + 1)) [0.25 - 8MHz] */

  ADC0FMT_DJM = 0;        /* Left justified result data */
  ADC0FMT_SRES = 4;       /* 12-bit resolution */

  /* ADC0 Command & Result Base Pointers */
  ADC0CBP = ADC0CommandList;
  ADC0RBP = ADC0ResultList;
  ADC0CROFF1 = 0;

  ADC0CONIE_1_EOL_IE = 1; /* Enable End-of-list interrupt */
  ADC0EIE = 0xEE;         /* Enable all errors interrupts */
  ADC0CTL_0_ADC_EN = 1;   /* Enable ADC0 */

  ADC0FLWCTL_RSTA = 1;    /* Issue a restart event */
  while (1 == ADC0FLWCTL_RSTA) ;   /* Wait until restart flag is cleared */
}
```

Application Block Diagram – GDU

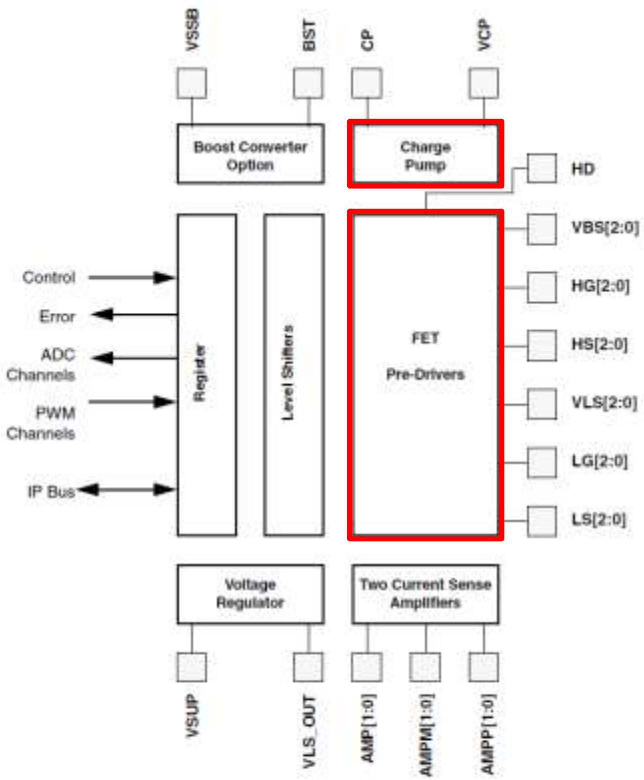


GDU - Gate Drive Unit



- 11V voltage regulator for FET pre-drivers
- Boost converter option for low supply voltage condition
- 3-phase bridge FET pre-drivers
- Bootstrap circuit for high-side FET pre-drivers
- Charge pump for static high-side driver operation
- Phase voltage measurement with internal ADC
- Two low-side current measurement amplifiers for DC phase current measurement
- Phase comparators for BEMF zero cossing detection in sensorless BLDC applications
- Voltage measurement on DC-Link voltage with internal ADC
- Desaturation comparator for high-side drivers and low-side drivers protection
- Undervoltage detection on FET pre-driver supply pin VLS
- Two overcurrent comparators with programmble voltage threshold
- Overvoltage detection on 3-phase bridge supply HD pin

GDU - Gate Drive Unit



- Configuration S/W routine:
  - Clear error flags
  - Configure and enable charge pump
  - Set blanking time to 1.1 us
  - Set desaturation level to 1.35V
  - Enable FET pre-driver
  - Clear desaturation error flags
  - Route Phase A from the Phase voltage Multiplexer to the ADC

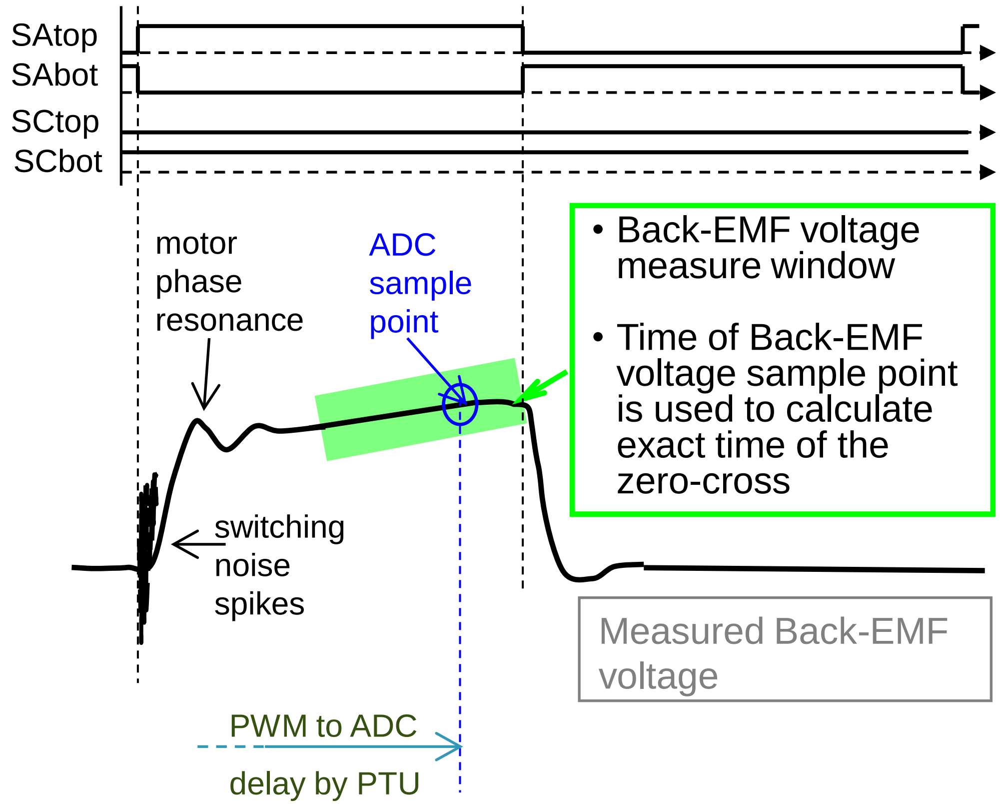
GDU Initialization

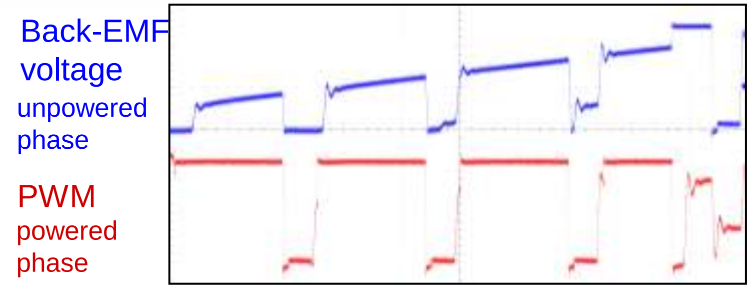
```
void initGDU(void){

   GDUE_GCPE = 1;   /* Enable charge pump */
   GDUF = 0xFF;     /* Clear High & Low Voltage Supply flags */
   GDUCLK2_GCPCD = 2;        /* Set GDU charge pump clock divider to fbus / 32 */
   GDUCTR = 0x09;   /* blanking time 14/12.5M = 1.1us, HD OV threshold = 20V */
   GDUDSLVL = 0x77;          /* Desaturation level (1.35 V) */
   GDUE_GFDE = 1;   /* Enable FET pre-driver */
   GDUDSE = 0x77;   /* Clear Desaturation Error Flags */
   GDUPHMUX = 0x01;/* Route Phase 0 to ADC multiplexer */

}
```

freescale™

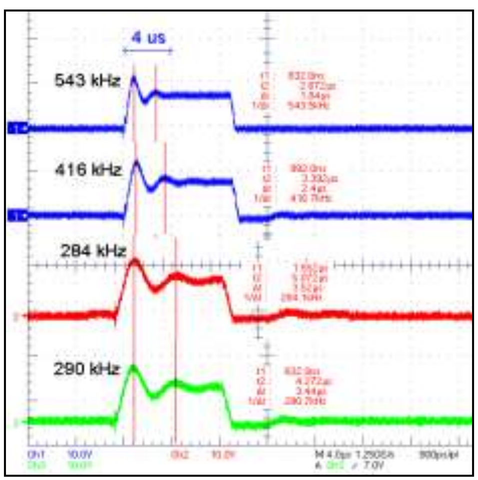
Back-EMF Voltage Measurement

- Back-EMF voltage can not be measured within all the active PWM pulse as there is switching noise and resonance transient at the beginning of the PWM pulse

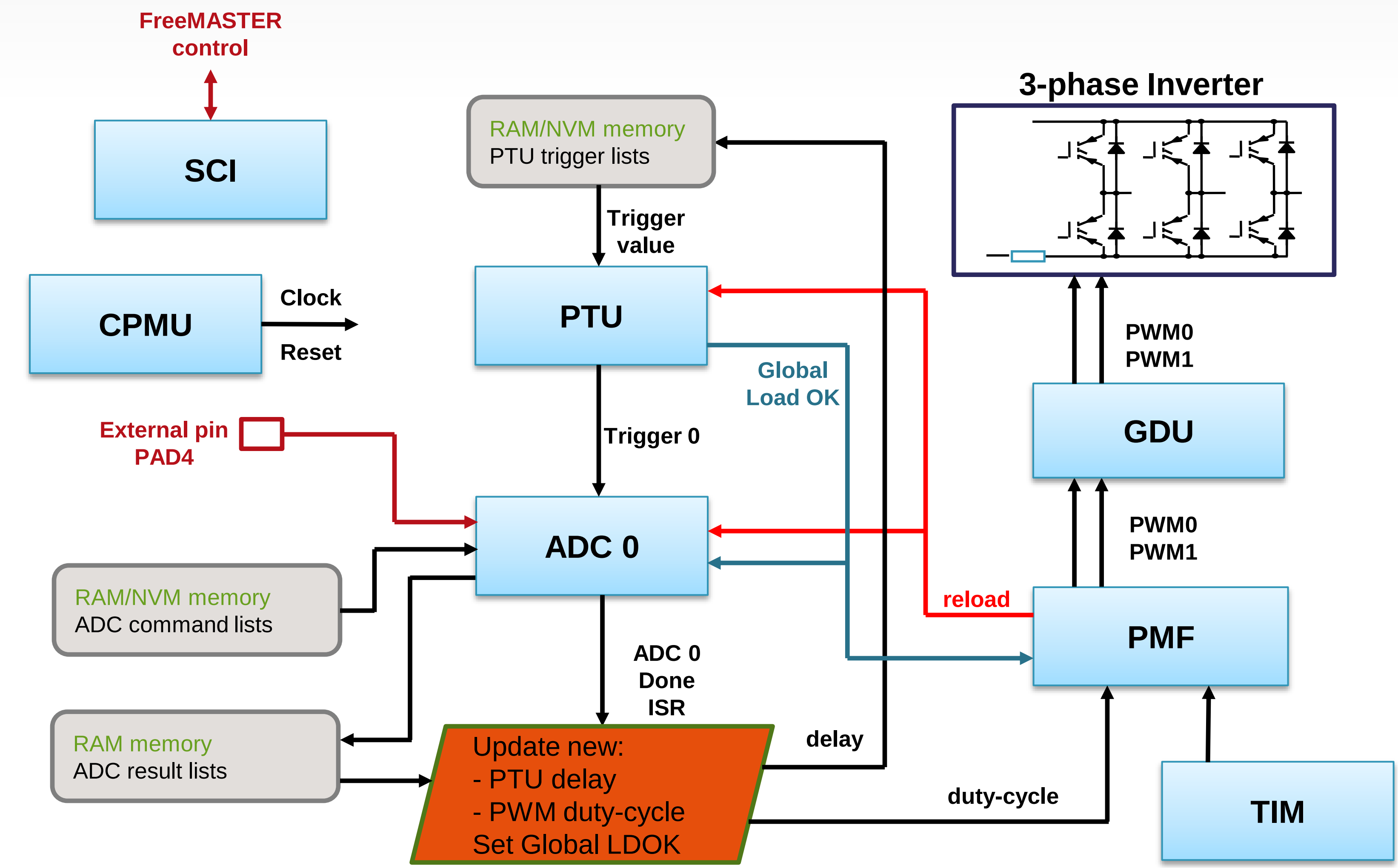




Resonance transient on Back-EMF voltage depends on motor and power stage parameters



Application Block Diagram – ADC ISR



ADC Conversion Complete ISR

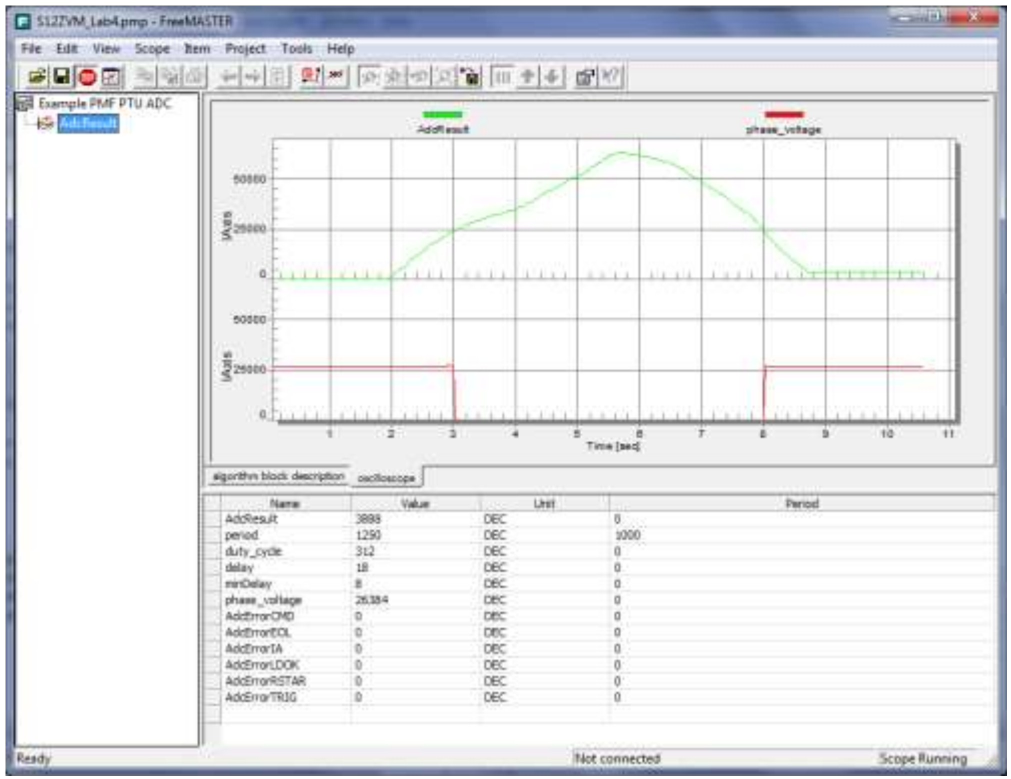
```
interrupt VectorNumber_Vadc0conv_compl void ADC0done_ISR(void)
{
  LED2 = 1;       /* debug pin ON */
  AdcResult = ADC0ResultList[0];
  phase_voltage = ADC0ResultList[1];

  /* scale ADC result value to PWM modulo value */
  delay = MLIB_MulSat(PWM_MODULO, ADC0ResultList[0]>>1);
  delay = delay>>2;       /* take 1/4 of the calculated value */
  if (delay < minDelay) delay = minDelay;

  PTUTriggerEventList[0][0] = delay;
  PTUC_PTULDOK = 1;

  ADC0CONIF = 1;       /* Clear flag */
  LED2 = 0;       /* debug pin OFF */
}
```

PMF Reload & PTU Trigger Done ISR

```
interrupt VectorNumber_Vpmfra void PMFreloadA_ISR(void)
{
  volatile char tmp;

  LED1 = 1;       /* debug pin ON */
  tmp = PMFFQCA;
  PMFFQCA = PMFFQCA_PWMRFA_MASK;   /* Clear flag */
  LED1 = 0;       /* debug pin OFF */

}


interrupt VectorNumber_Vptutg0dn void PTUTrigger0Done_ISR(void)
{
  LED2 = 1;       /* debug pin ON */
  PTUIFL = (1 << 0);       /* Clear flag */
  LED2 = 0;       /* debug pin OFF */
}
```

Visualizing the data in FreeMASTER

- Run the FreeMASTER application
  - Move the potentiometer and find the threshold at which the phase voltage returns 0.



freescale™

What we have learned until now?

- How to configure the PMF to output a 20KHz PWM signal
- How to configure the PTU and define a list of triggers
- How to enable the control loop events for synchronization of ADC trigger events based on PMF and PTU
- How to configure the GDU and route a phase voltage to ADC

6-step commutation

Hands-on session # 5

6-step commutation

- The PMF initialization now enables all three phases in edge-aligned PWM operation
- The FET outputs are configured in complementary mode
- The commutation event is enabled, and the appropriate phases will be configured in preparation for the next commutation event indicated by the TIM output compare channel 0
- The potentiometer value directly sets the PWM duty cycle
  - CAUTION! No speed control implemented

freescale™

BLDC Motor Commutation Sequence

- The following table shows the Hall sensor patterns corresponding to each Commutation vector in the motor part of the S12ZVM motor control kit

Table 2. Linix 45ZWN24-90 clockwise direction commutation sequence

Commutation vector			Vector	Hall sensor pattern definition			Decimal result
Phase A	Phase B	Phase C		Hall C	Hall B	Hall A	
$+V_{DBC}$	$-V_{DBC}$	NC	0	0	0	1	1
$+V_{DBC}$	NC	$-V_{DBC}$	1	0	1	1	3
NC	$+V_{DBC}$	$-V_{DBC}$	2	0	1	0	2
$-V_{DBC}$	$+V_{DBC}$	NC	3	1	1	0	6
$-V_{DBC}$	NC	$+V_{DBC}$	4	1	0	0	4
NC	$-V_{DBC}$	$+V_{DBC}$	5	1	0	1	5

Table 3. Linix 45ZWN24-90 counterclockwise direction commutation sequence

Commutation vector			Vector	Hall sensor pattern definition			Decimal result
Phase A	Phase B	Phase C		Hall C	Hall B	Hall A	
$-V_{DBC}$	$+V_{DBC}$	NC	3	0	0	1	1
NC	$+V_{DBC}$	$-V_{DBC}$	2	1	0	1	5
$+V_{DBC}$	NC	$-V_{DBC}$	1	1	0	0	4
$+V_{DBC}$	$-V_{DBC}$	NC	0	1	1	0	6
NC	$-V_{DBC}$	$+V_{DBC}$	5	0	1	0	2
$-V_{DBC}$	NC	$+V_{DBC}$	4	0	1	1	3

Commutation Sector Constant Declaration

- The previous values are used to define the masks for the PWM outputs, and their sequence depending of Clockwise or Counter-clockwise operation

```
const unsigned char MaskVal[7] = {0x34, 0x1C, 0x13, 0x31, 0x0D, 0x07, 0x3F};
const unsigned char OutCtl[7]  = {0x0C,0x30, 0x30,0x03, 0x03,0x0C, 0x00};

const unsigned char BLDCPatternBasedOnHall[2][7] = {
          {0, 0, 2, 1, 4, 5, 3},
          {0, 3, 5, 4, 1, 2, 0}
};
```

Applying the new masks

- Inside the TIM channel 1 ISR (occurring upon every edge of any of the Hall Sensor signals), the hall pattern is evaluated and the commutation sector is determined.

- The PWM mask and the SW controlled outputs are configured, and a commutation event is forced on TIM output compare channel 0

```
hall_pattern = 0x07 & (PTIT >> 1);
cmtSector = BLDCPatternBasedOnHall[rotDir][hall_pattern];
PMFCFG2 = 0x40 + MaskVal[cmtSector];
PMFOUTC_OUTCTL = OutCtl[cmtSector];
TIM0CFORC_FOC0 = 1;
```

freescale™

Running the project

- CAUTION: move the potentiometer all the way clockwise, to set the minimum duty cycle

- When the code is executed, it will immediately start spinning the motor. The speed is set with the potentiometer.
  - There is no speed control, so please avoid any drastic changes to the potentiometer settings to make it easy for the motor to adjust to its new speed.

freescale™

References

References

- Freescale Automotive Motor Control Development Solutions:

http://www.freescale.com/webapp/sps/site/overview.jsp?code=AUTOMCDEVKITS

- S12ZVM Motor Control Kit for BLDC applications

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MTRCKTSBNZVM128

- S12ZVM family web site

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=S12ZVM

freescale™

Q & A

Thank you!

freescale™