Edge Node Development with MagniV

Allen Willson

FAE

October 2019 | Session #AMF-AUT-T3876

SECURE CONNECTIONS FOR A SMARTER WORLD

Agenda

01. MagniV Product Overview

02. DEVKIT-S12ZVC Development Board

03. Set up the Development Environment

04. Hands-on Sessions

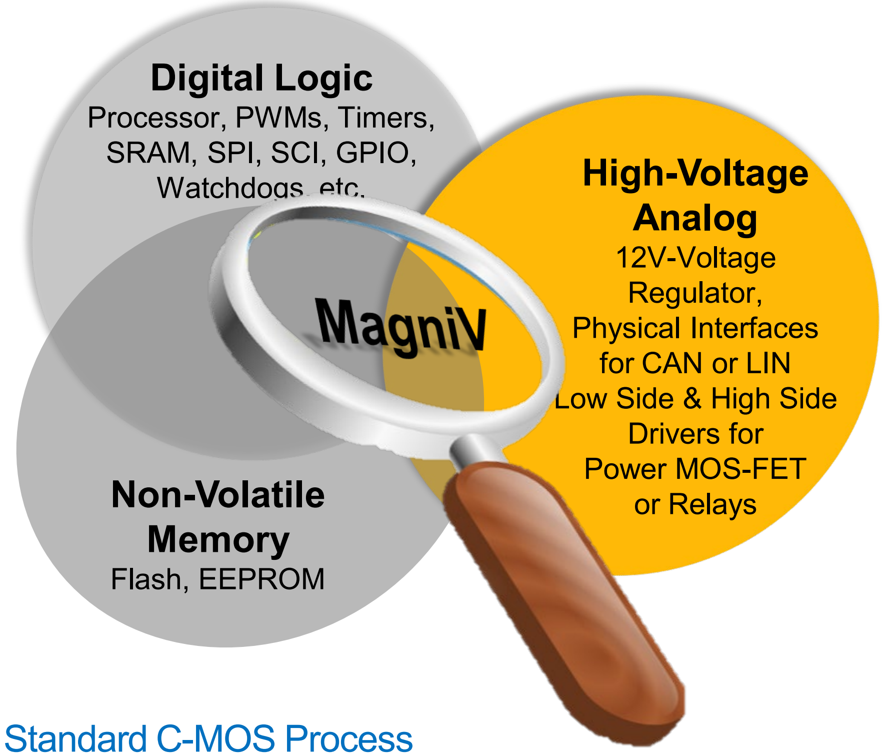
01. MagniV Overview

S12 MagniV – Integrated Solutions

Technology Sweet-spot for Actuators, Sensors & User Interfaces

Act *Motorcontrol*

Sensor Interface Sense



Digital Logic
Processor, PWMs, Timers,
SRAM, SPI, SCI, GPIO,
Watchdogs, etc.

Non-Volatile Memory
Flash, EEPROM

High-Voltage Analog
12V-Voltage Regulator,
Physical Interfaces for CAN or LIN
Low Side & High Side Drivers for Power MOS-FET or Relays

Standard C-MOS Process
(LL18 Low Leakage 180nm)

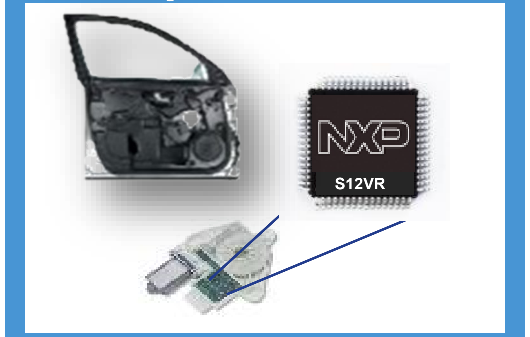
UHV Process

S12 MagniV: Product Families

Act *Motorcontrol*

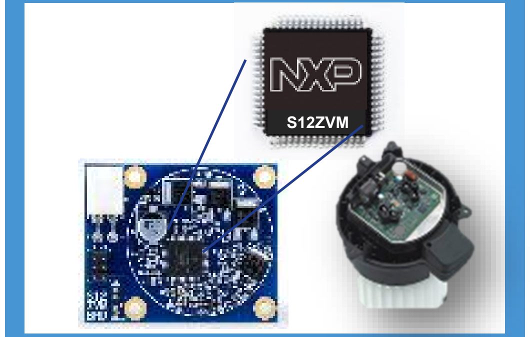
Sensor Interface Sense

S12VR
Relay driven motors



- Window lift
- Sunroof
- Power doors

S12ZVM
BLDC/DC motors

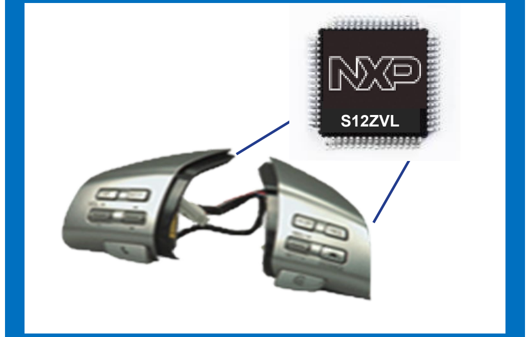


- Fuel pump
- Oil pump
- Fans
- Wipers

S12ZVC
CAN nodes

- Safety sensors
- Emission sensors
- Gear shift

S12ZVL
LIN Nodes



- Sensors
- Door modules
- Steering wheel switches

✓ Reduced PCB Space

✓ Reduced Bill of Material

✓ Improved manufacturing efficiency and quality

✓ Simplified development

Industry’s broadest portfolio of integrated solutions for motor control and interface nodes

- Broadest memory range – up to 256K
- Industry’s only integrated CAN Phy
- True AEC-Q100 Grade 0 Temperature performance (up to 150C Ta)
- ASIL-A-compliant, ASIL-B-capable (S12Z Devices)
- Unmatched tools and software ecosystem
- Conforms to global robustness standards and OEM requirements

MagniV Building Blocks

High-Voltage Components

- **VREG for tot. supply:**
  - 70mA w/o ext comp. or
  - 170mA with ext. ballast
- **LIN-PHY**
- **CAN-PHY**
- **V-BAT SENSE**
- **V-SUP SENSE**
- **1-4ch HVI** (12V-input with wake-up and ADC)
- **LS-drivers**
- **HS-drivers**
- **Charge Pump**
- **4-6ch Gate Drive Unit for FET Qg=50-150nC**

Digital Components

- **MSCAN**
- **Sent**
- **SCI**
- **SPI**
- **IIC**
- **GPIO**
- **PGPIO 20mA**
- **NGPIO 25mA**
- **BDM/BDC**
- **Key Wakeups**
- **Win Wdog**
- **RTC**
- **Timer** 16Bit (25-64MHz)
- **PWM** 8/16Bit (25-64MHz)
- **Motorcontrol PWM With Fault protection**
- **Programmable Trigger Unit**
- **Sound Generator**
- **Segment LCD (4x40)**
- **Stepper Motor Driver with SSD**

MCU Core and Memories

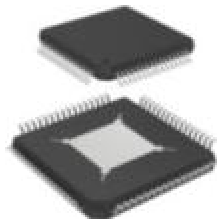
- **S12- or S12Z-CPU 25/32/50MHz bus**
- **Flash (ECC) 8kB – 192kB**
- **EEPROM (ECC) 128B – 4kB**
- **RAM (ECC) 512B - 8kB**

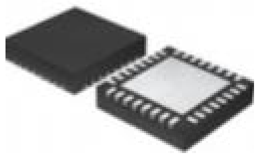
5V Analogue Components

- **List Based ADC**
  10-12Bit resolution
  1-2 S/H-units
  Up to 16ch total
- **Temp Sense**
- **Current Sense** (2 x Op-Amp)
- **2ch ACMP** With 1x6-Bit-DAC
- **Pierce Osc.**
- **32kHz low power Osc**
- **RCosc. +/-1.3%**
- **PLL**

Packaging

- **LQFP:** 32/48/64/100/144-pin
- **LQFP-EP:** 48/64-pin
- **QFN:** 32-pin (5x5mm)

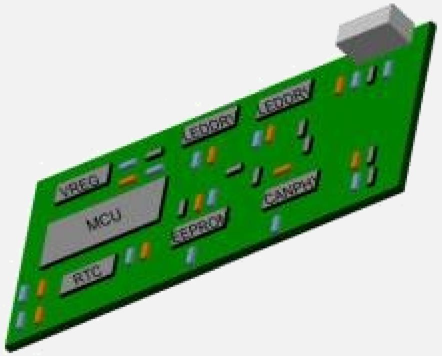


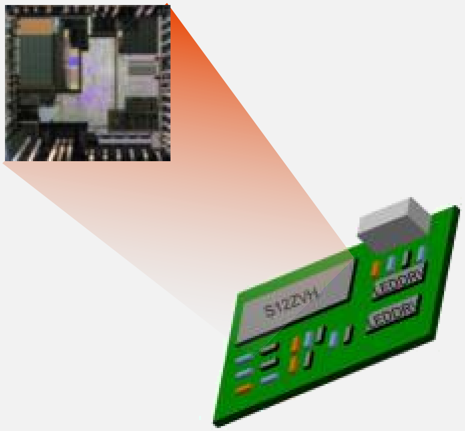


MagniV Concept: Shrink Your Application

Integration of High-Voltage (HV) Analog Features into a Standard Automotive MCU

Traditional Solution	MagniV
• Standard MCU • Multiple analog ICs	• Single device • Reduced space





SafeAssure^{TM} Program Applied to S12 MagniV

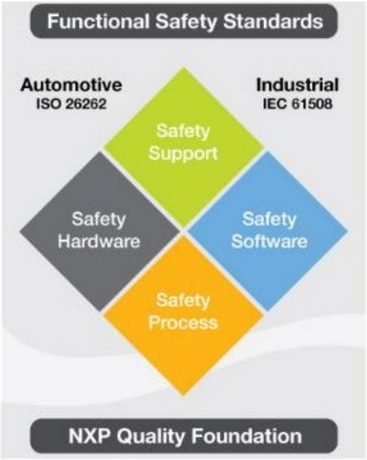
Safety Hardware

Common safe hardware platform for application software:

✓ Voltage/clocks monitoring
✓ Memories w/ error correction
✓ Window Watchdog...

Safety Process

✓ ISO26262 development process for most products
✓ Safety-Element-Out-Of-Context



Safety Support

✓ FIT rates
✓ Dynamic FMEDA
✓ Safety manual
✓ Technical support as required

Safety Software

S12Z core self-test available to complement the built-in hardware safety features

Product Families	Part Codename	Development Process	FMEDA Report Availability	Dependant Failure Analysis	Safety Manual	Core Self test and User Guide
S12ZVL	Knox	ISO 26262	Yes (autopad)	Yes, included in FMEDA report (autopad)	Yes (www)	Yes (autopad)
S12ZVC	Hearst	ISO 26262				
S12ZVM	Carcassonne	Standard				
S12ZVM	Obidos	ISO 26262				
S12ZVM	Carcassonne+	ISO 26262	Q4'16			
S12ZVMB	Toledo	ISO 26263				
S12ZVMA	VMA32	ISO 26264				
S12ZVH/VHY	Lumen2W	Standard	Upon request			
S12ZVH	Lumen4W				No	No
S12VR	Tomar					
S12VR	Tomarino					
S12VRP	Tomar+					

S12ZVC Family (Hearst)

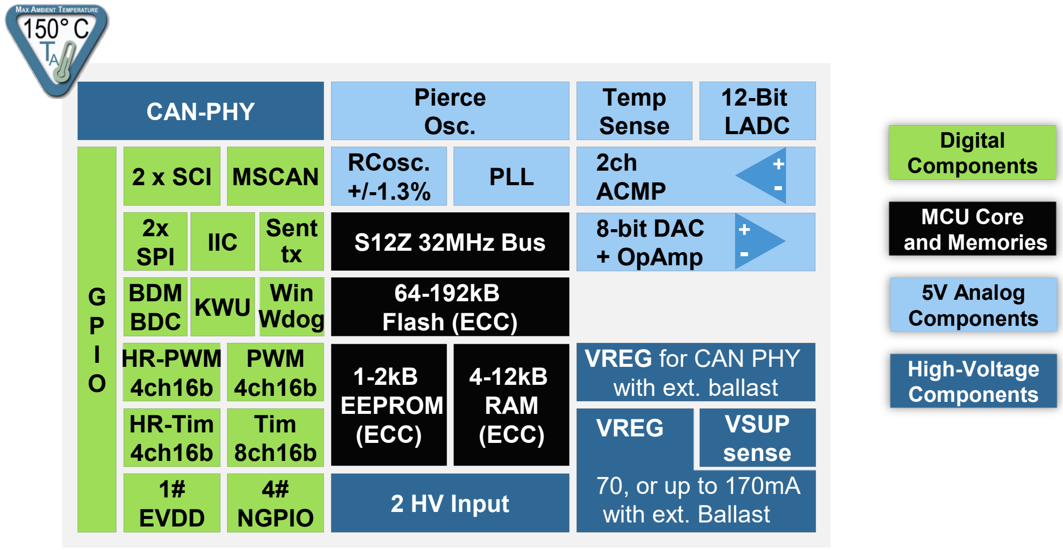
Integrated small CAN nodes

Key Features:

- S12Z core (up to 32MHz bus frequency)
- On chip CAN PHY:
  - CAN-supply requires ext. Ballast Transistor
  - dominant Txd timeout
  - Emission limits of major OEMs can be met without need of Choke (up to 500kbps)
- On-chip-voltage-regulator - Supply-cpability:
  - 70mA total with no ext components
  - 170mA total with ext. Ballast transistor
- High Voltage Input (HVI) with internal connection to ADC for analog 12V measurements
- Specific features for sensor type applications:
  - list-based 12-Bit ADC (LADC)
  - 16ns resolution Timer / PWM
  - 2x Analog Comparator with 8Bit DAC
  - SENT
- ISO26262 support (FMEDA, safety guide)

Target Applications:

- Any kind of automotive CAN-node (non-Autosar)
- Powertrain sensors & actuators
- CAN-based user-interfaces



Max Ambient Temperature
150° C
T_A

CAN-PHY
Pierce Osc.
Temp Sense
12-Bit LADC
GPIO
2 x SCI
MSCAN
RCosc. +/-1.3%
PLL
2ch ACMP
2x SPI
IIC
Sent tx
S12Z 32MHz Bus
8-bit DAC + OpAmp
BDM BDC
KWU
Win Wdog
64-192kB Flash (ECC)
HR-PWM 4ch16b
PWM 4ch16b
1-2kB EEPROM (ECC)
4-12kB RAM (ECC)
VREG for CAN PHY with ext. ballast
HR-Tim 4ch16b
Tim 8ch16b
VREG
VSUP sense
1# EVDD
4# NGPIO
2 HV Input
70, or up to 170mA with ext. Ballast

Digital Components
MCU Core and Memories
5V Analog Components
High-Voltage Components

Family Options:

- Flexible Memory Options: 64kB to 128kB Flash version
- 48-LQFP or 64-LQFP-EP Packaging
- C / V / M / W Temperature options (up to 150°C Ta)
- Fully featured (S12ZVCAx) or reduced featureset S12ZVCx)

S12ZVC Family Differences in Feature Set

Product Name	S12ZVCAx (fully featured)				S12ZVCx (reduced feature set)			
Package	64-LQFP-EP		48-LQFP		64-LQFP-EP		48-LQFP	
Flash memory (ECC)	192 / 128 / 96kB	64kB	192 / 128 / 96kB	64kB	192 / 128 / 96kB	64kB	192 / 128 / 96kB	64kB
EEPROM (ECC)	2kB	1kB	2kB	1kB	2kB	1kB	2kB	1kB
RAM (ECC)	12kB	4kB	12kB	4kB	12kB	4kB	12kB	4kB
CAN / SCI / SPI / IIC	1/2/2/1	1/2/2/1	1/1/1/1	1/1/1/1	1/2/2/1	1/2/2/1	1/1/1/1	1/1/1/1
SENT (Tx)	1	1	1	1	1	1	1	1
16-bit Timer (16ns)	4ch	4ch	4ch	4ch	4ch	4ch	4ch	4ch
16-bit Timer (std.)	8ch	8ch	4ch	4ch	8ch	8ch	4ch	4ch
16-bit PWM (16ns)	4ch	4ch	3ch	3ch	4ch	4ch	3ch	3ch
16-bit PWM (std.)	4ch	4ch	4ch	4ch	4ch	4ch	4ch	4ch
LADC	16c/12b	16c/12b	10c/12b	10c/12b	16c/10b	16c/10b	10c/10b	10c/10b
ACMP 5V (rail to rail)	2	2	2	2	-	-	-	-
DAC (8-bit)	1	1	1	1	-	-	-	-
Temperature	V / M / W		C / V / M		V / M / W		C / V / M	

MagniV S12ZVC One Pager

S12ZVC Smallest Integrated CAN MCU

- **System in a Package -** Highly integrated part which is ideal for space constrained applications such as Actuators, Sensors, CAN nodes etc.
- **Low System Cost -** Directly powered by Battery. Integrated CAN Phy, Vreg, High Voltage pins, and Op Amps reduce system-, test-, qualification- and manufacturing cost. Emission limits of major OEMs can be met without need of Choke (up to 500kbps)
- **High Reliability -** High immunity to EMI and ESD stresses, CAN HS/LS compliant with +/- 8kV ESD capability.
- **Enablement -** Supported by comprehensive hardware and software system (free low-level drivers to enterprise 3rd party tools) which reduces development costs and time to market.

S12ZVC(A)			
Flash	64 – 192 kB	**12V VREG**	12V/70mA, 170mA with ballast
RAM	4-12 kB	**EVDD**	1ch 5V/20mA (source)
EEPROM	1-2 kB	**NGPIO**	4ch 5V/25mA (sink)
Core	S12Z	**ADC**	10-16ch 10Bit (12Bit)
Speed	32 MHz	**DAC**	8bit DAC with OpAmp
Op Range	5.5V – 18V	**Comparator**	2# rail to rail
HVI	2	**Timer**	8ch/16B + 4ch/16B (16ns)
CAN Phy	1	**PWM**	4ch16B + 4ch/16B (16ns)
Op range	5.5V – 18V	**Comms**	1MSCAN, 2SCI, 2SPI, 1IIC, 1SENT-Tx
Temp	150°C Ta	**Packages**	64-LQFP-EP, 48-LQFP

Part Numbers			
	64LQFP EP	**48LQFP**	**Flash**
Enhanced Analog	S912ZVCA19F0VKH	S912ZVCA19F0CLF	192kB
	S912ZVCA12F0VKH	S912ZVCA12F0CLF	128kB
	S912ZVCA96F0VKH	S912ZVCA96F0CLF	96kB
	S912ZVCA64F0VKH	S912ZVCA64F0CLF	64kB
Reduced Analog	S912ZVC19F0VKH	S912ZVC19F0CLF	192kB
	S912ZVC12F0VKH	S912ZVC12F0CLF	128kB
	S912ZVC96F0VKH	S912ZVC96F0CLF	96kB
	S912ZVC64F0VKH	S912ZVC64F0CLF	64kB
Temp Options	"V"=105°C Ta; "M"=125°C Ta		
	"W"=150°C Ta	“C”=85°C Ta	

MagniV™

Directly powered by Car Battery

Integrated CAN Transceiver

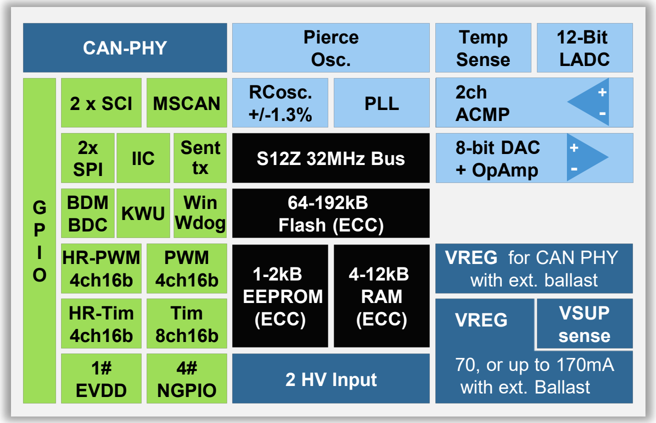
Safe Assure™

Ultra-Reliable Industrial

15 Year Longevity

Fast 12Bit ADC, 2 Op Amps, high res PWM/Timer

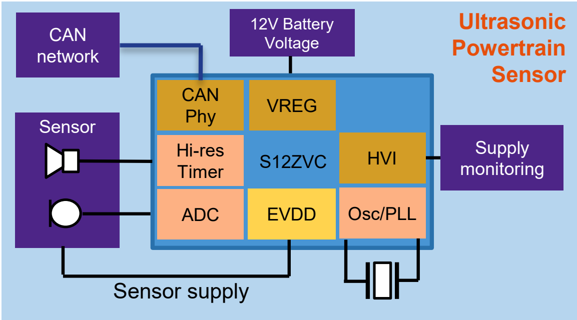
AEC-Q100 Grade0
Up to 150°C Ta



CAN-PHY
Pierce Osc.
Temp Sense
12-Bit LADC
GPIO
2 x SCI
MSCAN
RCosc. +/-1.3%
PLL
2ch ACMP
2x SPI
IIC
Sent tx
S12Z 32MHz Bus
8-bit DAC + OpAmp
BDM BDC
KWU
Win Wdog
64-192kB Flash (ECC)
HR-PWM 4ch16b
PWM 4ch16b
1-2kB EEPROM (ECC)
4-12kB RAM (ECC)
VREG for CAN PHY with ext. ballast
HR-Tim 4ch16b
Tim 8ch16b
VREG
VSUP sense
1# EVDD
4# NGPIO
2 HV Input
70, or up to 170mA with ext. Ballast

Targeted Applications

- CAN nodes
- CAN switch panel / user interface
- CAN actuators, sensors
- HVAC
- Lighting controls
- Seat positioning
- Seatbelt pretentionner
- Ultrasonic Sensors
- Occupant detection
- Powertrain Sensors (Nox)



Ultrasonic Powertrain Sensor

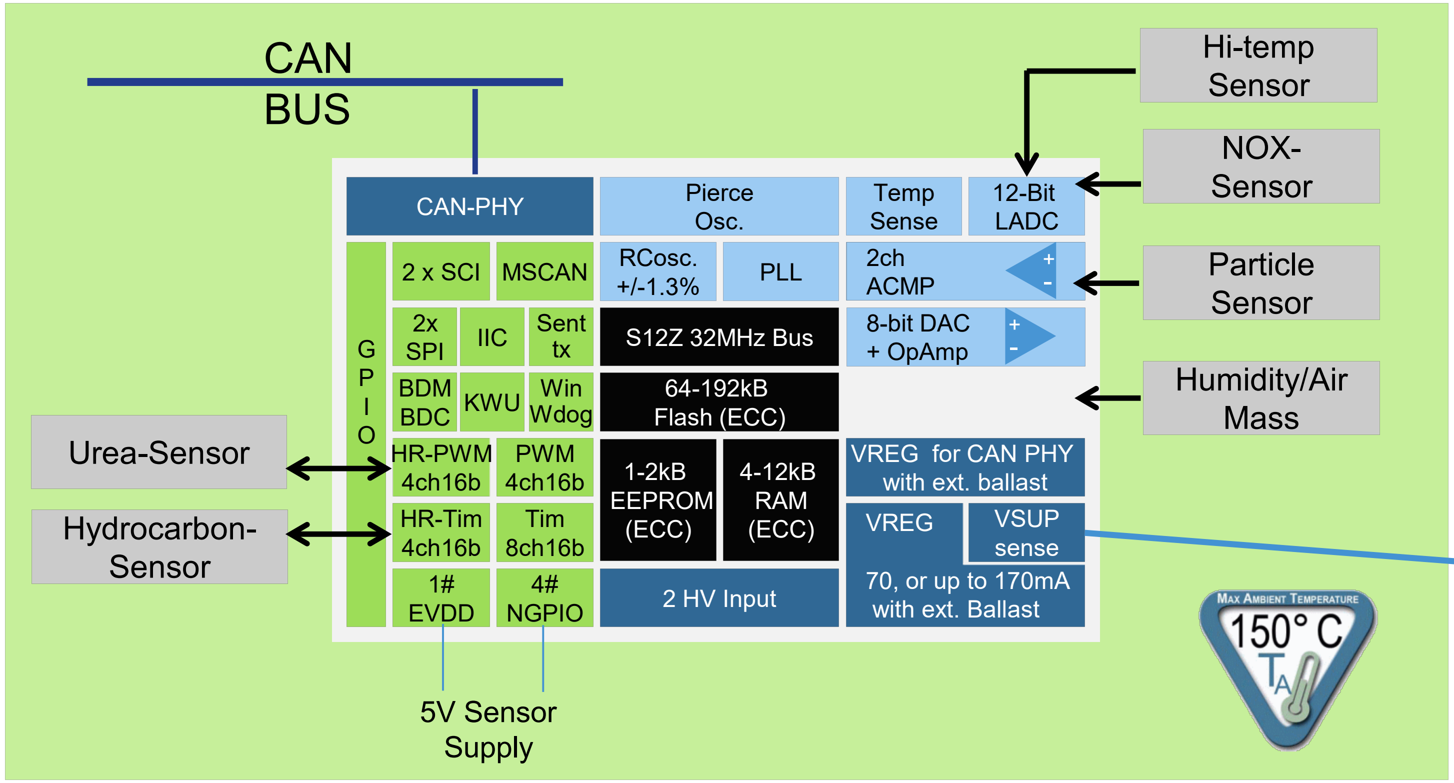
Enablement Tools

- Evaluation Boards / Hardware
  - VLG-MC9S12ZVC
- CodeWarrior, Cosmic
- LIN drivers

CLICK FOR

Sample | Data Sheet | Tools

S12ZVC for Sensors in Powertrain



CAN BUS

CAN-PHY | Pierce Osc. | Temp Sense | 12-Bit LADC

GPIO

2 x SCI | MSCAN | RCosc. +/-1.3% | PLL | 2ch ACMP

2x SPI | IIC | Sent tx | S12Z 32MHz Bus | 8-bit DAC + OpAmp

BDM BDC | KWU | Win Wdog | 64-192kB Flash (ECC)

HR-PWM 4ch16b | PWM 4ch16b | 1-2kB EEPROM (ECC) | 4-12kB RAM (ECC) | VREG for CAN PHY with ext. ballast

HR-Tim 4ch16b | Tim 8ch16b | VREG | VSUP sense

1# EVDD | 4# NGPIO | 2 HV Input | 70, or up to 170mA with ext. Ballast

5V Sensor Supply

Hi-temp Sensor

NOX-Sensor

Particle Sensor

Humidity/Air Mass

Urea-Sensor

Hydrocarbon-Sensor

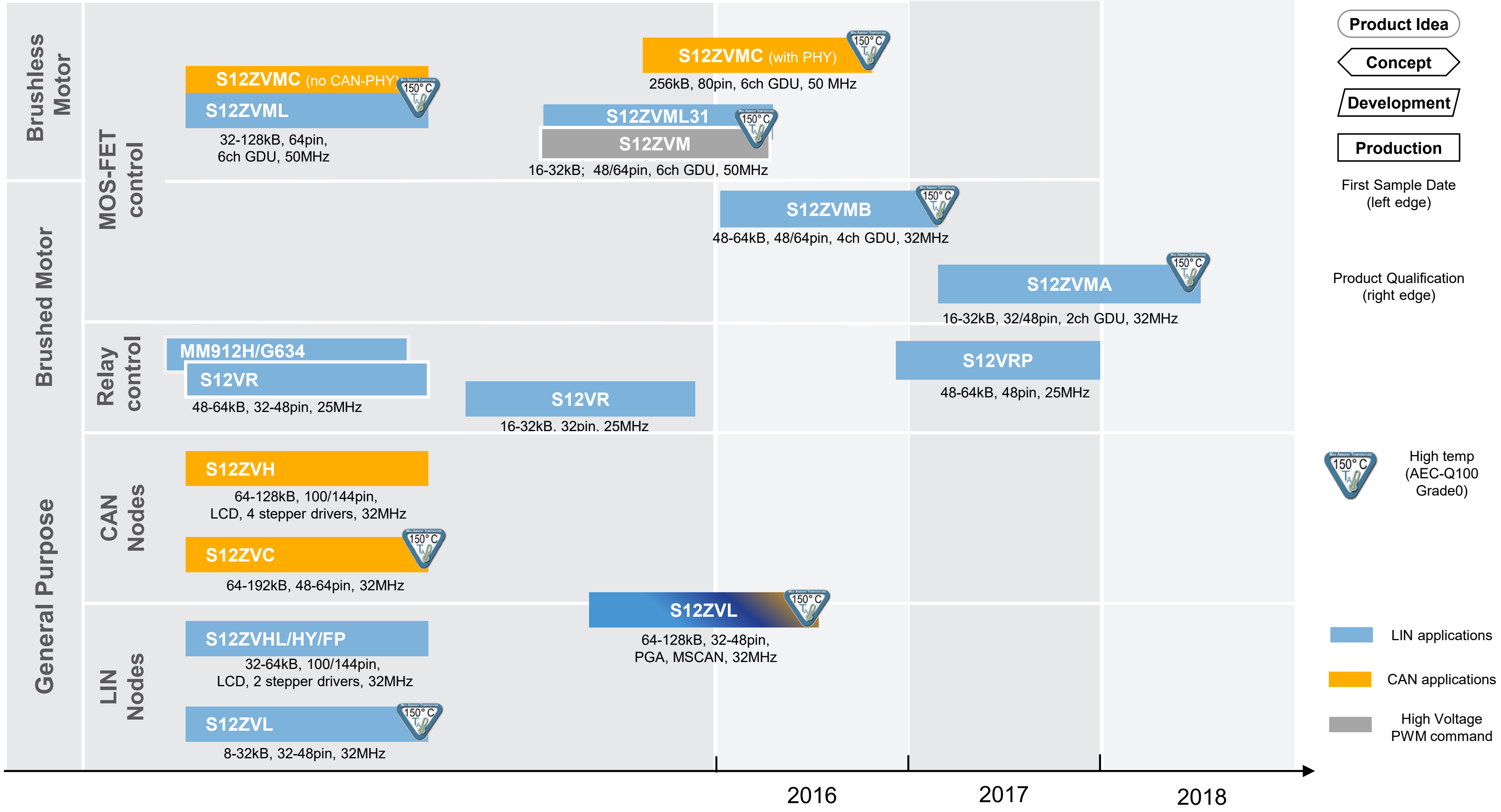
Battery Voltage

3.5 to 40V

S12ZVC benefits:

- Limited PCB-space
- ASIL-requirements
- High resolution timers and DMA enabled LADC
- On-chip analog comparator and DAC
- EVDD 5V switchable sensor supply

MagniV Integrated Solutions Roadmap



General Business Use

S12 MagniV Software

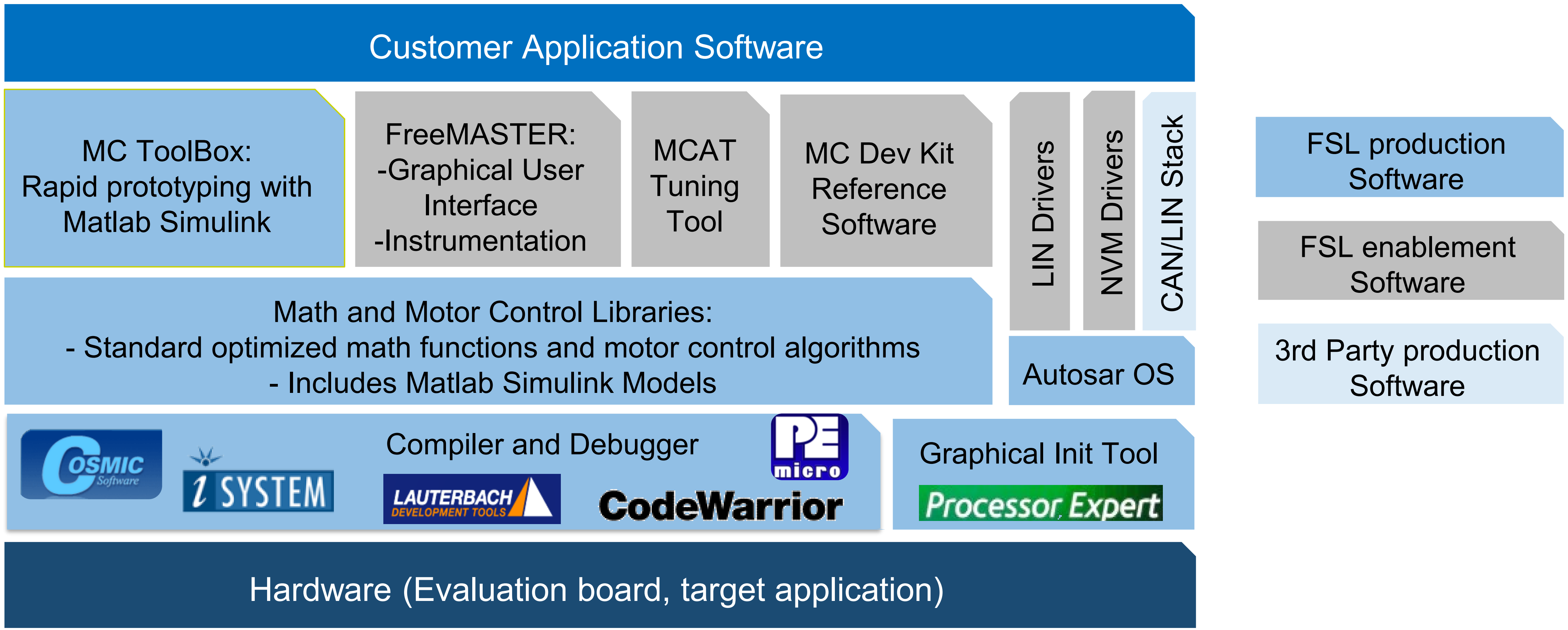
Type	Software Package	Availability	Price Model	Device Support
Software Development Tool	Cosmic Dev Tool	Available	Paid	VR,ZVL,ZVC, ZVM
	CodeWarrior Dev Tool	Available	Paid	VR,ZVL,ZVC, ZVM
	Processor Expert- configuration tool and low level drivers	Available	Free	VR,ZVL,ZVC, ZVM
	Model Based Development Toolbox for MATLAB®/Simulink®	Available	Free	ZVC, ZVM
Runtime Software	S12Z NVM Standard Software Driver for flash module	Available	Free	VR,ZVL,ZVC, ZVM
	LIN Stack- full implementation of LIN2.x and SAE J2602	Available	Free	VR,ZVL,ZVC, ZVM
	Core Self Test- ASIL-A support	Available	Paid	VR,ZVL,ZVC, ZVM
	Core Self Test- ASIL-B support >90% cov.	Available	Paid	VR,ZVL,ZVC, ZVM
	Bootloader	Available	Free	VR,ZVL,ZVC, ZVM
	AMMCLIB- Automotive Math and Motor Control Lib	Available	Free Evaluation Paid Production	ZVM
	Autosar MCAL 4.0 - S12ZVM256	Available	Paid	ZVM256
	Autosar MCAL 4.0 - S12ZVC192	Available	Paid	ZVC
	Autosar OS 4.0.80 Release to Market	Available	Paid	ZVM128
	Autosar OS 4.0.80 Patch for ZVC	Available	Free	ZVC

Preliminary - Subject to change

Last Updated 24APR17

02. DEVKIT-S12ZVC Development Board

MagniV Ecosystem – The Complete Solution



Customer Application Software

MC ToolBox:
Rapid prototyping with
Matlab Simulink

FreeMASTER:
-Graphical User
Interface
-Instrumentation

MCAT
Tuning
Tool

MC Dev Kit
Reference
Software

LIN Drivers

NVM Drivers

CAN/LIN Stack

Math and Motor Control Libraries:
- Standard optimized math functions and motor control algorithms
- Includes Matlab Simulink Models

Autosar OS

Compiler and Debugger

Graphical Init Tool

Hardware (Evaluation board, target application)

FSL production Software

FSL enablement Software

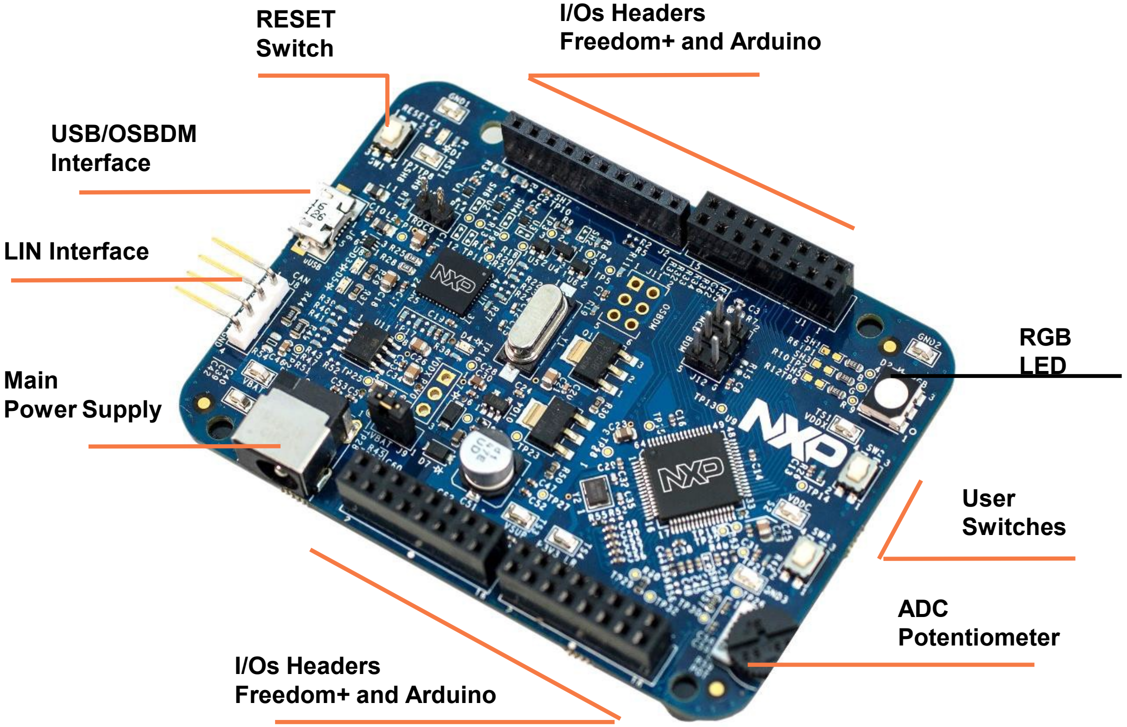
3rd Party production Software

Get to Know the DEVKIT-S12ZVC

http://www.nxp.com/devkit

The DEVKIT-S12ZVC is an ultra-low-cost development platform for S12Z microcontrollers.

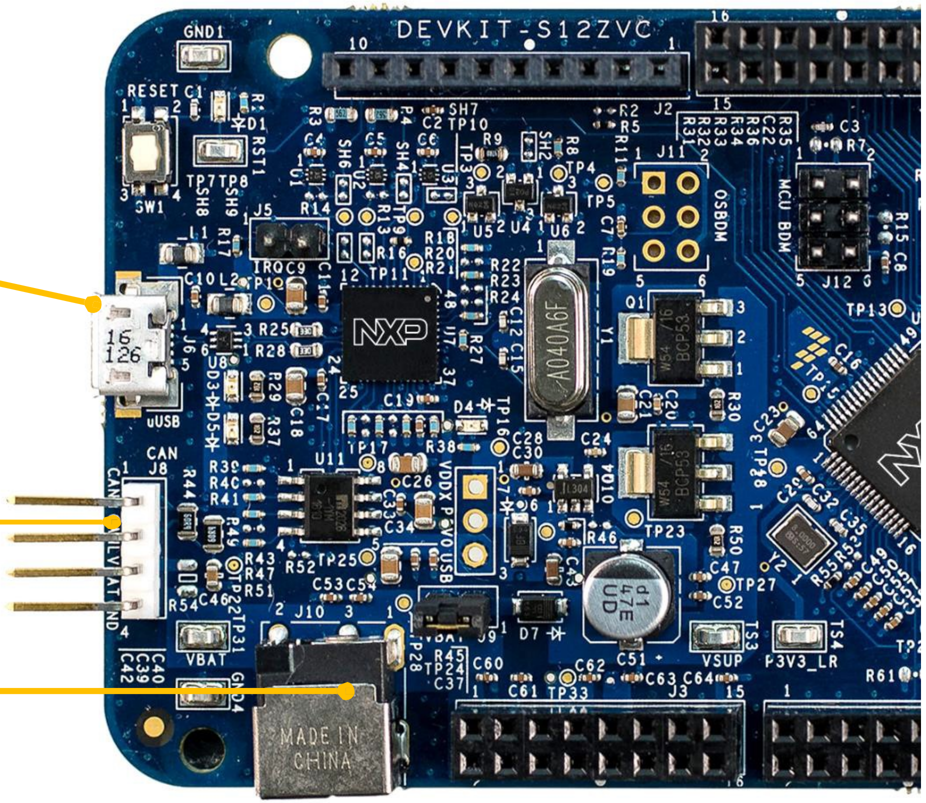
Features include easy access to all MCU I/O´s, a standard- based form factor compatible with the Arduino™ pin layout, providing a broad range of expansion board options, and an USB serial port interface for connection to the IDE, the board has option to be powered via USB or an external power supply.



DEVKIT-S12ZVC
Features

Power Supply and Communications

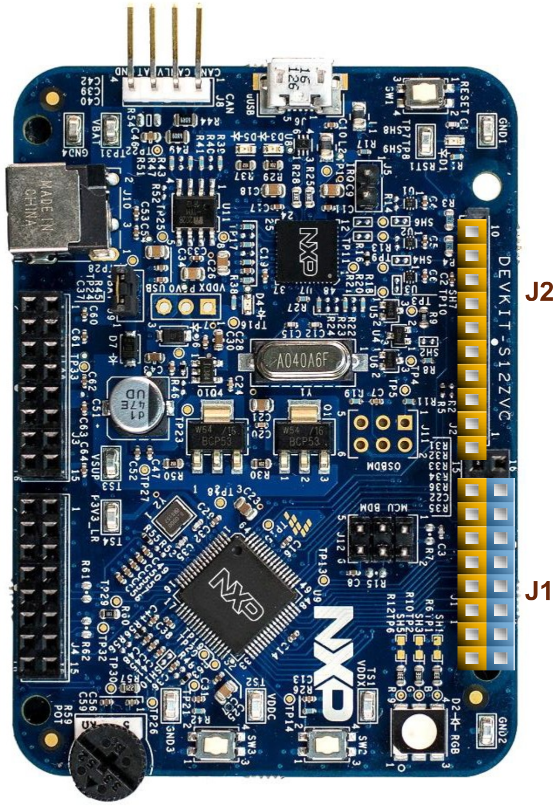
USB/OSBDM Connector
CON 1X5 USB_MICRO_AB_RECEPTACLE
RA SMT 0.65MM SP 105H AU



Description	Name	PIN
	CANH	**J8-01**
	CANL	**J8-02**
	VBAT	**J8-03**
	GND	**J8-04**

Description	Name	PIN
	VBAT	**J10-01**
	GND	**J10-03**

Input/Output Connectors



Arduino Compatibility

The internal rows of the I/O headers on the DEVKIT-S12ZVC are arranged to fulfill Arduino™ shields compatibility .

FUNCTION	PORT	PIN
PWM	PT7	**J2-01**
PWM	PP7	**J2-02**
SPISS PWM	PS3	**J2-03**
SPIMOSI PWM	PS1	**J2-04**
SPIMISO	PS0	**J2-05**
SPISCK	PS2	**J2-06**
	GND	**J2-07**
	AREF	**J2-08**
SDA	PJ1	**J2-09**
SCL	PJ0	**J2-10**

FUNCTION	PORT	PIN	PIN	PORT	FUNCTION
RXD	PS4	**J1-01**	**J1-02**	PT2	GPIO
TXD	PS5	**J1-03**	**J1-04**	PT3	GPIO
PWM	PP0	**J1-05**	**J1-06**	PT6	GPIO
PWM	PP1	**J1-07**	**J1-08**	PT4	GPIO
PWM	PP2	**J1-09**	**J1-10**	PT5	GPIO
PWM	PP3	**J1-11**	**J1-12**		
PWM	PP4	**J1-13**	**J1-14**		
PWM	PP5	**J1-15**	**J1-16**		

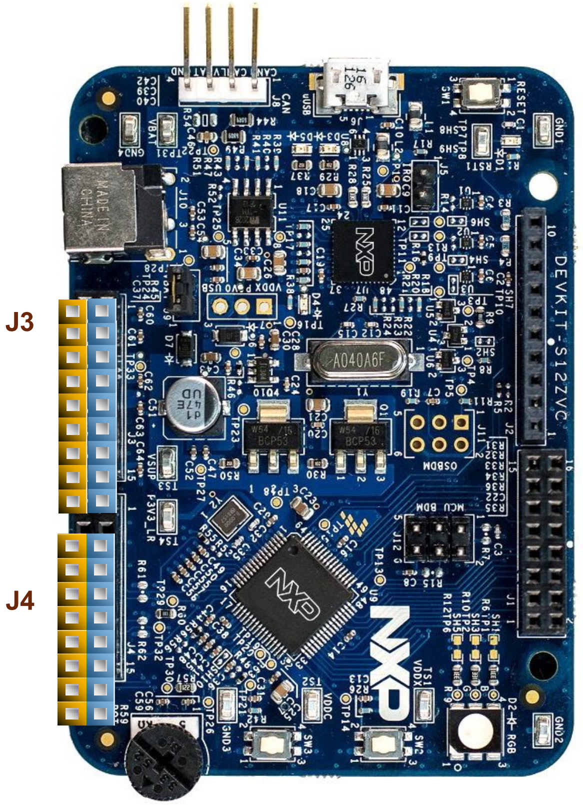
Input/Output Connectors

FUNCTION	PORT	PIN
		J3-02
		J3-04
	PDA10	J3-06
	PDA11	J3-08
	PDA12	J3-10
	PDA13	J3-12
	PDA14	J3-14
	PDA15	J3-16

FUNCTION	PORT	PIN
VIN		J3-01
VDD		J3-02
RESET		J3-03
P3V3		J3-04
P5V0		J3-05
GND		J3-06
GND		J3-07
VIN		J3-08

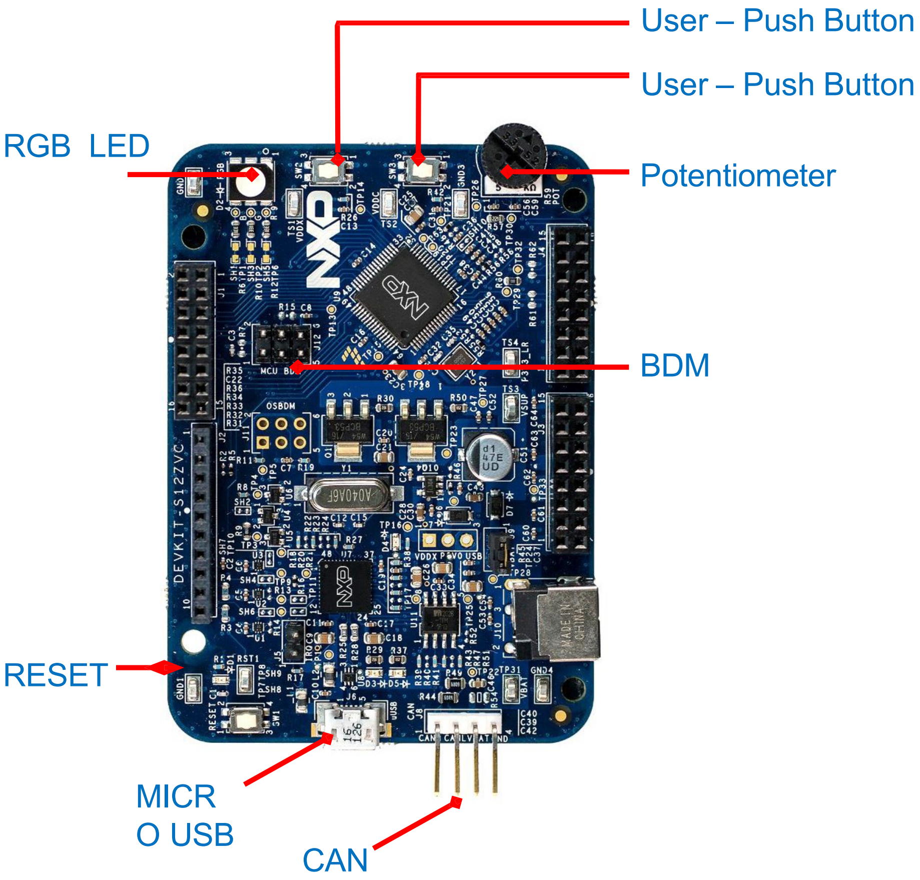
FUNCTION	PORT	PIN
ADC	AN7	J4-01
ADC	AN6	J4-03
ADC	AN5	J4-05
ADC	AN4	J4-07
ADC	AN3	J4-09
ADC	AN2	J4-11
ADC	AN1	J4-13
ADC	AN0	J4-15

PIN	PORT	FUNCTION
J4-02		
J4-04		
J4-06	PS6	
J4-08	PS7	
J4-10	PL1	
J4-12	PL0	
J4-14	AN9	
J4-16	AN8	



Arduino Compatibility
The internal rows of the I/O headers on the DEVKIT-S12VC are arranged to fulfill Arduino™ shields compatibility .

Programming Interface and User Peripherals



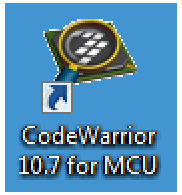
Peripheral	ID	MCU Port	Description
Buttons	SW2	PAD11	User switch (Active high)
	SW3	PAD10	User switch (Active high)
	SW1	RESET	RESET Switch
Potentiometers	R59	AN1	Potentiometer connected to ADC port AN0/AN1
LED	D2	PP4	RGB LED - Green
		PP5	RGB LED - Red
		PP6	RGB LED - Blue
	D3	-	OSBDM PWR LED, ON when OSBDM is successfully enumerated as USB device.
	D5	-	OSBDM STATUS LED. ON when OSBDM is successfully transmitting as USB device.
	D4	VDDX	MCU Power LED Indicator. ON when VDDX is regulating to +5V
	D1	RESET	RESET LED Indicator
Communication	J6	-	OSBDM USB
	J8	CAN	CAN Interface

Caution: When powered from the USB bus, do not exceed the 500mA maximum allowable current drain. Damage to the target board or host PC may result.

03. Set Up the Development Environment

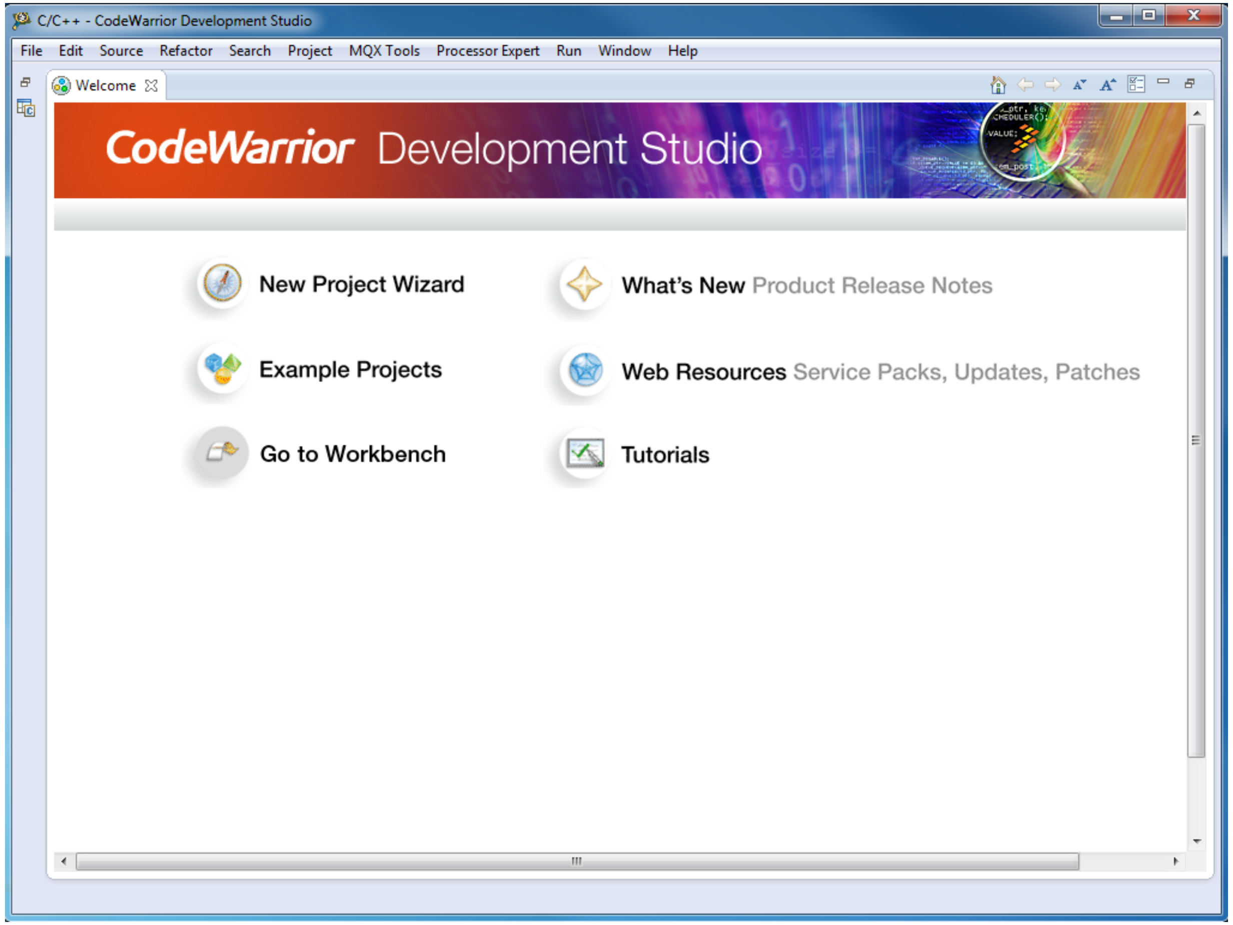
CodeWarrior v10.7 – Startup

Launch CodeWarrior

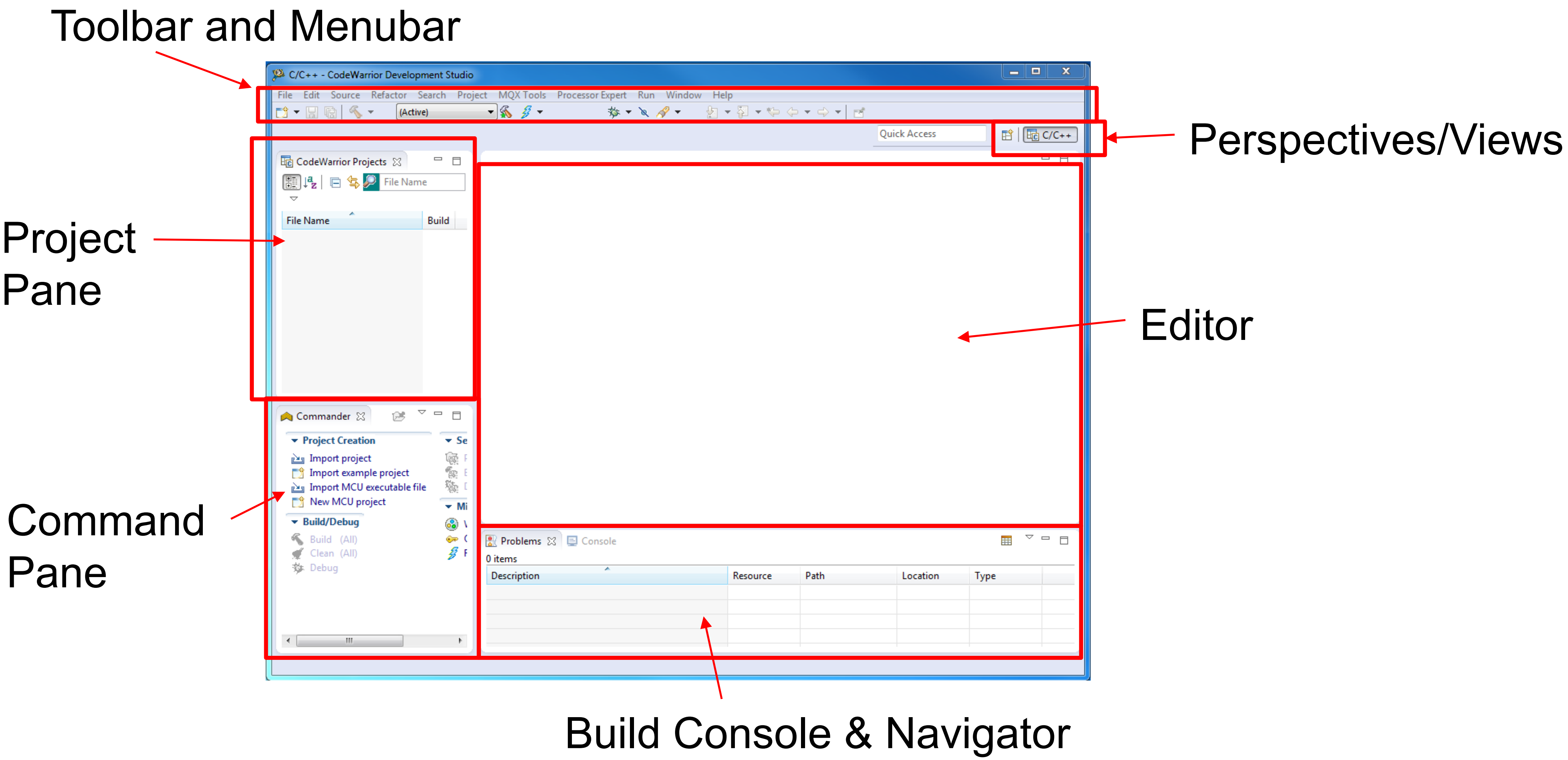


View Welcome screen

- Close it, or just wait 30seconds
- Processor Expert starts in background


C/C++ - CodeWarrior Development Studio
File
Edit
Source
Refactor
Search
Project
MQX Tools
Processor Expert
Run
Window
Help
Welcome
CodeWarrior Development Studio
New Project Wizard
What's New Product Release Notes
Example Projects
Web Resources Service Packs, Updates, Patches
Go to Workbench
Tutorials

Default / Empty View



Toolbar and Menubar

Perspectives/Views

Project Pane

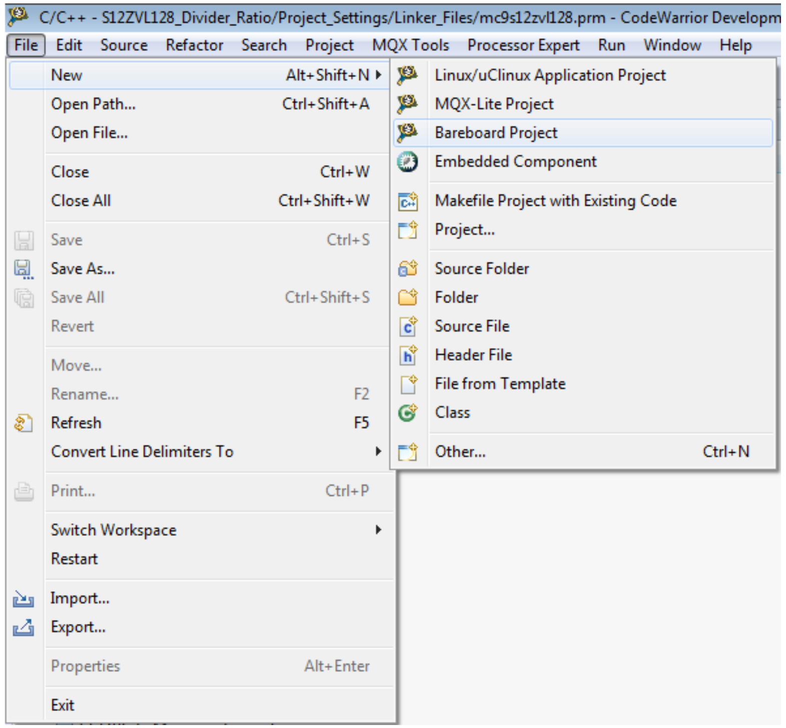
Editor

Command Pane

Build Console & Navigator

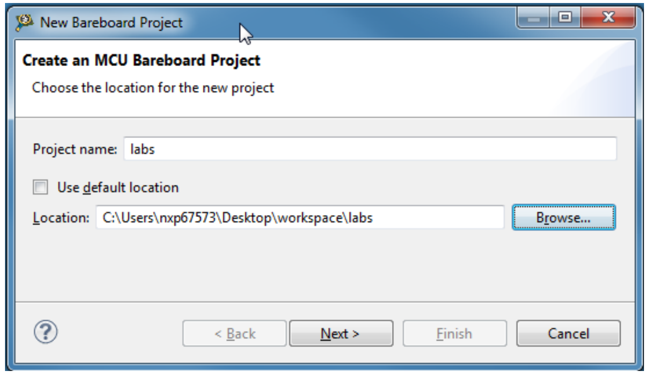
Create Bareboard Project (1)

1) Go to file → New → Bareboard Project



2) Make a name for your Project (e.g. “labs”)

3) Create a workspace folder→ click next


New Bareboard Project

Create an MCU Bareboard Project

Choose the location for the new project

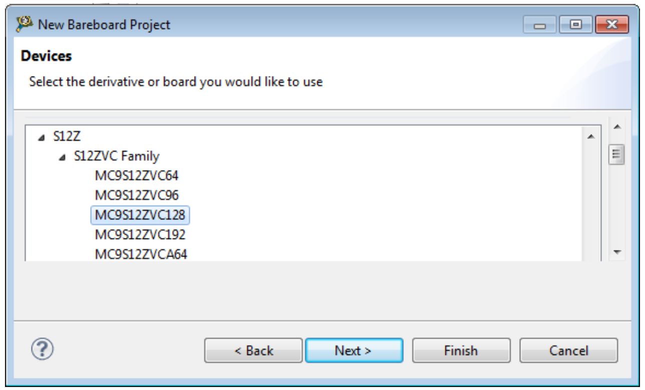
Project name: labs

☐ Use default location

Location: C:\Users\nxp67573\Desktop\workspace\labs

4) Select your device

S12Z → MC9S12ZVCA128 → click next


New Bareboard Project

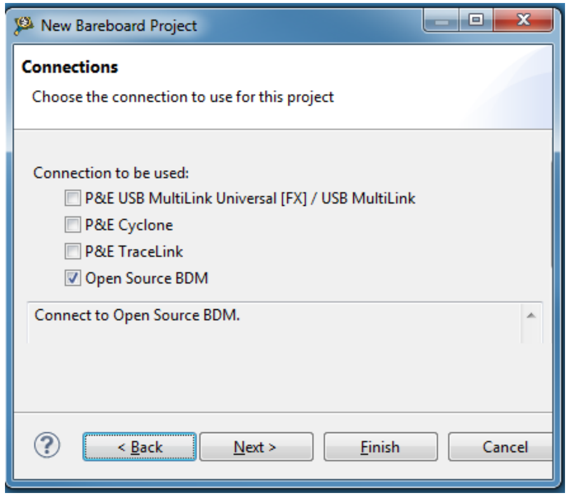
Devices

Select the derivative or board you would like to use

- S12Z
  - S12ZVC Family
    - MC9S12ZVC64
    - MC9S12ZVC96
    - MC9S12ZVC128
    - MC9S12ZVC192
    - MC9S12ZVCA64

Create Bareboard Project (2)

5) Select OSBDM connection → click next



Connections

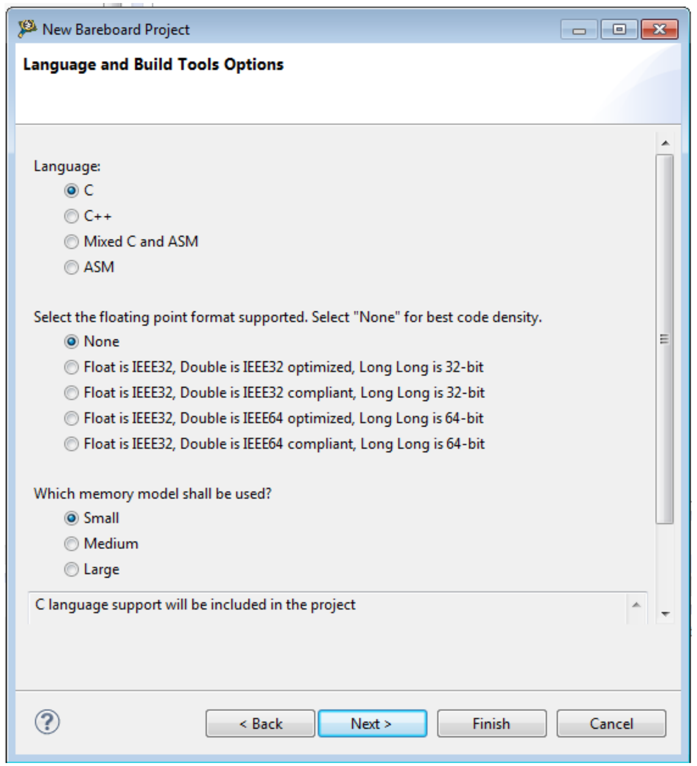
Choose the connection to use for this project

Connection to be used:

- □ P&E USB MultiLink Universal [FX] / USB MultiLink
- □ P&E Cyclone
- □ P&E TraceLink
- ☑ Open Source BDM

Connect to Open Source BDM.

6) Use defaults → click next



Language and Build Tools Options

Language:

- ● C
- ○ C++
- ○ Mixed C and ASM
- ○ ASM

Select the floating point format supported. Select "None" for best code density.

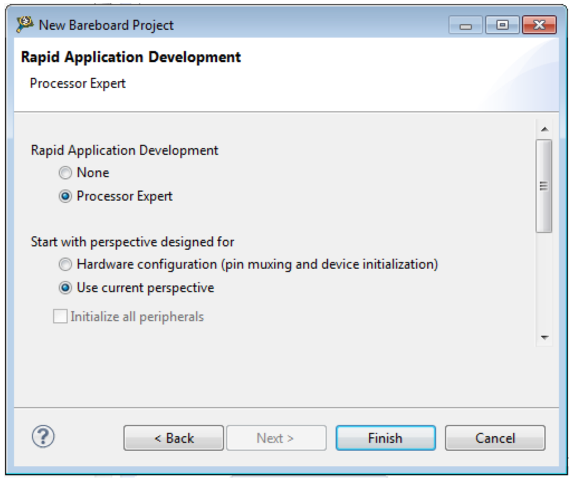
- ● None
- ○ Float is IEEE32, Double is IEEE32 optimized, Long Long is 32-bit
- ○ Float is IEEE32, Double is IEEE32 compliant, Long Long is 32-bit
- ○ Float is IEEE32, Double is IEEE64 optimized, Long Long is 64-bit
- ○ Float is IEEE32, Double is IEEE64 compliant, Long Long is 64-bit

Which memory model shall be used?

- ● Small
- ○ Medium
- ○ Large

C language support will be included in the project

7) Select Processor Expert → click Finish



Rapid Application Development

Processor Expert

Rapid Application Development

- ○ None
- ● Processor Expert

Start with perspective designed for

- ○ Hardware configuration (pin muxing and device initialization)
- ● Use current perspective

□ Initialize all peripherals

Project View

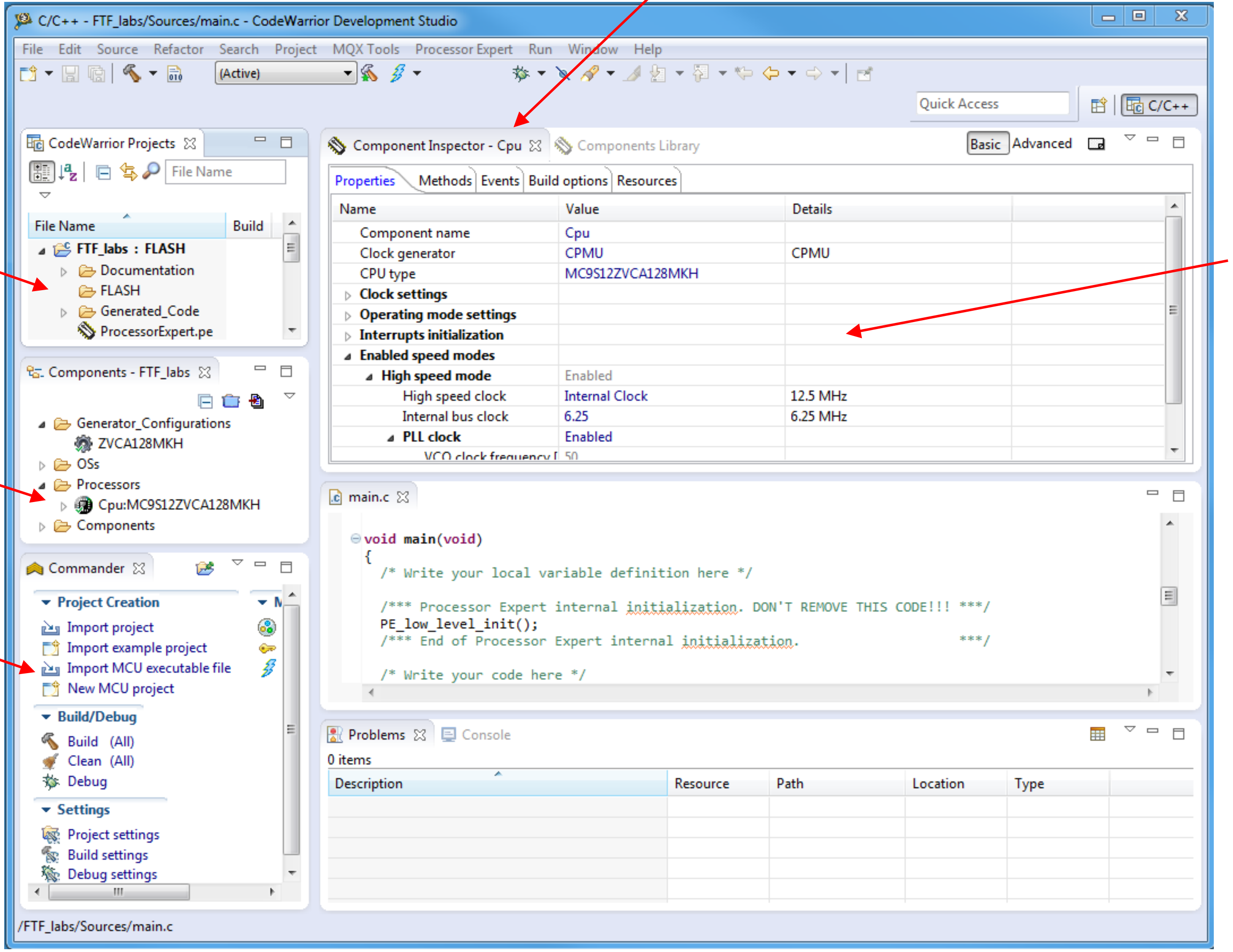
Processor Expert Configuration Pane

Project Files

PEx Components

Project Controls & Commands

PEx Module Parameters



CodeWarrior Projects

- FTF_labs : FLASH
  - Documentation
  - FLASH
  - Generated_Code
  - ProcessorExpert.pe

Components - FTF_labs

- Generator_Configurations
  - ZVCA128MKH
- OSs
- Processors
  - Cpu:MC9S12ZVCA128MKH
- Components

Commander

- Project Creation
  - Import project
  - Import example project
  - Import MCU executable file
  - New MCU project
- Build/Debug
  - Build (All)
  - Clean (All)
  - Debug
- Settings
  - Project settings
  - Build settings
  - Debug settings

Component Inspector - Cpu

Name	Value	Details
Component name	Cpu	
Clock generator	CPMU	CPMU
CPU type	MC9S12ZVCA128MKH	
Clock settings		
Operating mode settings		
Interrupts initialization		
Enabled speed modes		
High speed mode	Enabled	
High speed clock	Internal Clock	12.5 MHz
Internal bus clock	6.25	6.25 MHz
PLL clock	Enabled	
VCO clock frequency [	50	

main.c

```c
void main(void)
{
  /* Write your local variable definition here */

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  /* Write your code here */
```

Problems

0 items

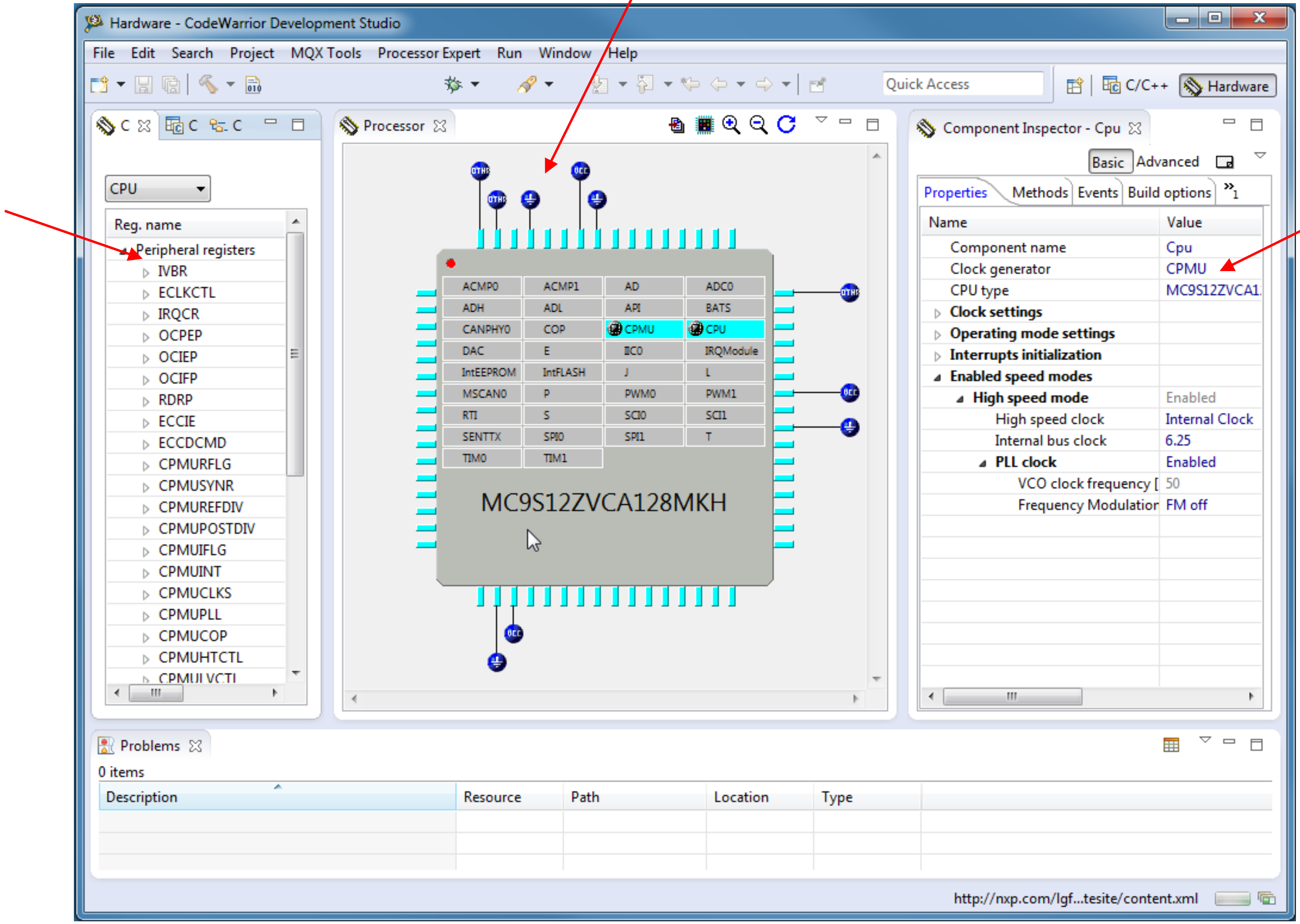
Description	Resource	Path	Location	Type

Hardware View

Pin Muxing

Register List

Configurations



CPU

Reg. name

- Peripheral registers
  - IVBR
  - ECLKCTL
  - IRQCR
  - OCPEP
  - OCIEP
  - OCIFP
  - RDRP
  - ECCIE
  - ECCDCMD
  - CPMURFLG
  - CPMUSYNR
  - CPMUREFDIV
  - CPMUPOSTDIV
  - CPMUIFLG
  - CPMUINT
  - CPMUCLKS
  - CPMUPLL
  - CPMUCOP
  - CPMUHTCTL
  - CPMULVCTL

ACMP0	ACMP1	AD	ADC0
ADH	ADL	API	BATS
CANPHY0	COP	CPMU	CPU
DAC	E	IIC0	IRQModule
IntEEPROM	IntFLASH	J	L
MSCAN0	P	PWM0	PWM1
RTI	S	SCI0	SCI1
SENTTX	SPI0	SPI1	T
TIM0	TIM1		

MC9S12ZVCA128MKH

Properties | Methods | Events | Build options | »1

Name	Value
Component name	Cpu
Clock generator	CPMU
CPU type	MC9S12ZVCA1:
▷ **Clock settings**	
▷ **Operating mode settings**	
▷ **Interrupts initialization**	
◢ **Enabled speed modes**	
◢ **High speed mode**	Enabled
High speed clock	Internal Clock
Internal bus clock	6.25
◢ **PLL clock**	Enabled
VCO clock frequency [	50
Frequency Modulatior	FM off

0 items

Description	Resource	Path	Location	Type

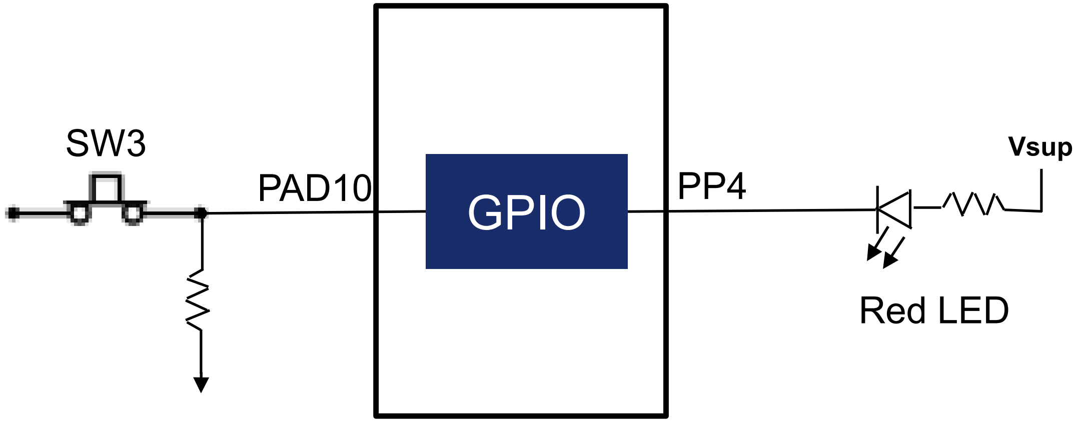
http://nxp.com/lgf...tesite/content.xml

04. Hands-on LABS

Hello World (GPIO)

Hello World: Introduction

Summary: A GPIO input (PAD10) is continuously polled to detect a high or low level. A GPIO output (PP4) is set corresponding to the level and drives the LED.



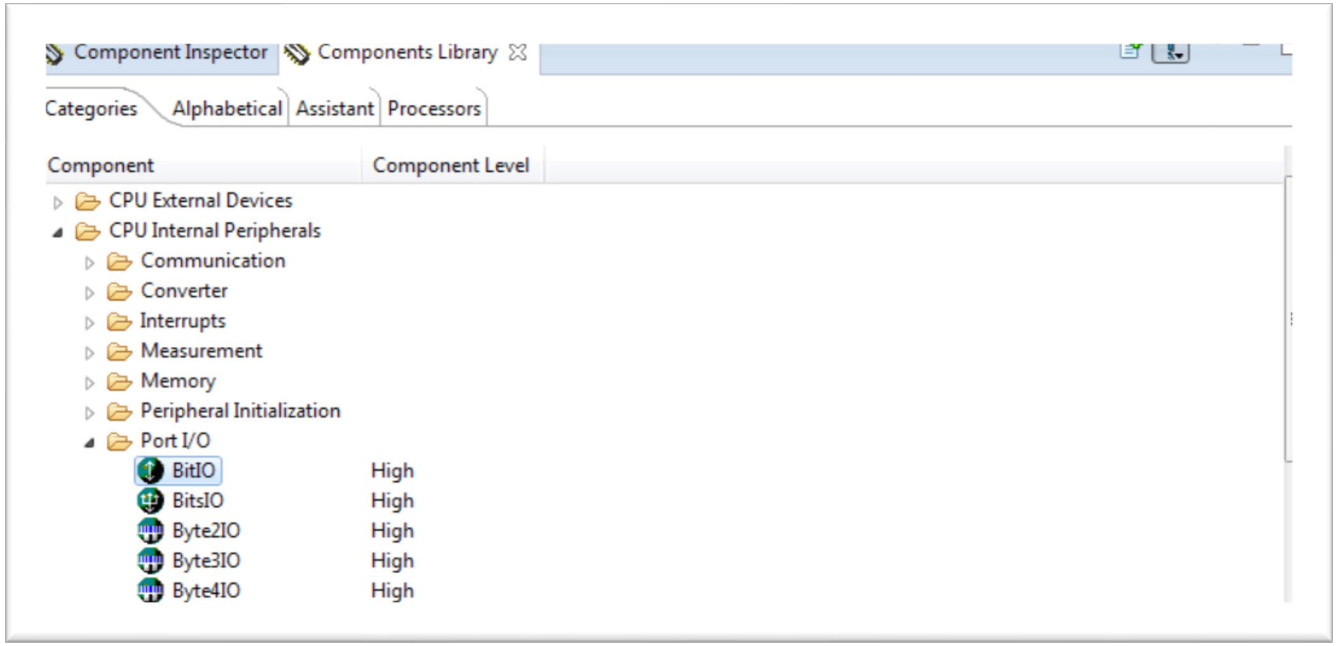
Hello World: Key Points

- Out of reset, default clocks and GPIO are enabled
- GPIO requires configuring
  - Input or output direction
  - GPIO function

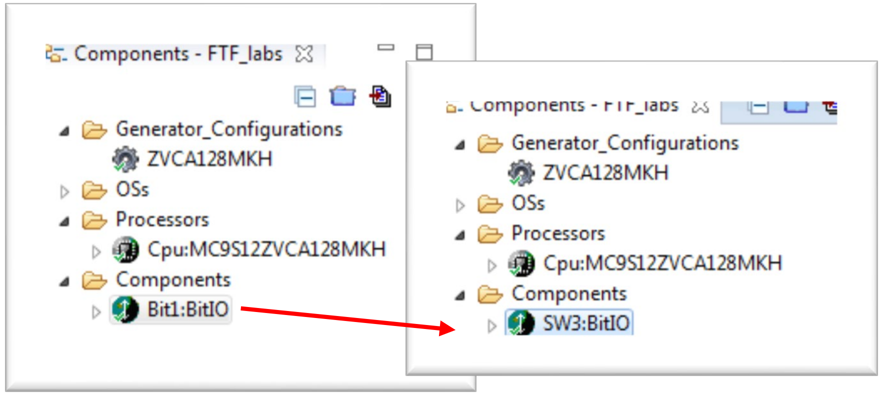
- Hands on: Using next slides:
  - Create GPIO components using PEx
  - Generate PEx initialization code, and create main routine
  - Build and debug

Create SW3 Input

1) Component Library
2) CPU Internal Peripherals
3) Port I/O
4) Double-click “BitIO”

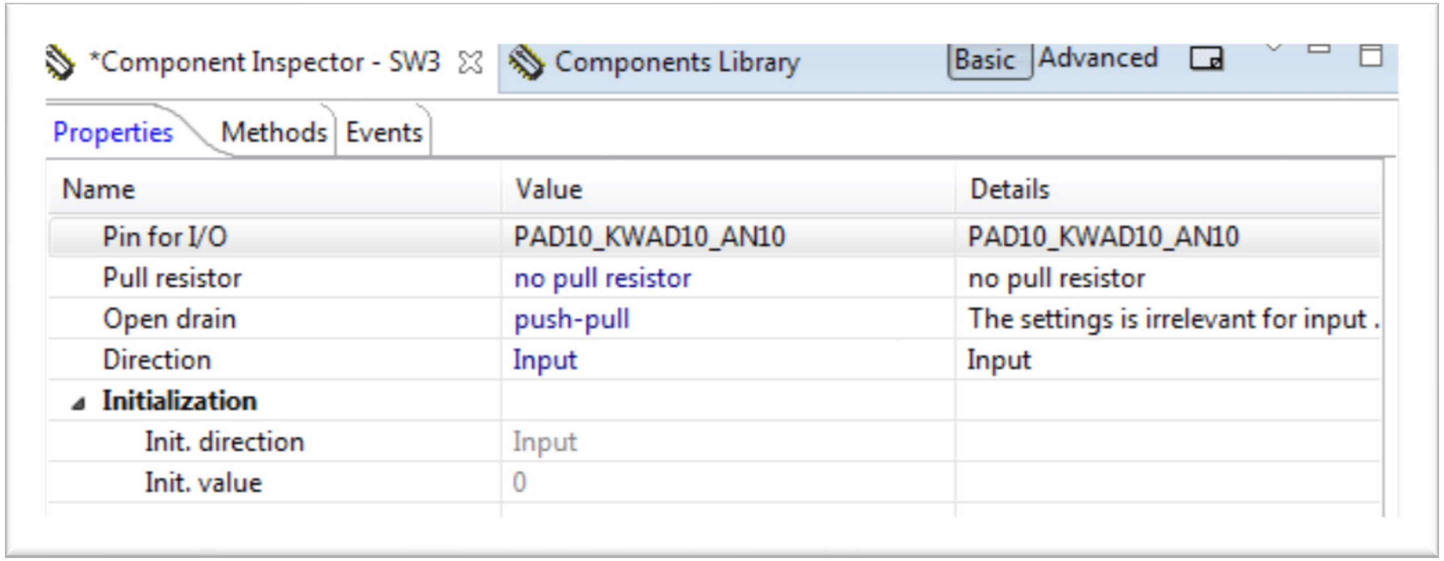


5) Move to PEX component pane
6) Right-click “Bit1:BitIO” and rename it to “SW3”
7) Double-click “SW3” to edit its configuration



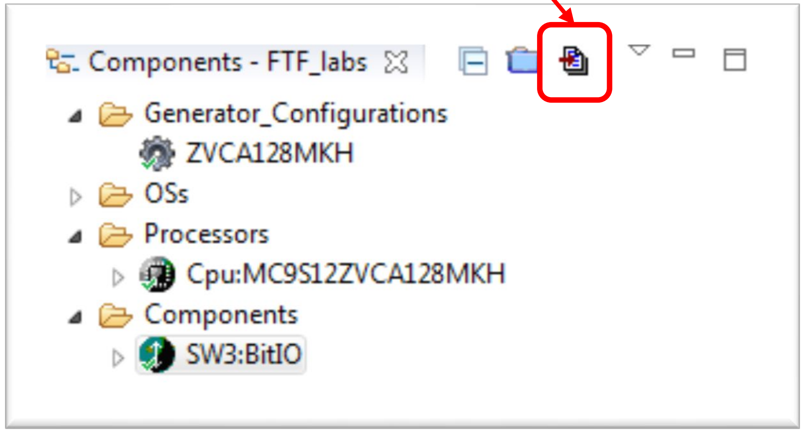
Configure SW3

1) Choose the configurations to make this pin an input with no pull resistor.

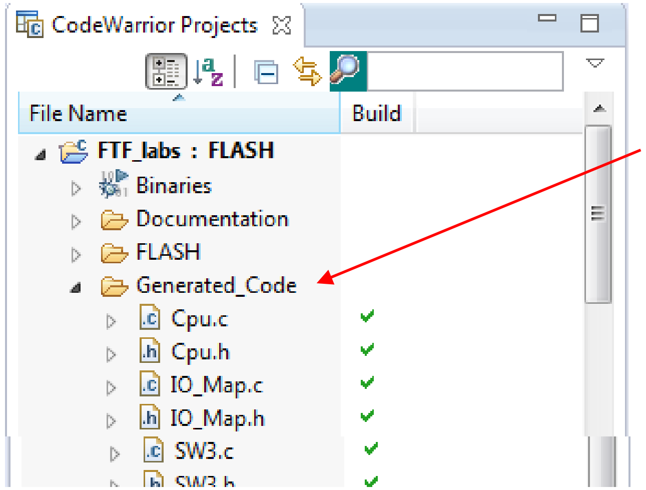


Name	Value	Details
Pin for I/O	PAD10_KWAD10_AN10	PAD10_KWAD10_AN10
Pull resistor	no pull resistor	no pull resistor
Open drain	push-pull	The settings is irrelevant for input .
Direction	Input	Input
Initialization		
Init. direction	Input	
Init. value	0	

2) Generate code.



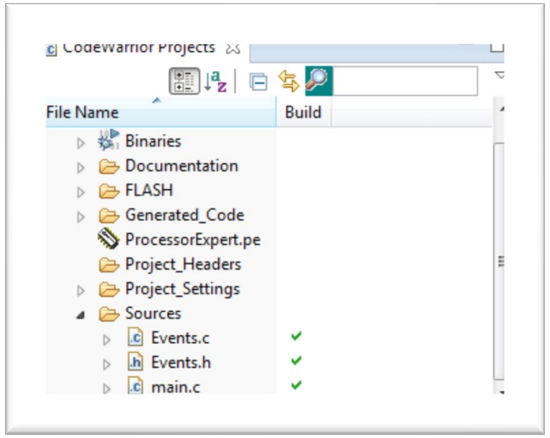
- Generator_Configurations
  - ZVCA128MKH
- OSs
- Processors
  - Cpu:MC9S12ZVCA128MKH
- Components
  - SW3:BitIO



- FTF_labs : FLASH
  - Binaries
  - Documentation
  - FLASH
  - Generated_Code
    - Cpu.c ✓
    - Cpu.h ✓
    - IO_Map.c ✓
    - IO_Map.h ✓
    - SW3.c ✓
    - SW3.h ✓

3) Open “Generated_Code” in the project tree to see what was made by PEX.

Inspect the Generated Code

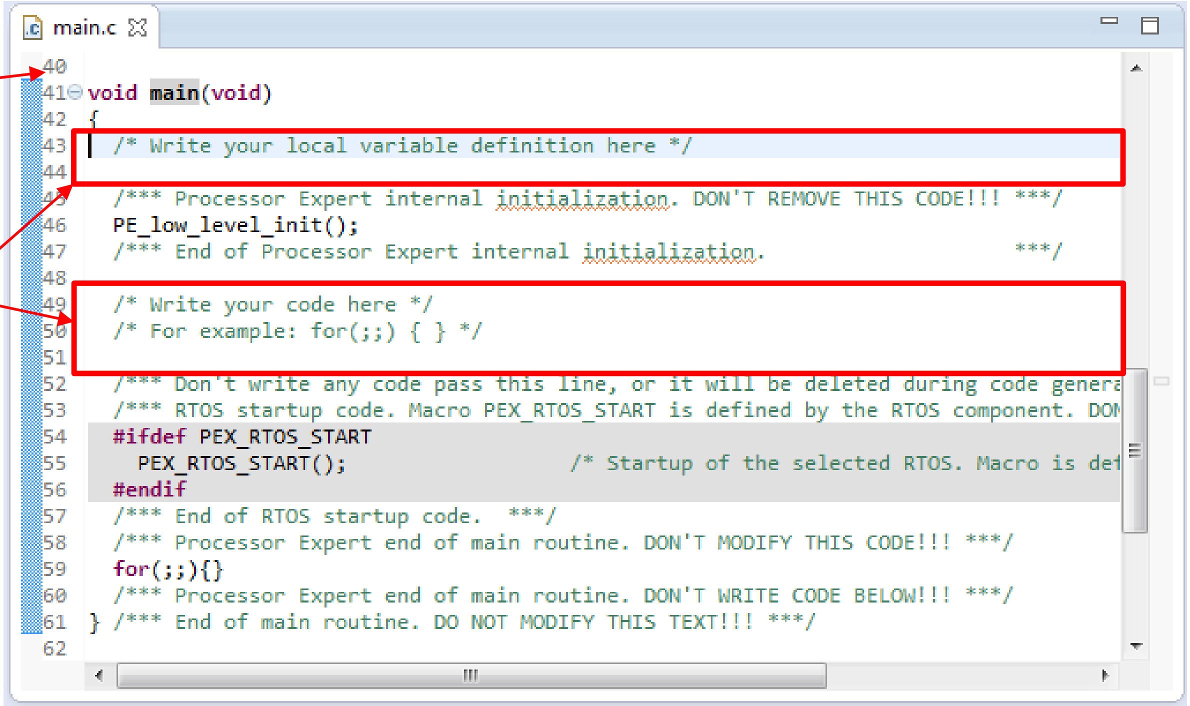


1) Double-click “main.c” to edit

2) In the left margin of the editor window, right-click and choose “show line numbers”

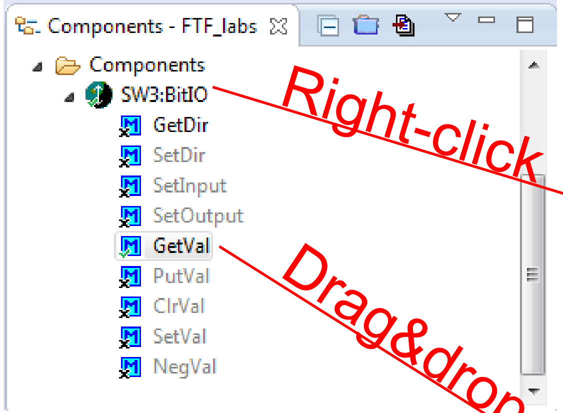
3) Be careful where you add user code!

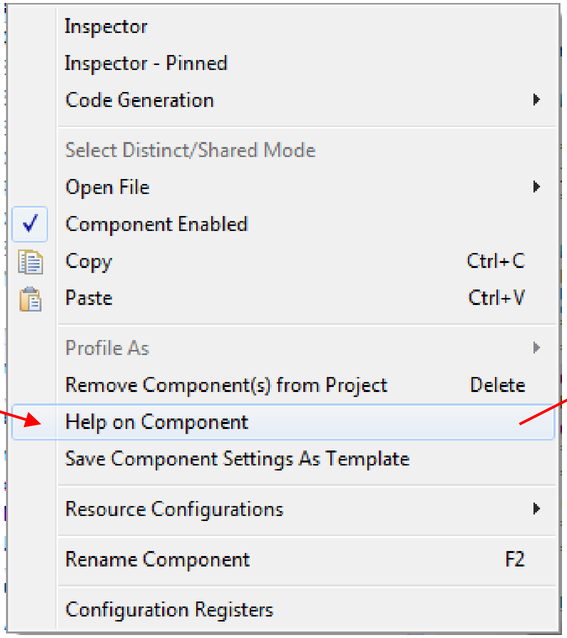
4) Right-click “PE_low_level_init()” and choose “open declaration” to see the code generated by PEX.

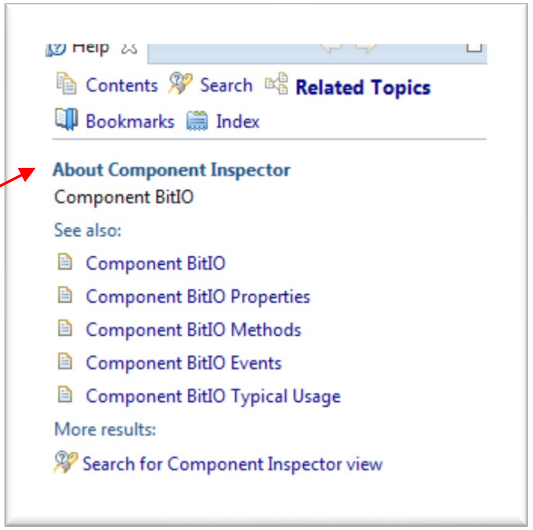


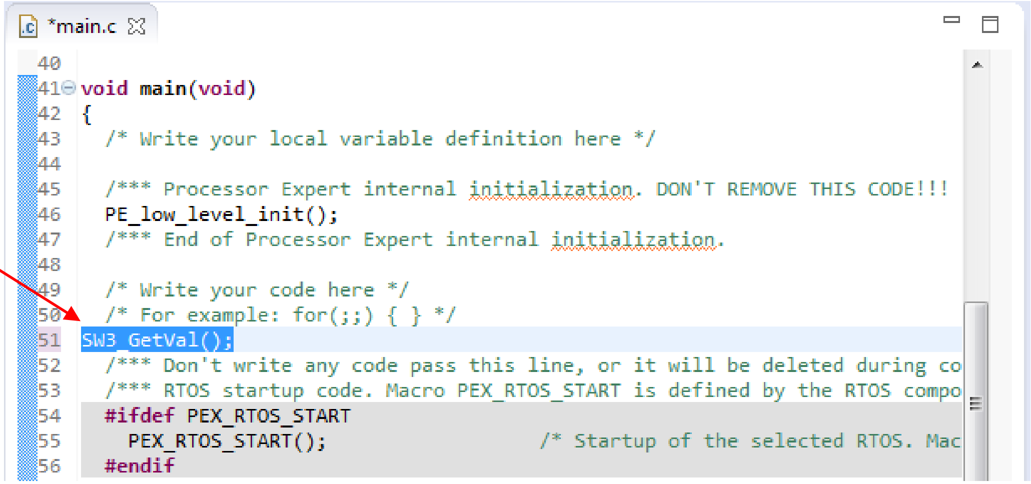
```
void main(void)
{
  /* Write your local variable definition here */

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  /* Write your code here */
  /* For example: for(;;) { } */

  /*** Don't write any code pass this line, or it will be deleted during code genera
  /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component. DON
  #ifdef PEX_RTOS_START
    PEX_RTOS_START();                  /* Startup of the selected RTOS. Macro is def
  #endif
  /*** End of RTOS startup code.  ***/
  /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
  for(;;){}
  /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
} /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/
```

PEX Software API’s









```

void main(void)
{
  /* Write your local variable definition here */

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!!
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.

  /* Write your code here */
  /* For example: for(;;) { } */
SW3_GetVal();
  /*** Don't write any code pass this line, or it will be deleted during co
  /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS compo
  #ifdef PEX_RTOS_START
    PEX_RTOS_START();                  /* Startup of the selected RTOS. Mac
  #endif
```

- Open SW3:BitIO component view
- Inspect SW3:BitIO Methods list
  - Why are some enabled/disabled?
- Right-click methods and choose “Help”
- Drag-n-drop “GetVal” into main code

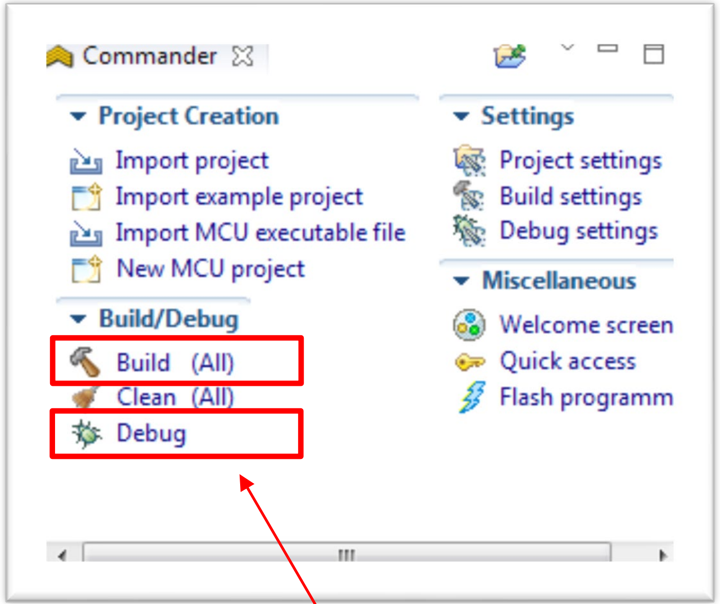
Exercise!

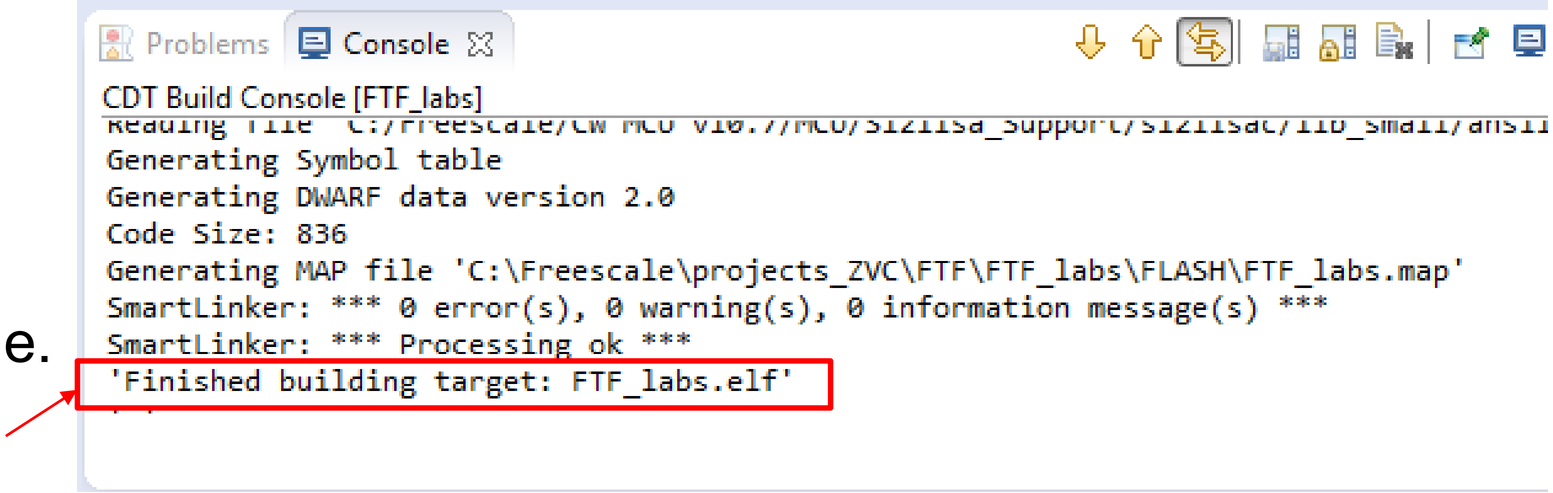
1) Create a new BitIO component
   - Pin: PP4
   - Name: “LedRed”
   - Direction: Output
   - Init. Value: 1
2) Generate PEX code
3) In main():
   - Add a local variable “bool level”
   - In a forever loop, use available BitIO Methods
     - Read SW3 into “level”
     - Put “level” to RedLed
4) Build and debug

Code Solution

```
void main(void)
{
  /* Write your local variable definition here */
bool level;

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                ***/

  /* Write your code here */
  /* For example: for(;;) { } */
  for (;;)
  {
  level = SW3_GetVal();
  LedRed_PutVal(level);
  }
  /*** Don't write any code pass this line, or it will be deleted during code generation. ***/
  /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component. DON'T MODIFY THIS
CODE!!! ***/
  #ifdef PEX_RTOS_START
    PEX_RTOS_START();                  /* Startup of the selected RTOS. Macro is defined by the RTOS component.
*/
  #endif
  /*** End of RTOS startup code.  ***/
  /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
  for(;;){}
  /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
} /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/
```

Build





CDT Build Console [FTF_labs]

```
Reading file 'C:/Freescale/CW MCU v10.7/MCU/S12lisa_Support/s12lisac/lib_small/ansil
Generating Symbol table
Generating DWARF data version 2.0
Code Size: 836
Generating MAP file 'C:\Freescale\projects_ZVC\FTF\FTF_labs\FLASH\FTF_labs.map'
SmartLinker: *** 0 error(s), 0 warning(s), 0 information message(s) ***
SmartLinker: *** Processing ok ***
'Finished building target: FTF_labs.elf'
```

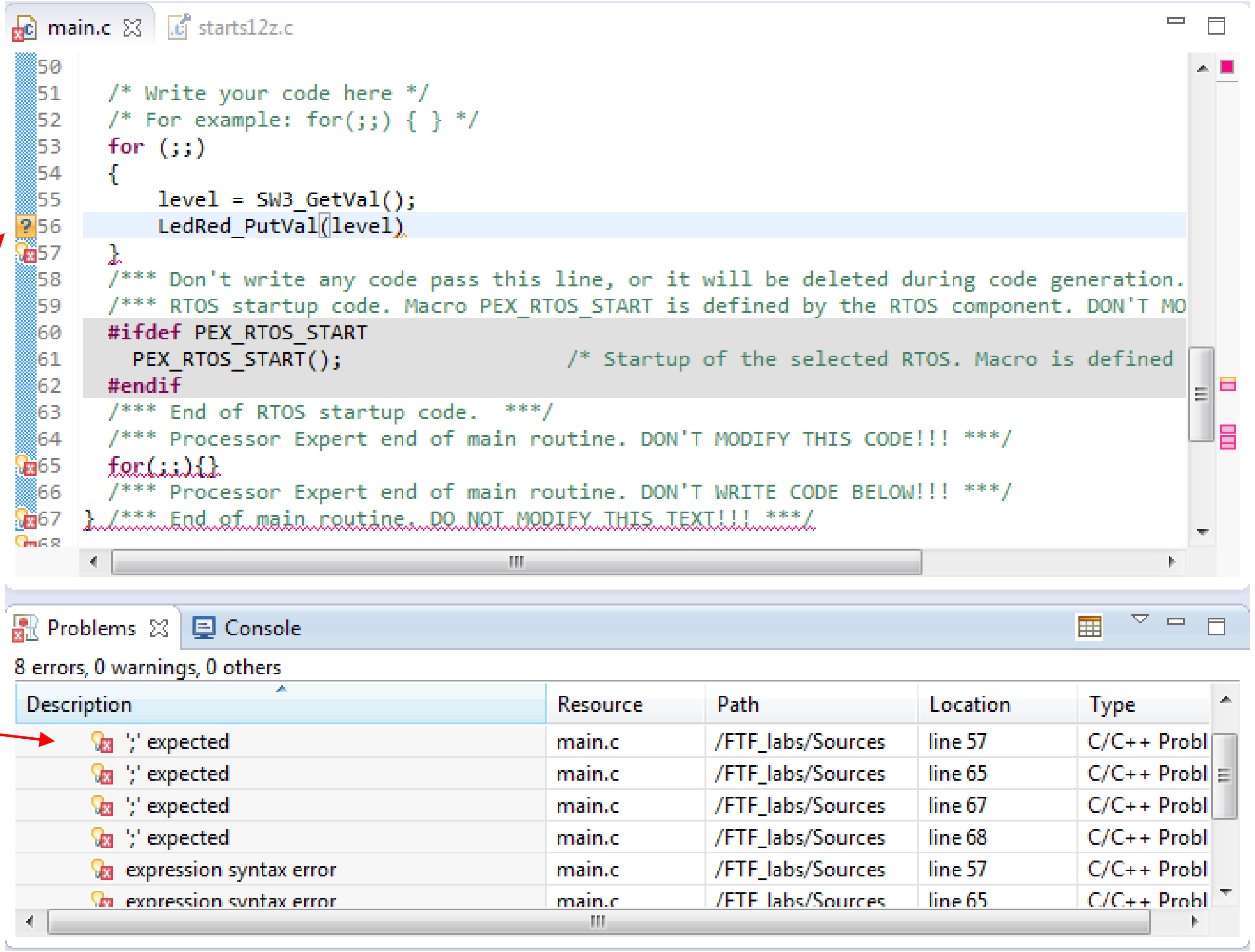
Clicking Build or Debug launches the built-in makefile.

Console shows success/fail status of the session.

Errors Can Be Navigated

Move the editor to the suspicious line of code

by double-clicking the error

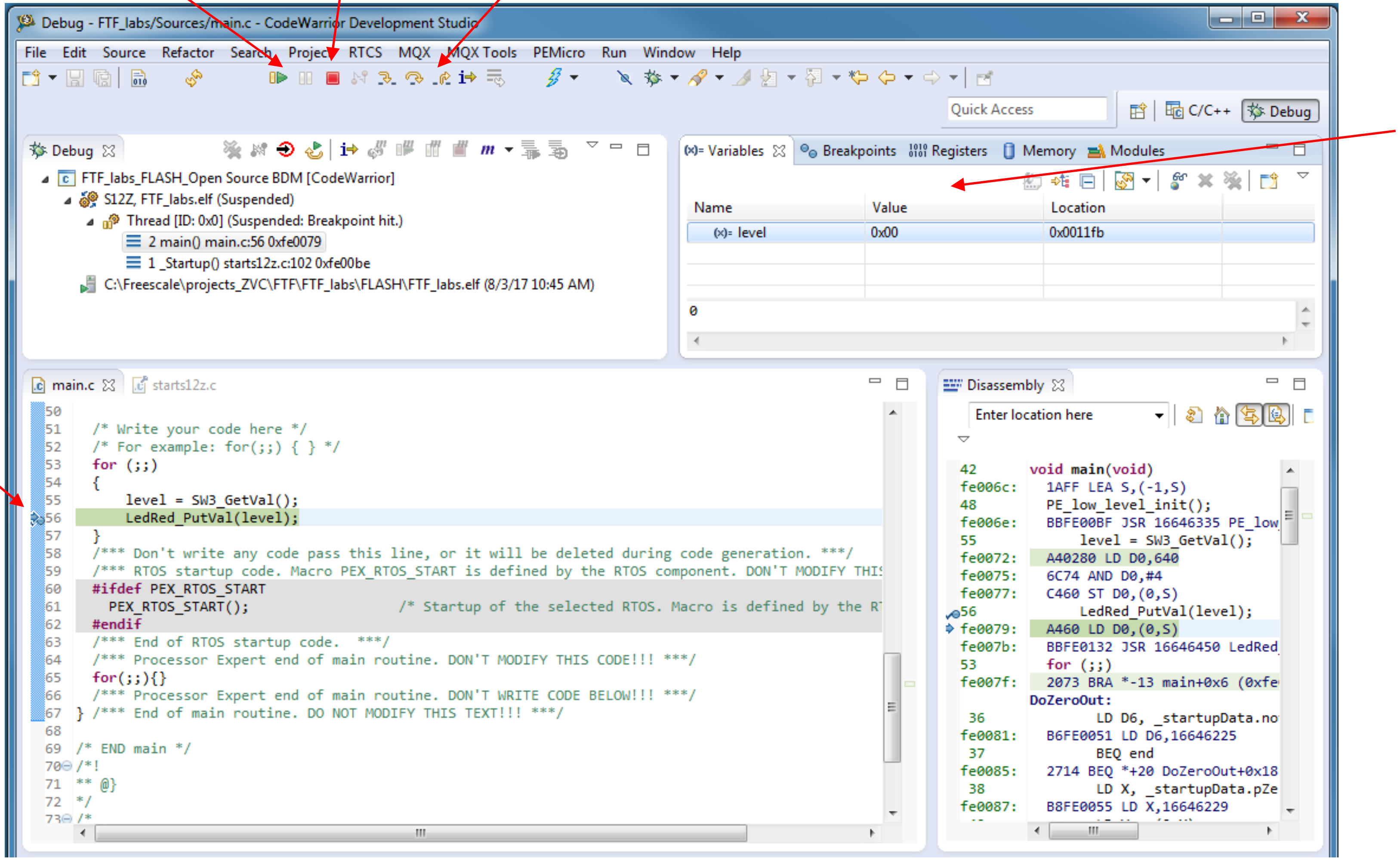


```
50
51    /* Write your code here */
52    /* For example: for(;;) { } */
53    for (;;)
54    {
55        level = SW3_GetVal();
56        LedRed_PutVal(level)
57    }
58    /*** Don't write any code pass this line, or it will be deleted during code generation.
59    /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component. DON'T MO
60    #ifdef PEX_RTOS_START
61      PEX_RTOS_START();                  /* Startup of the selected RTOS. Macro is defined
62    #endif
63    /*** End of RTOS startup code.  ***/
64    /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
65    for(;;){}
66    /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
67  } /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/
```

8 errors, 0 warnings, 0 others

Description	Resource	Path	Location	Type
';' expected	main.c	/FTF_labs/Sources	line 57	C/C++ Probl
';' expected	main.c	/FTF_labs/Sources	line 65	C/C++ Probl
';' expected	main.c	/FTF_labs/Sources	line 67	C/C++ Probl
';' expected	main.c	/FTF_labs/Sources	line 68	C/C++ Probl
expression syntax error	main.c	/FTF_labs/Sources	line 57	C/C++ Probl
expression syntax error	main.c	/FTF_labs/Sources	line 65	C/C++ Probl

Debug



FTF_labs_FLASH_Open Source BDM [CodeWarrior]
S12Z, FTF_labs.elf (Suspended)
Thread [ID: 0x0] (Suspended: Breakpoint hit.)
2 main() main.c:56 0xfe0079
1 _Startup() starts12z.c:102 0xfe00be
C:\Freescale\projects_ZVC\FTF\FTF_labs\FLASH\FTF_labs.elf (8/3/17 10:45 AM)

Name	Value	Location
(x)= level	0x00	0x0011fb

0

```

  /* Write your code here */
  /* For example: for(;;) { } */
  for (;;)
  {
      level = SW3_GetVal();
      LedRed_PutVal(level);
  }
  /*** Don't write any code pass this line, or it will be deleted during code generation. ***/
  /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component. DON'T MODIFY THI
  #ifdef PEX_RTOS_START
    PEX_RTOS_START();                  /* Startup of the selected RTOS. Macro is defined by the R
  #endif
  /*** End of RTOS startup code.  ***/
  /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
  for(;;){}
  /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
} /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/

/* END main */
/*!
** @}
*/
/*
```

```
42         void main(void)
fe006c:      1AFF LEA S,(-1,S)
48           PE_low_level_init();
fe006e:      BBFE00BF JSR 16646335 PE_low
55               level = SW3_GetVal();
fe0072:      A40280 LD D0,640
fe0075:      6C74 AND D0,#4
fe0077:      C460 ST D0,(0,S)
56               LedRed_PutVal(level);
fe0079:      A460 LD D0,(0,S)
fe007b:      BBFE0132 JSR 16646450 LedRed
53           for (;;)
fe007f:      2073 BRA *-13 main+0x6 (0xfe
           DoZeroOut:
 36                 LD D6, _startupData.no
fe0081:      B6FE0051 LD D6,16646225
 37                 BEQ end
fe0085:      2714 BEQ *+20 DoZeroOut+0x18
 38                 LD X, _startupData.pZe
fe0087:      B8FE0055 LD X,16646229
```

End Session

Step/Over/Return/Instruction-mode

Resume/Pause

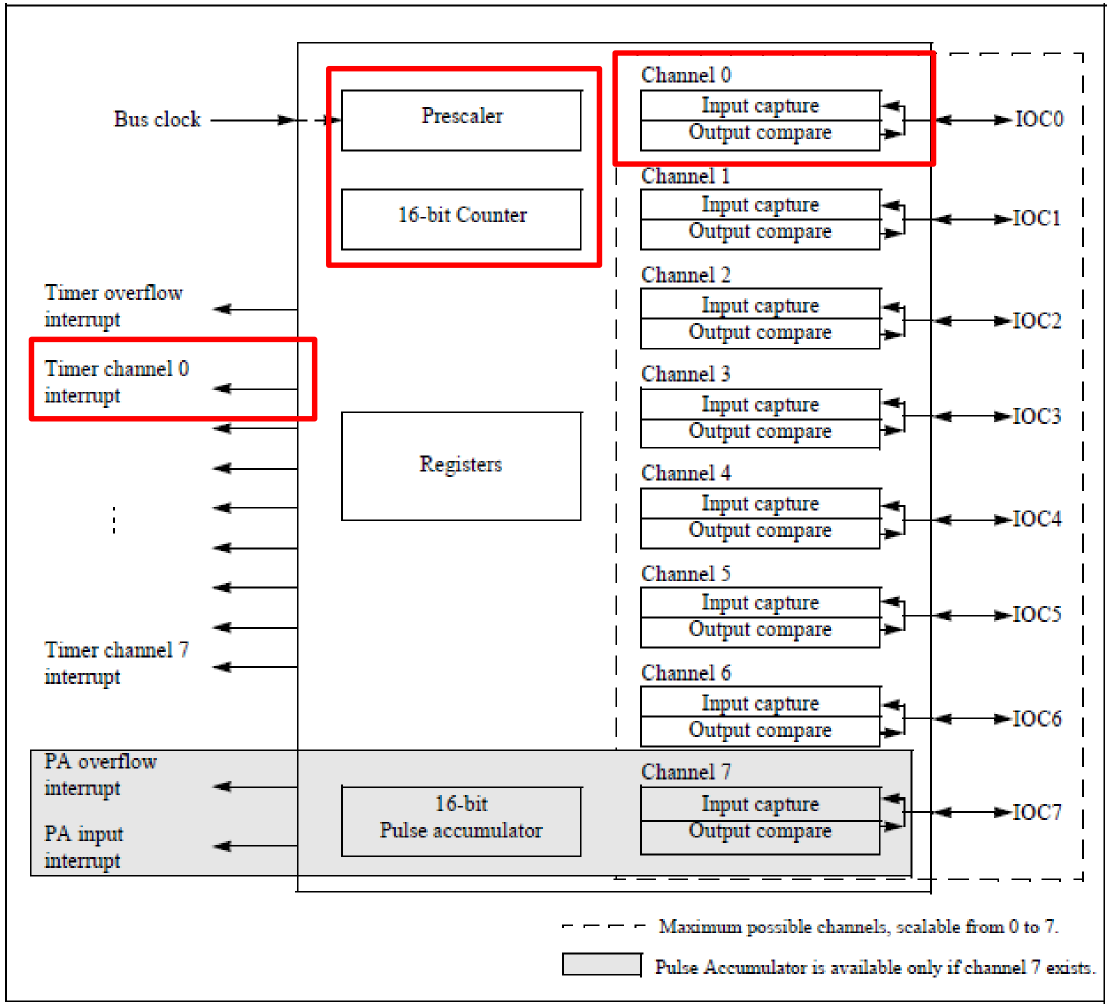
Locals
Globals
Registers
Peripherals

Breakpoints

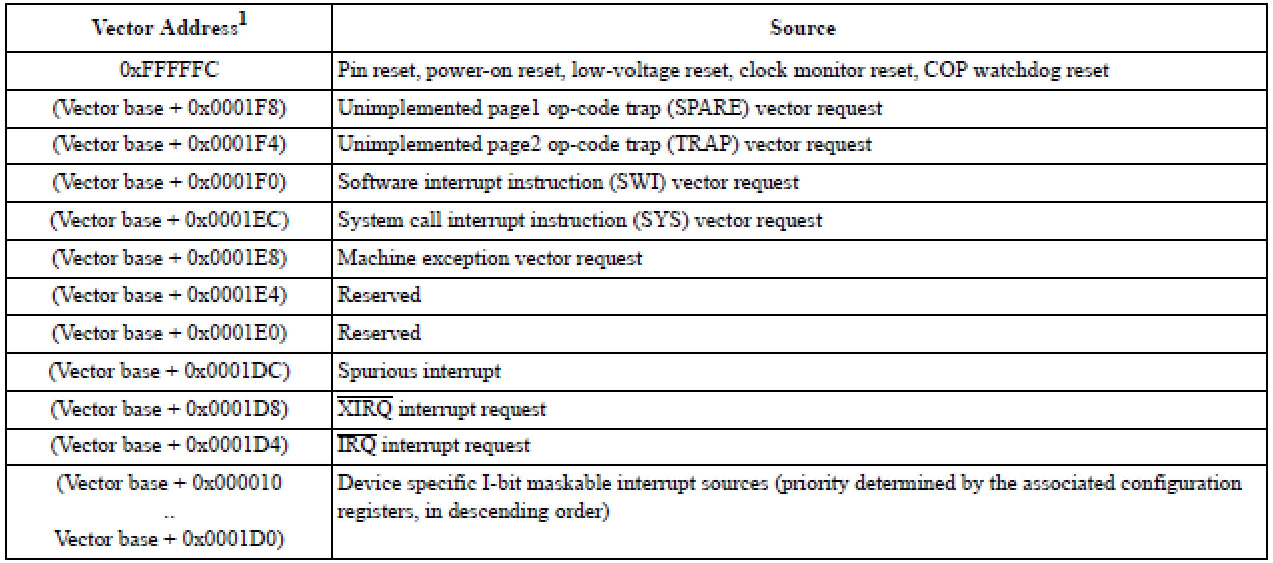
Hello World + Interrupts (GPIO + Timer)

Hello World + Interrupts: Introduction

Summary: An interrupt is implemented to service the TIM0 instead of software polling of the prior project



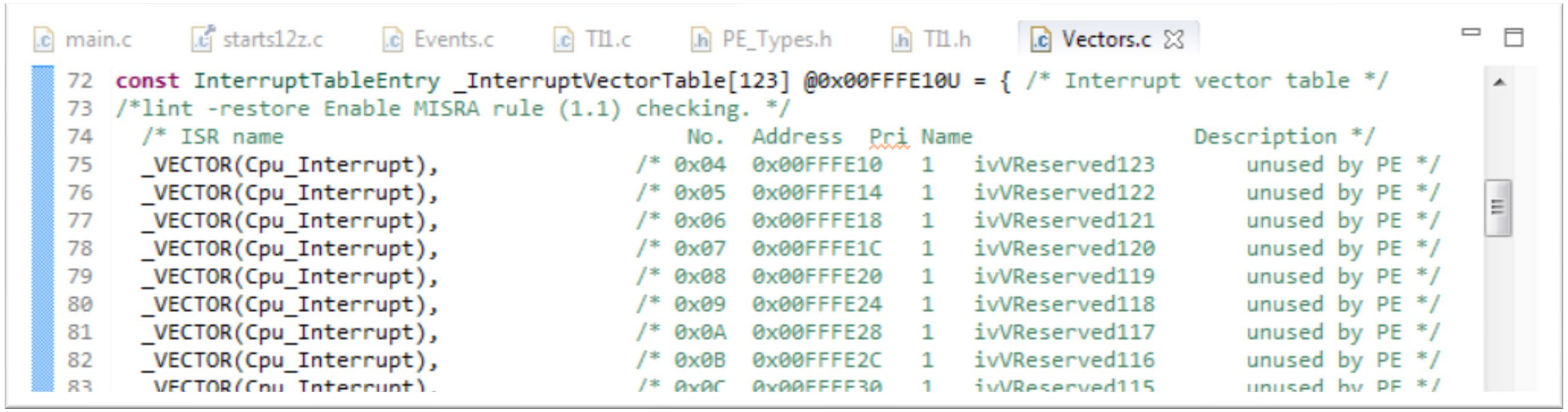
Hello World + Interrupts: Reference Manual + PEx Files



Vector Address[1]	Source
0xFFFFFC	Pin reset, power-on reset, low-voltage reset, clock monitor reset, COP watchdog reset
(Vector base + 0x0001F8)	Unimplemented page1 op-code trap (SPARE) vector request
(Vector base + 0x0001F4)	Unimplemented page2 op-code trap (TRAP) vector request
(Vector base + 0x0001F0)	Software interrupt instruction (SWI) vector request
(Vector base + 0x0001EC)	System call interrupt instruction (SYS) vector request
(Vector base + 0x0001E8)	Machine exception vector request
(Vector base + 0x0001E4)	Reserved
(Vector base + 0x0001E0)	Reserved
(Vector base + 0x0001DC)	Spurious interrupt
(Vector base + 0x0001D8)	$\overline{\text{XIRQ}}$ interrupt request
(Vector base + 0x0001D4)	$\overline{\text{IRQ}}$ interrupt request
(Vector base + 0x000010 .. Vector base + 0x0001D0)	Device specific I-bit maskable interrupt sources (priority determined by the associated configuration registers, in descending order)

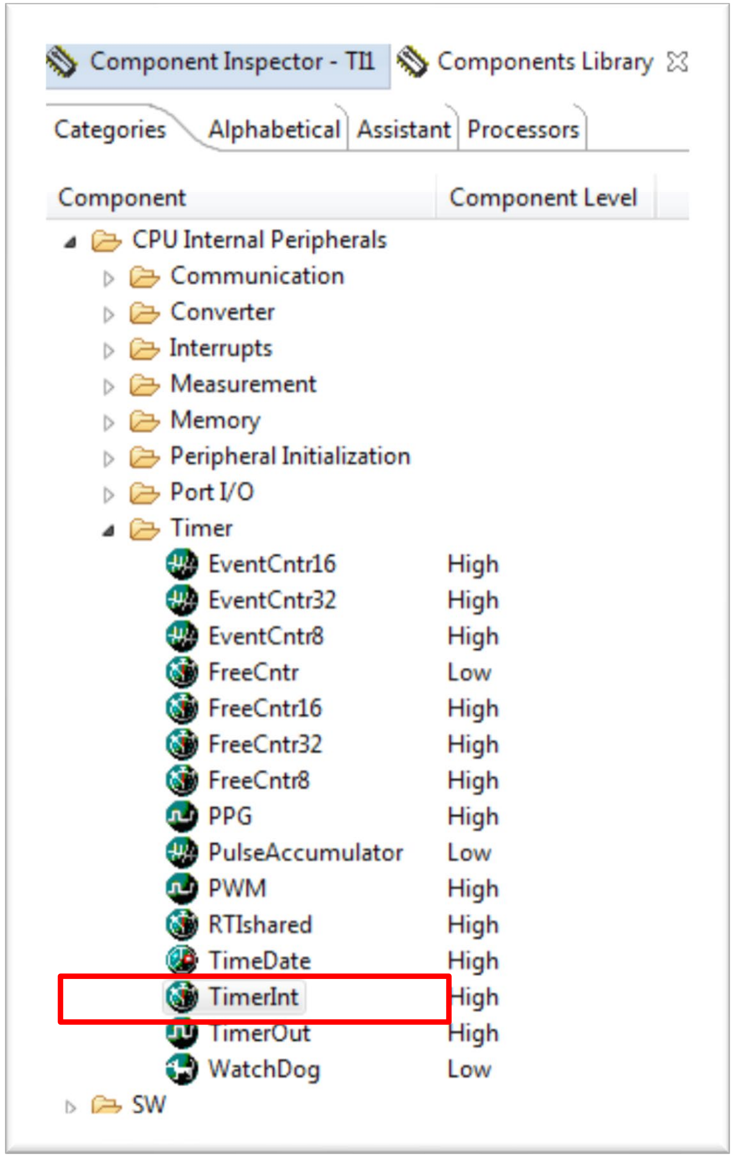
- Reference manual includes a generic vector table text description.

- Processor Expert includes vector table source code



```
const InterruptTableEntry _InterruptVectorTable[123] @0x00FFFE10U = { /* Interrupt vector table */
/*lint -restore Enable MISRA rule (1.1) checking. */
  /* ISR name                              No.  Address   Pri Name                Description */
  _VECTOR(Cpu_Interrupt),                 /* 0x04  0x00FFFE10   1   ivVReserved123      unused by PE */
  _VECTOR(Cpu_Interrupt),                 /* 0x05  0x00FFFE14   1   ivVReserved122      unused by PE */
  _VECTOR(Cpu_Interrupt),                 /* 0x06  0x00FFFE18   1   ivVReserved121      unused by PE */
  _VECTOR(Cpu_Interrupt),                 /* 0x07  0x00FFFE1C   1   ivVReserved120      unused by PE */
  _VECTOR(Cpu_Interrupt),                 /* 0x08  0x00FFFE20   1   ivVReserved119      unused by PE */
  _VECTOR(Cpu_Interrupt),                 /* 0x09  0x00FFFE24   1   ivVReserved118      unused by PE */
  _VECTOR(Cpu_Interrupt),                 /* 0x0A  0x00FFFE28   1   ivVReserved117      unused by PE */
  _VECTOR(Cpu_Interrupt),                 /* 0x0B  0x00FFFE2C   1   ivVReserved116      unused by PE */
  _VECTOR(Cpu_Interrupt),                 /* 0x0C  0x00FFFE30   1   ivVReserved115      unused by PE */
```

Timer Component

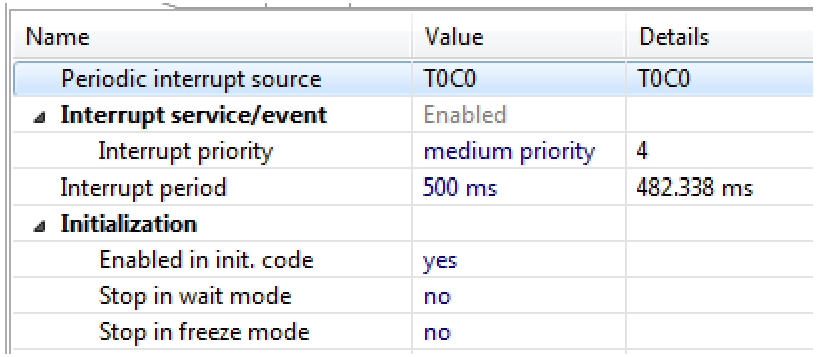


Component Inspector - TI1 | Components Library

Categories | Alphabetical | Assistant | Processors

Component	Component Level
CPU Internal Peripherals	
Communication	
Converter	
Interrupts	
Measurement	
Memory	
Peripheral Initialization	
Port I/O	
Timer	
EventCntr16	High
EventCntr32	High
EventCntr8	High
FreeCntr	Low
FreeCntr16	High
FreeCntr32	High
FreeCntr8	High
PPG	High
PulseAccumulator	Low
PWM	High
RTIshared	High
TimeDate	High
TimerInt	High
TimerOut	High
WatchDog	Low
SW	

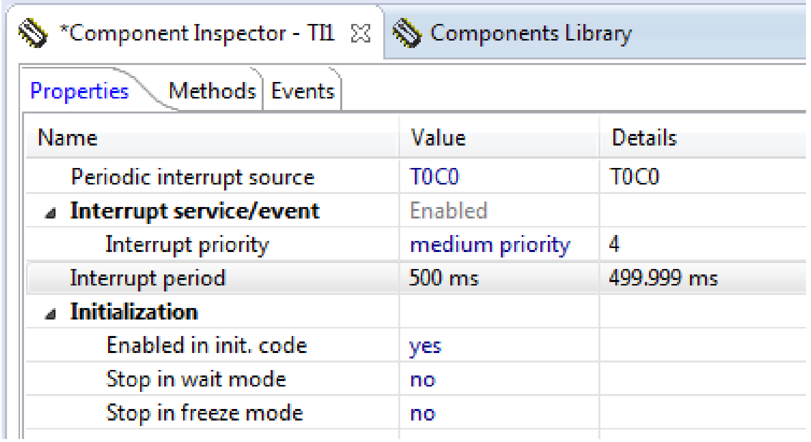
1) Component Library
2) CPU Internal Peripherals
3) Timer
4) Add “TimerInt” to your project
5) Move to PEX Components pane
6) Double-click “TI1:TimerInt” to edit its configuration



Name	Value	Details
Periodic interrupt source	TOC0	TOC0
Interrupt service/event	Enabled	
Interrupt priority	medium priority	4
Interrupt period	500 ms	482.338 ms
Initialization		
Enabled in init. code	yes	
Stop in wait mode	no	
Stop in freeze mode	no	

Timer Component: Configuration

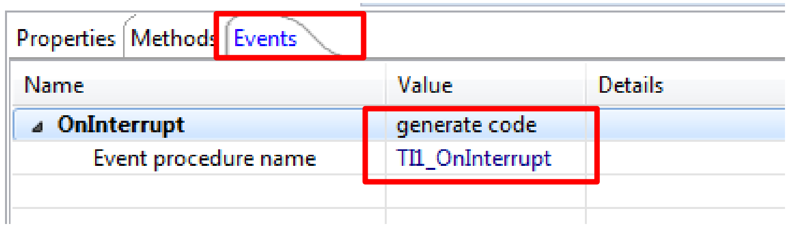
1) Choose source “T0C0”



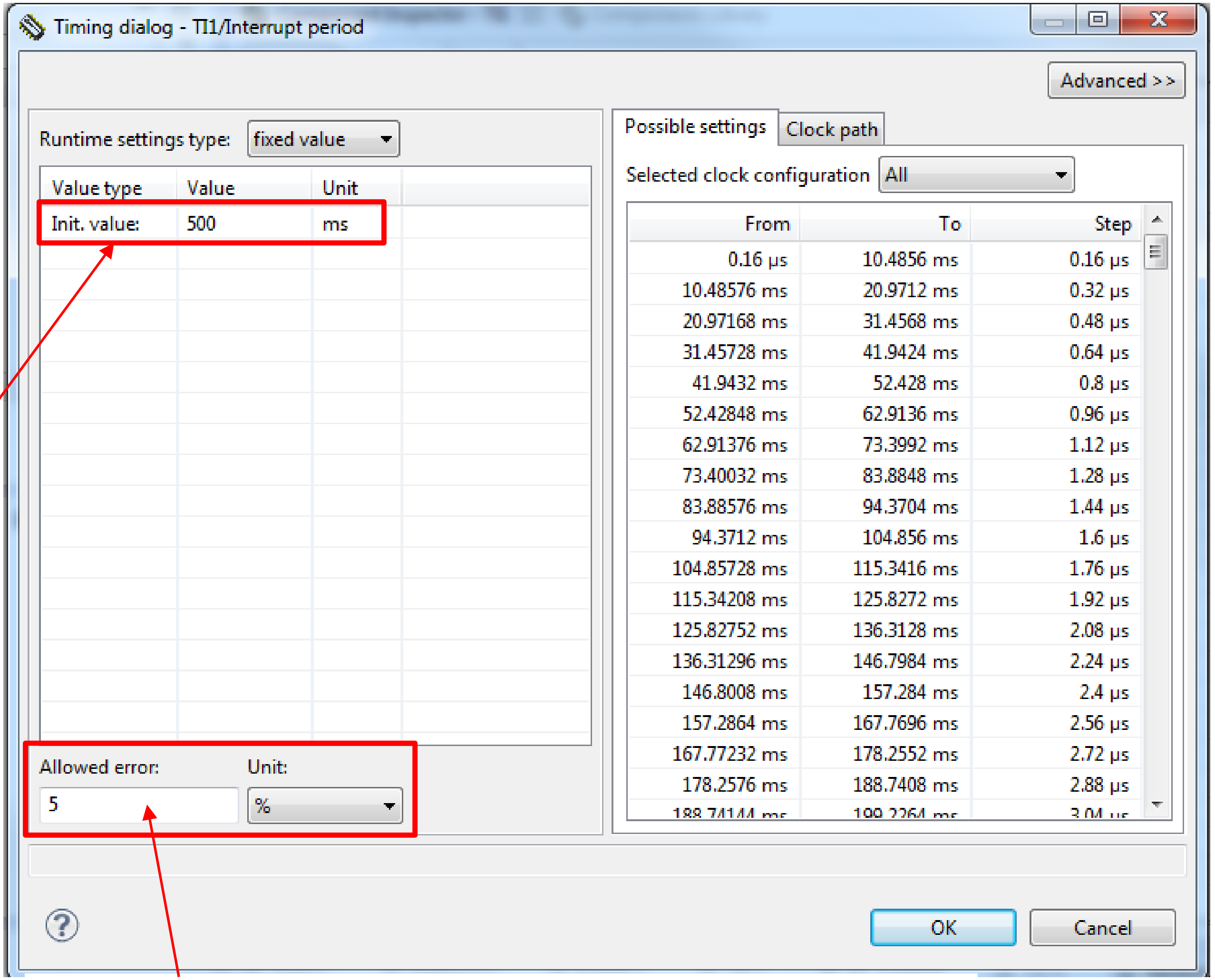
Name	Value	Details
Periodic interrupt source	T0C0	T0C0
Interrupt service/event	Enabled	
Interrupt priority	medium priority	4
Interrupt period	500 ms	499.999 ms
Initialization		
Enabled in init. code	yes	
Stop in wait mode	no	
Stop in freeze mode	no	

2) Configure the interval period to be 500ms (type into ‘Value’ field)

3) Inspect Events tab to ensure interrupt is defined



Name	Value	Details
OnInterrupt	generate code	
Event procedure name	TI1_OnInterrupt	



Runtime settings type: fixed value

Value type	Value	Unit
Init. value:	500	ms

Allowed error: 5

Unit: %

Selected clock configuration: All

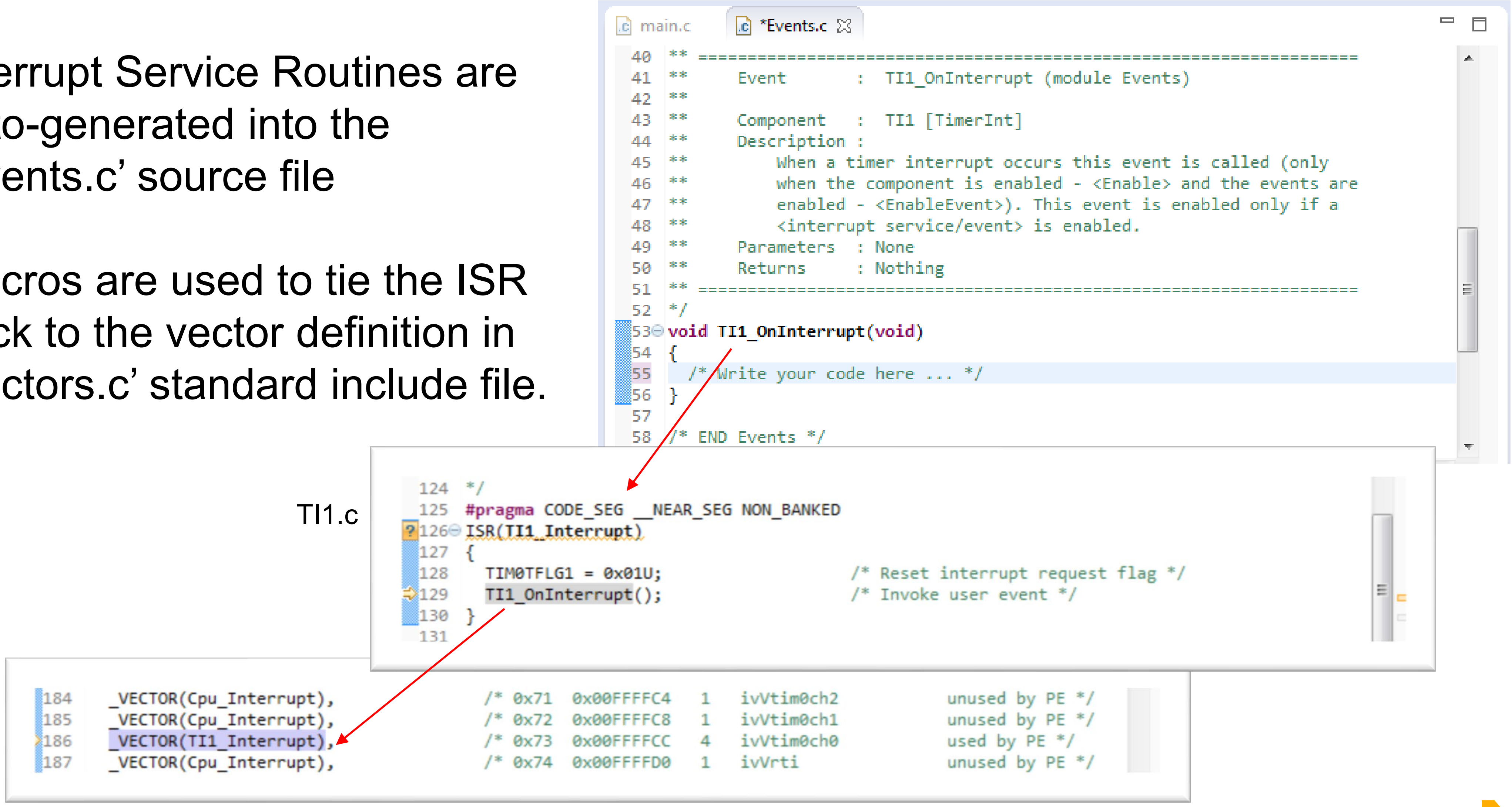
From	To	Step
0.16 µs	10.4856 ms	0.16 µs
10.48576 ms	20.9712 ms	0.32 µs
20.97168 ms	31.4568 ms	0.48 µs
31.45728 ms	41.9424 ms	0.64 µs
41.9432 ms	52.428 ms	0.8 µs
52.42848 ms	62.9136 ms	0.96 µs
62.91376 ms	73.3992 ms	1.12 µs
73.40032 ms	83.8848 ms	1.28 µs
83.88576 ms	94.3704 ms	1.44 µs
94.3712 ms	104.856 ms	1.6 µs
104.85728 ms	115.3416 ms	1.76 µs
115.34208 ms	125.8272 ms	1.92 µs
125.82752 ms	136.3128 ms	2.08 µs
136.31296 ms	146.7984 ms	2.24 µs
146.8008 ms	157.284 ms	2.4 µs
157.2864 ms	167.7696 ms	2.56 µs
167.77232 ms	178.2552 ms	2.72 µs
178.2576 ms	188.7408 ms	2.88 µs
188.74144 ms	199.2264 ms	3.04 µs

Experiment with this value and observe result in Component Inspector

Navigate to the Interrupt Service Routine

1) Interrupt Service Routines are auto-generated into the ‘Events.c’ source file

2) Macros are used to tie the ISR back to the vector definition in ‘Vectors.c’ standard include file.



```
** ===================================================================
**     Event       :  TI1_OnInterrupt (module Events)
**
**     Component   :  TI1 [TimerInt]
**     Description :
**         When a timer interrupt occurs this event is called (only
**         when the component is enabled - <Enable> and the events are
**         enabled - <EnableEvent>). This event is enabled only if a
**         <interrupt service/event> is enabled.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void TI1_OnInterrupt(void)
{
  /* Write your code here ... */
}

/* END Events */
```

TI1.c

```
*/
#pragma CODE_SEG __NEAR_SEG NON_BANKED
ISR(TI1_Interrupt)
{
  TIM0TFLG1 = 0x01U;                   /* Reset interrupt request flag */
  TI1_OnInterrupt();                   /* Invoke user event */
}
```

Vectors.c

```
  _VECTOR(Cpu_Interrupt),             /* 0x71  0x00FFFFC4   1   ivVtim0ch2           unused by PE */
  _VECTOR(Cpu_Interrupt),             /* 0x72  0x00FFFFC8   1   ivVtim0ch1           unused by PE */
  _VECTOR(TI1_Interrupt),             /* 0x73  0x00FFFFCC   4   ivVtim0ch0           used by PE */
  _VECTOR(Cpu_Interrupt),             /* 0x74  0x00FFFFD0   1   ivVrti               unused by PE */
```

Exercise #2 – Create

Objective

- Toggle the Green LED in the Timer interrupt service routine

Steps 1

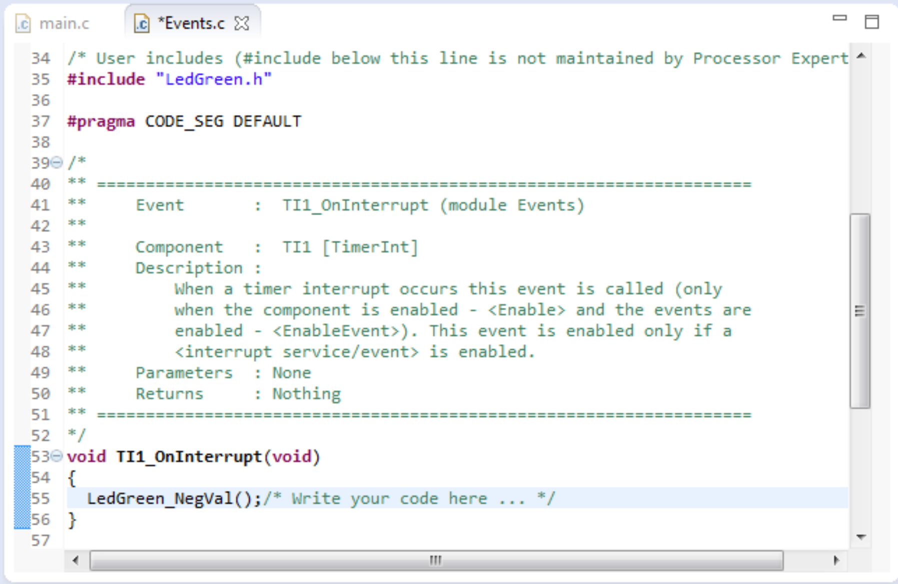
1) Delete the SW3 code from main() before continuing.

2) Create a new BitIO component at pin PP5 as an output to drive the Green LED
   - Configure parameters
   - Generate code

3) Add code to the TI1_Interrupt() function to toggle the LED – use 'NegVal()' method
   - May need to enable this method in the components pane

Steps 2

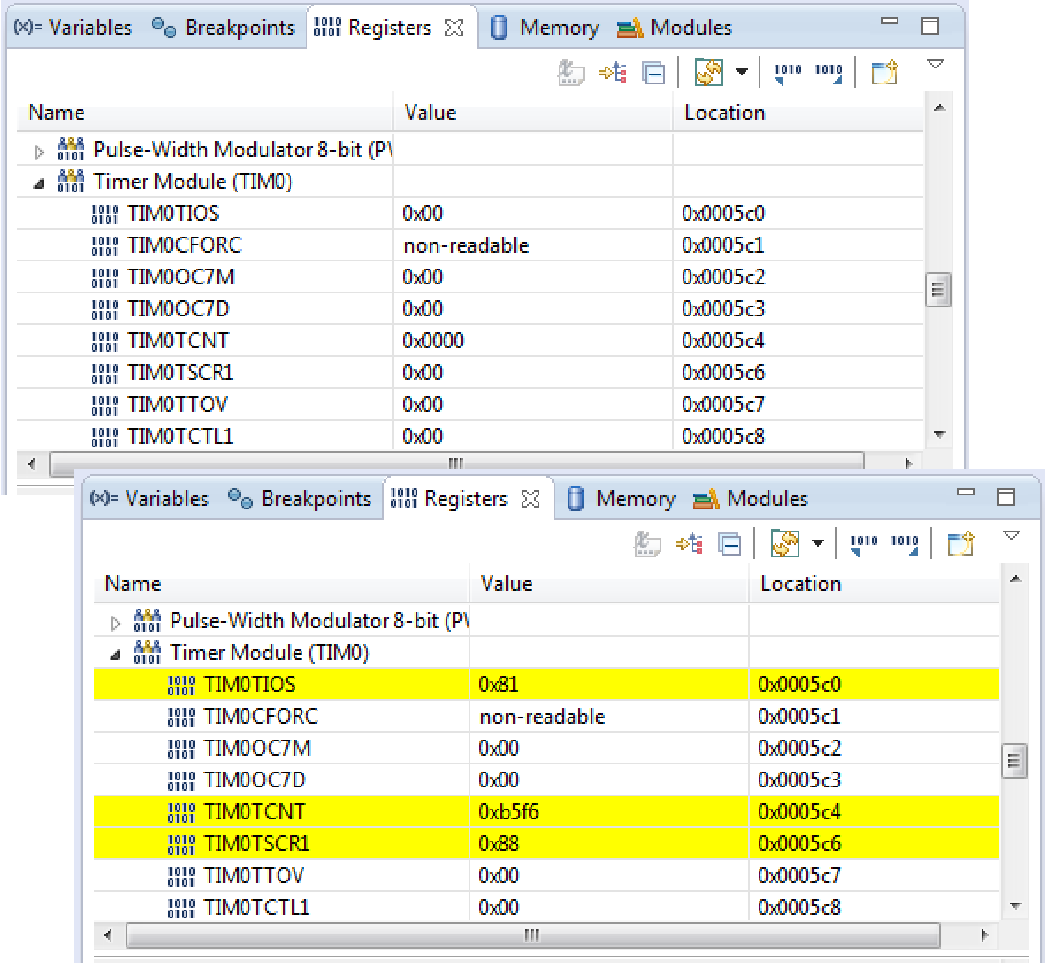
4) 'Events.c' will need to have access to the NegVal() method
   - Find the generated header file for your new BitIO component and add it to the appropriate location in Events.c



```
/* User includes (#include below this line is not maintained by Processor Expert
#include "LedGreen.h"

#pragma CODE_SEG DEFAULT

/*
** ===================================================================
**     Event       :  TI1_OnInterrupt (module Events)
**
**     Component   :  TI1 [TimerInt]
**     Description :
**         When a timer interrupt occurs this event is called (only
**         when the component is enabled - <Enable> and the events are
**         enabled - <EnableEvent>). This event is enabled only if a
**         <interrupt service/event> is enabled.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void TI1_OnInterrupt(void)
{
  LedGreen_NegVal();/* Write your code here ... */
}

```

Exercise #2 – Build & Debug (1)

1) Build & Debug the project
2) Run the code to make sure it works!
3) Use the Reset button to set the processor and peripherals back to initial hardware configuration
4) Place a breakpoint in main.c around line 50 at ‘PE_low_level_init()’
5) Run to breakpoint
6) Open the ‘Registers’ view and scroll down to “Timer Module TIM0”
7) Expand the list to view timer registers
8) Step-over the ‘PE_low_level_init()’ call
9) See changes in timer at Registers view

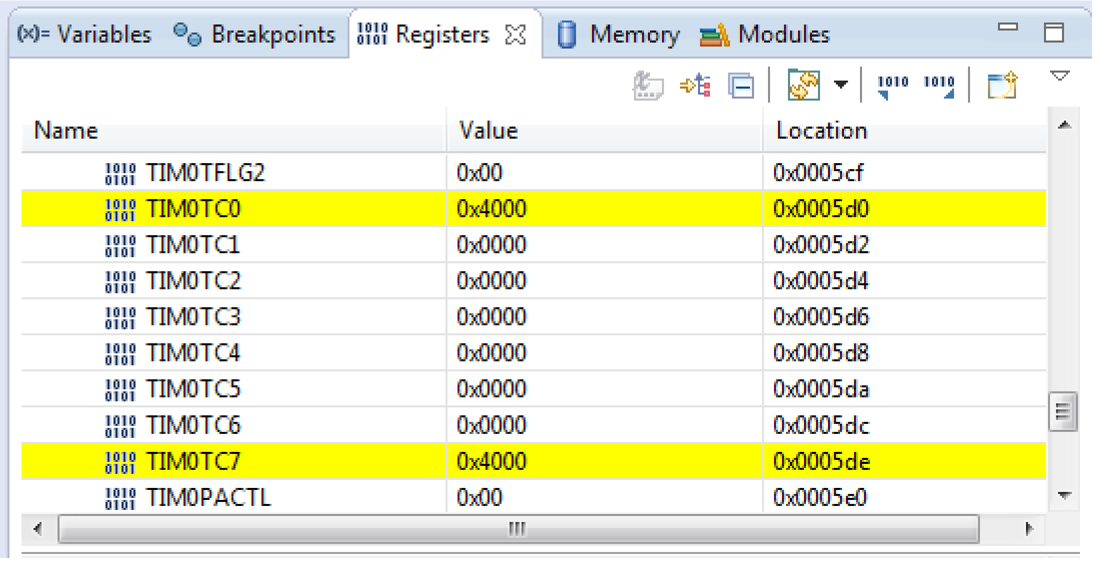


Name	Value	Location
Pulse-Width Modulator 8-bit (P\		
Timer Module (TIM0)		
TIM0TIOS	0x00	0x0005c0
TIM0CFORC	non-readable	0x0005c1
TIM0OC7M	0x00	0x0005c2
TIM0OC7D	0x00	0x0005c3
TIM0TCNT	0x0000	0x0005c4
TIM0TSCR1	0x00	0x0005c6
TIM0TTOV	0x00	0x0005c7
TIM0TCTL1	0x00	0x0005c8

Name	Value	Location
Pulse-Width Modulator 8-bit (P\		
Timer Module (TIM0)		
TIM0TIOS	0x81	0x0005c0
TIM0CFORC	non-readable	0x0005c1
TIM0OC7M	0x00	0x0005c2
TIM0OC7D	0x00	0x0005c3
TIM0TCNT	0xb5f6	0x0005c4
TIM0TSCR1	0x88	0x0005c6
TIM0TTOV	0x00	0x0005c7
TIM0TCTL1	0x00	0x0005c8

Exercise #2 – Build & Debug (2)

- Manually change the timer registers to make a new duty cycle of the LED
  - TIM0 compare register TIM0TC0
  - TIM0 modulo register TIM0TC7
  - Click on the ‘value’ cell and enter a new hexadecimal value to change the periodic rate of the timer.
  - Press ‘Enter’ to write it in.
- Just Resume ▶ running to see the new periodic rate



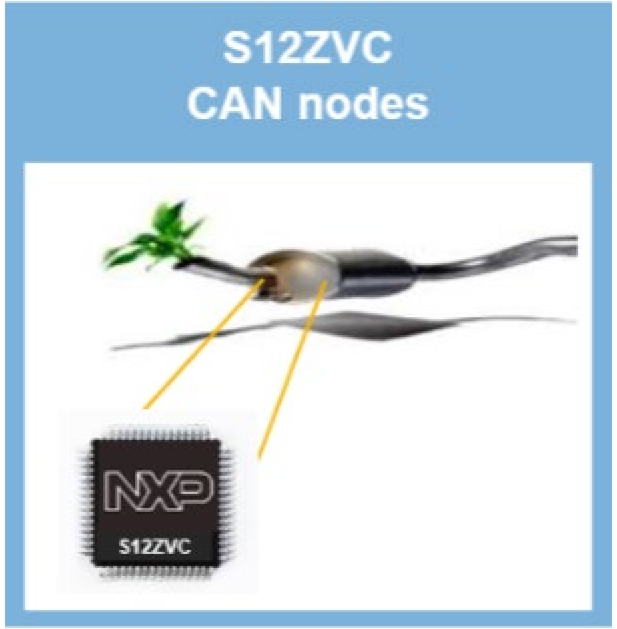
Name	Value	Location
TIM0TFLG2	0x00	0x0005cf
TIM0TC0	0x4000	0x0005d0
TIM0TC1	0x0000	0x0005d2
TIM0TC2	0x0000	0x0005d4
TIM0TC3	0x0000	0x0005d6
TIM0TC4	0x0000	0x0005d8
TIM0TC5	0x0000	0x0005da
TIM0TC6	0x0000	0x0005dc
TIM0TC7	0x4000	0x0005de
TIM0PACTL	0x00	0x0005e0

CAN Communications (MSCAN + CANPHY)

S12ZVC CAN Features

- MSCAN V3 module
  - Implementation of CAN 2.0 A/B protocol (Bosch)
  - Five receive buffers
  - Three transmit buffers

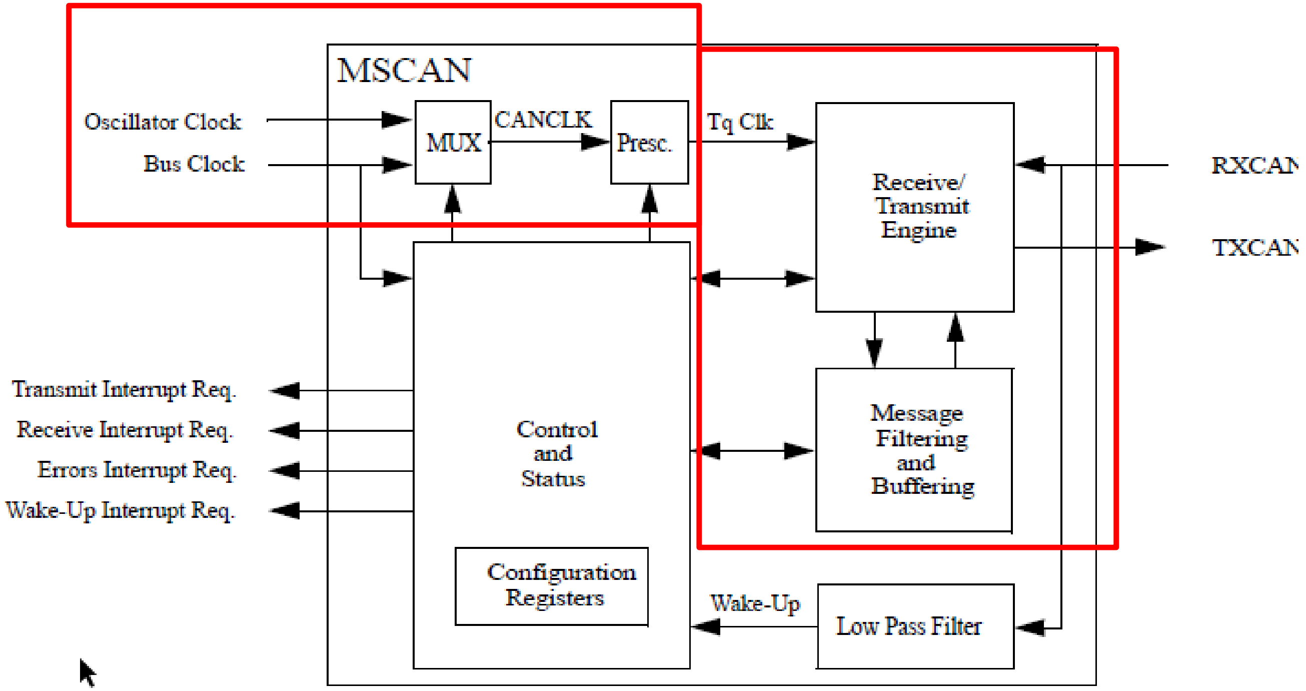
- One on-chip CAN physical layer module
  - ISO 11898-2 and ISO 11898-5 compliant for 12V battery system
  - Low-power mode with remote CAN wake-up
  - High Speed interface for baud rates of up to 1Mbit/s
  - CAN bus protection



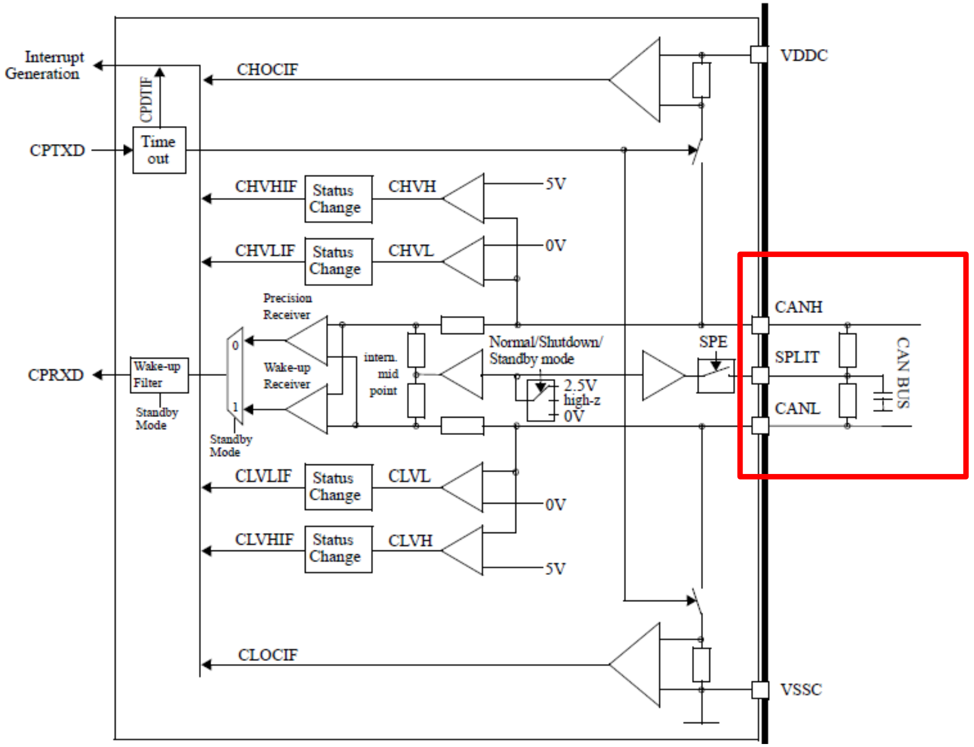
- Safety sensors
- Emission sensors
- Gear shift

S12ZVC CAN Architecture

MSCAN Digital Controller



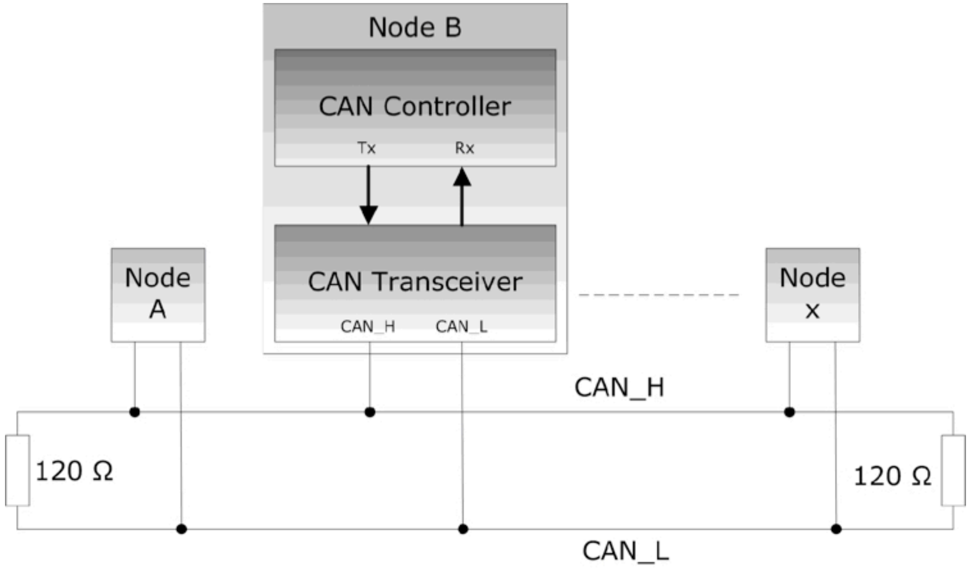
CAN Physical Layer



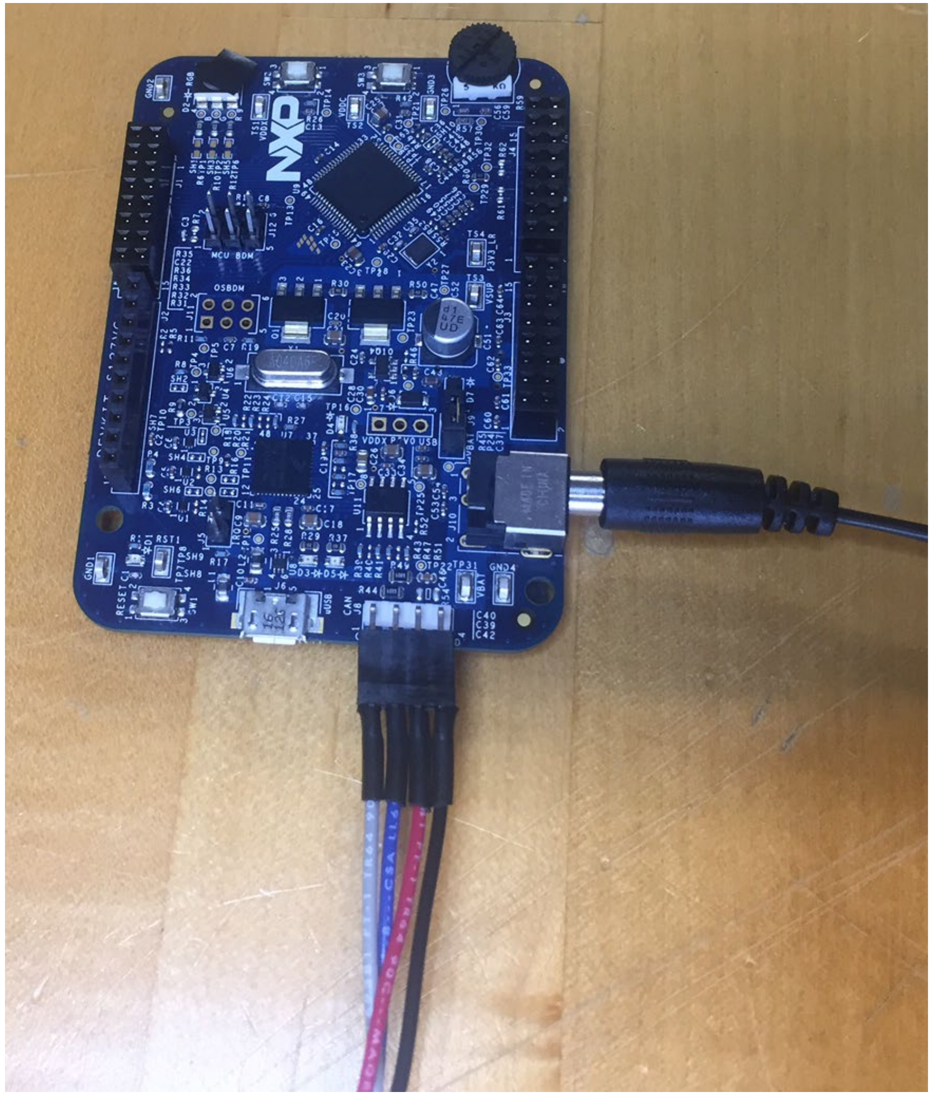
CAN Communication: Introduction

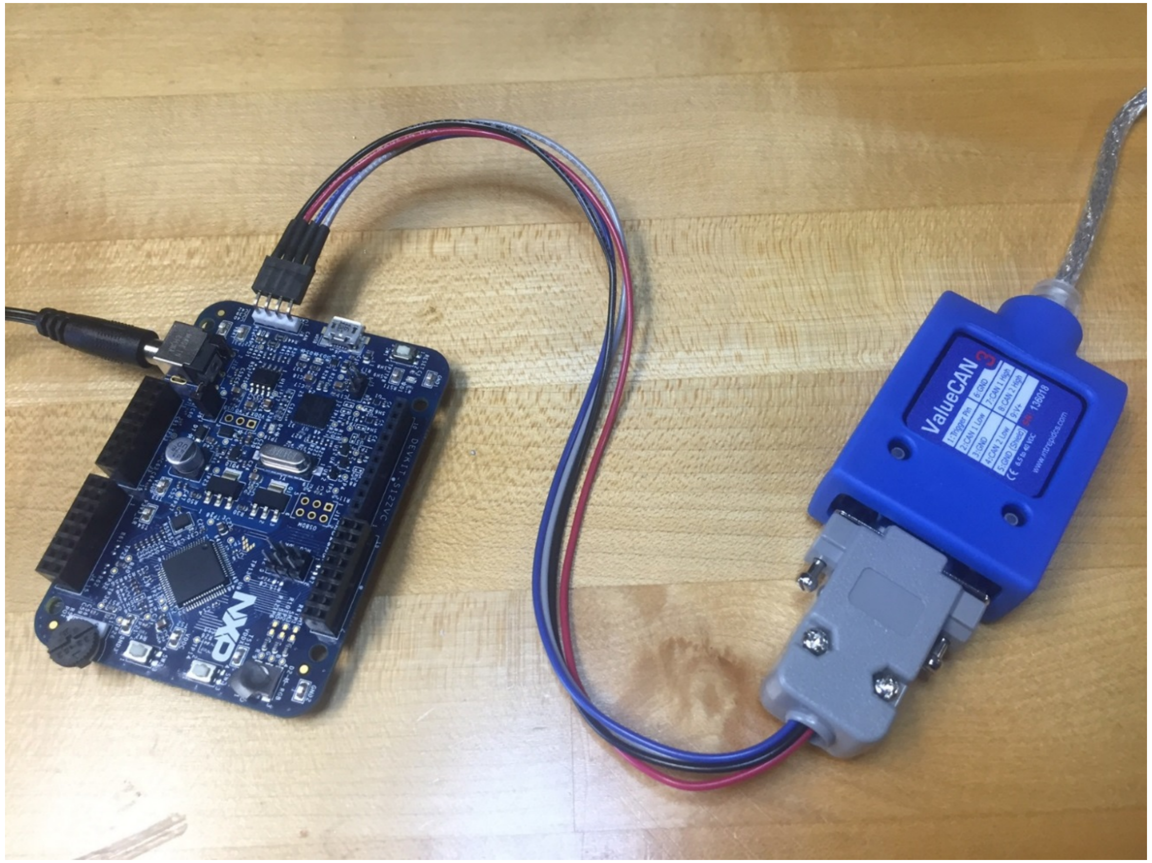
- Objective: A CAN frame is transmitted when SW3 is pressed. When a CAN message is received, the LED blinks.

- Key ingredients to communicate via CAN bus
  1) Accurate main clock
  2) PHY configuration
  3) Protocol engine configuration
  4) Message filter configuration

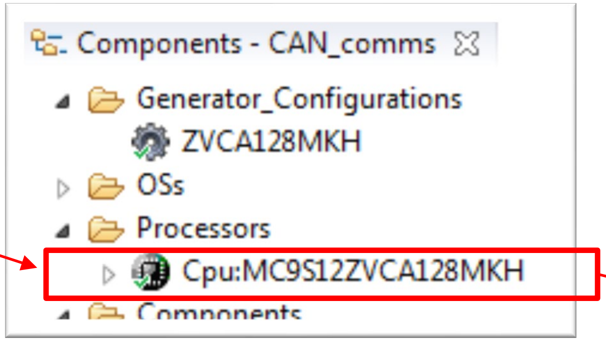


Connecting the EVB to VCAN4



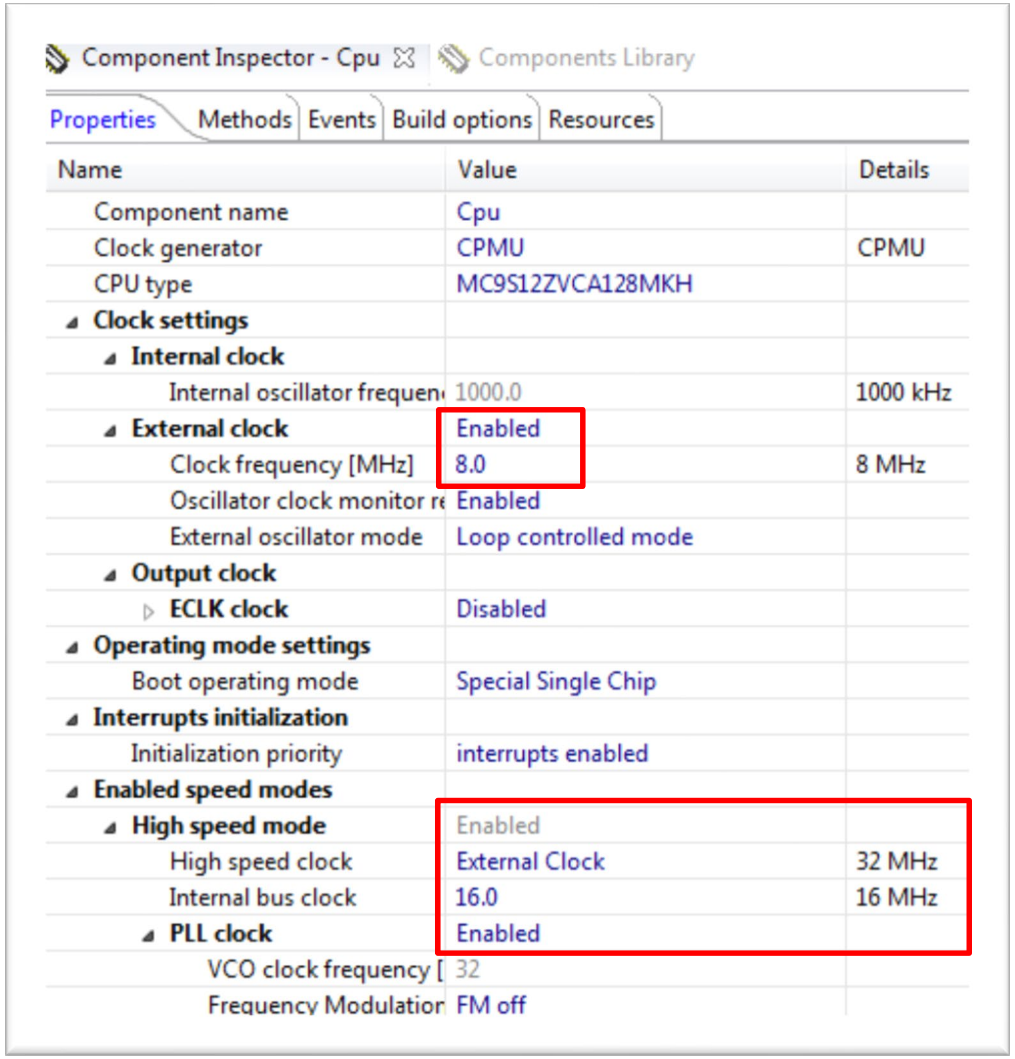


1) Main Clock Configuration



Components - CAN_comms

- Generator_Configurations
  - ZVCA128MKH
- OSs
- Processors
  - Cpu:MC9S12ZVCA128MKH
- Components



Component Inspector - Cpu | Components Library

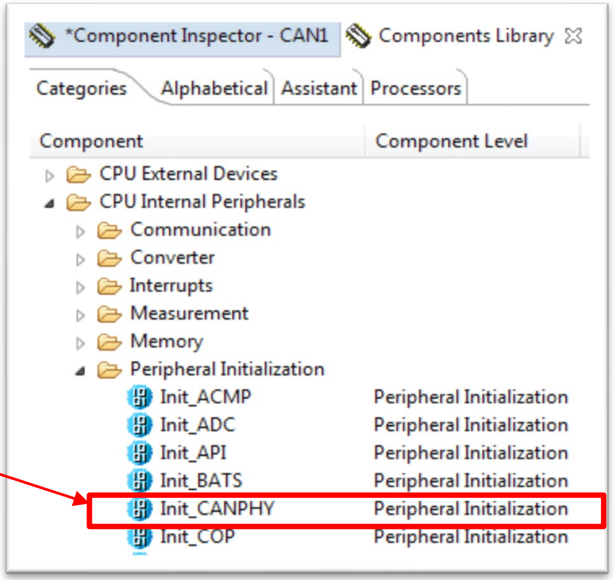
Properties | Methods | Events | Build options | Resources

Name	Value	Details
Component name	Cpu	
Clock generator	CPMU	CPMU
CPU type	MC9S12ZVCA128MKH	
Clock settings		
Internal clock		
Internal oscillator frequen	1000.0	1000 kHz
External clock	Enabled	
Clock frequency [MHz]	8.0	8 MHz
Oscillator clock monitor re	Enabled	
External oscillator mode	Loop controlled mode	
Output clock		
ECLK clock	Disabled	
Operating mode settings		
Boot operating mode	Special Single Chip	
Interrupts initialization		
Initialization priority	interrupts enabled	
Enabled speed modes		
High speed mode	Enabled	
High speed clock	External Clock	32 MHz
Internal bus clock	16.0	16 MHz
PLL clock	Enabled	
VCO clock frequency [	32	
Frequency Modulatior	FM off	

1. Start a new Bareboard project with S12ZVCA128 (File-> New…)
2. Open "CPU" in component inspector
3. Enable external clock at <u>8MHz</u>
4. Set internal bus clock to <u>16Mhz</u>
5. Set High Speed Clock to <u>External</u>
6. Generate code

2) Physical Layer Initialization

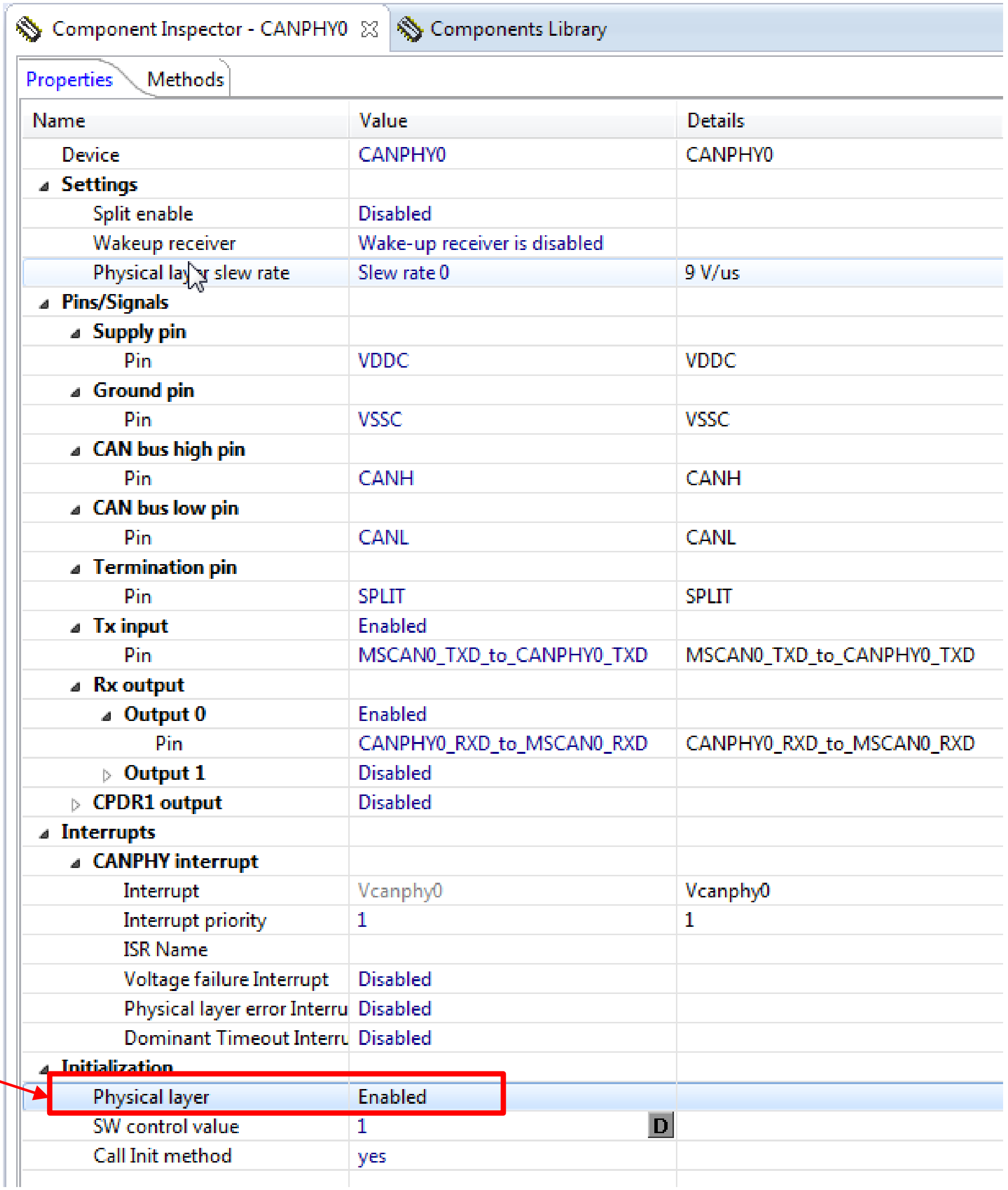
1. Add the Peripheral Initialization method “Init_CANPHY” from the Component Library.
2. ‘Enable’ the PHY
3. Check configurations.



*Component Inspector - CAN1 | Components Library

Categories | Alphabetical | Assistant | Processors

Component	Component Level
CPU External Devices	
CPU Internal Peripherals	
Communication	
Converter	
Interrupts	
Measurement	
Memory	
Peripheral Initialization	
Init_ACMP	Peripheral Initialization
Init_ADC	Peripheral Initialization
Init_API	Peripheral Initialization
Init_BATS	Peripheral Initialization
Init_CANPHY	Peripheral Initialization
Init_COP	Peripheral Initialization



Component Inspector - CANPHY0 | Components Library

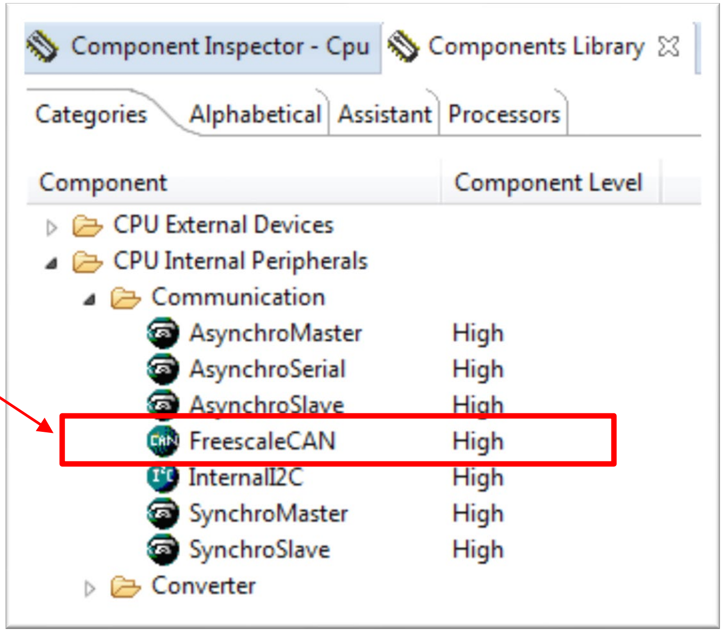
Properties | Methods

Name	Value	Details
Device	CANPHY0	CANPHY0
Settings		
Split enable	Disabled	
Wakeup receiver	Wake-up receiver is disabled	
Physical layer slew rate	Slew rate 0	9 V/us
Pins/Signals		
Supply pin		
Pin	VDDC	VDDC
Ground pin		
Pin	VSSC	VSSC
CAN bus high pin		
Pin	CANH	CANH
CAN bus low pin		
Pin	CANL	CANL
Termination pin		
Pin	SPLIT	SPLIT
Tx input	Enabled	
Pin	MSCAN0_TXD_to_CANPHY0_TXD	MSCAN0_TXD_to_CANPHY0_TXD
Rx output		
Output 0	Enabled	
Pin	CANPHY0_RXD_to_MSCAN0_RXD	CANPHY0_RXD_to_MSCAN0_RXD
Output 1	Disabled	
CPDR1 output	Disabled	
Interrupts		
CANPHY interrupt		
Interrupt	Vcanphy0	Vcanphy0
Interrupt priority	1	1
ISR Name		
Voltage failure Interrupt	Disabled	
Physical layer error Interru	Disabled	
Dominant Timeout Interru	Disabled	
Initialization		
Physical layer	Enabled	
SW control value	1	D
Call Init method	yes	

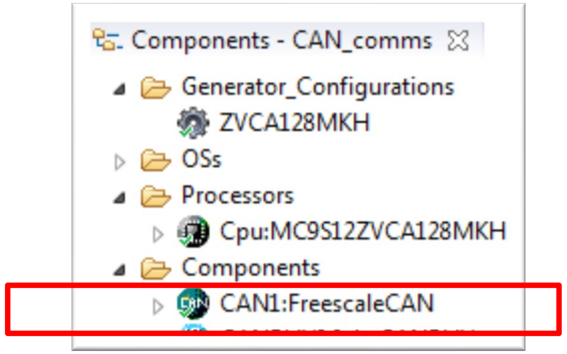
3a) CAN Protocol Engine Configuration

1. Add a “FreescaleCAN” component to the project, from the Components Library.

2. Open the CAN component’s properties by double-clicking it, in the Components tab.

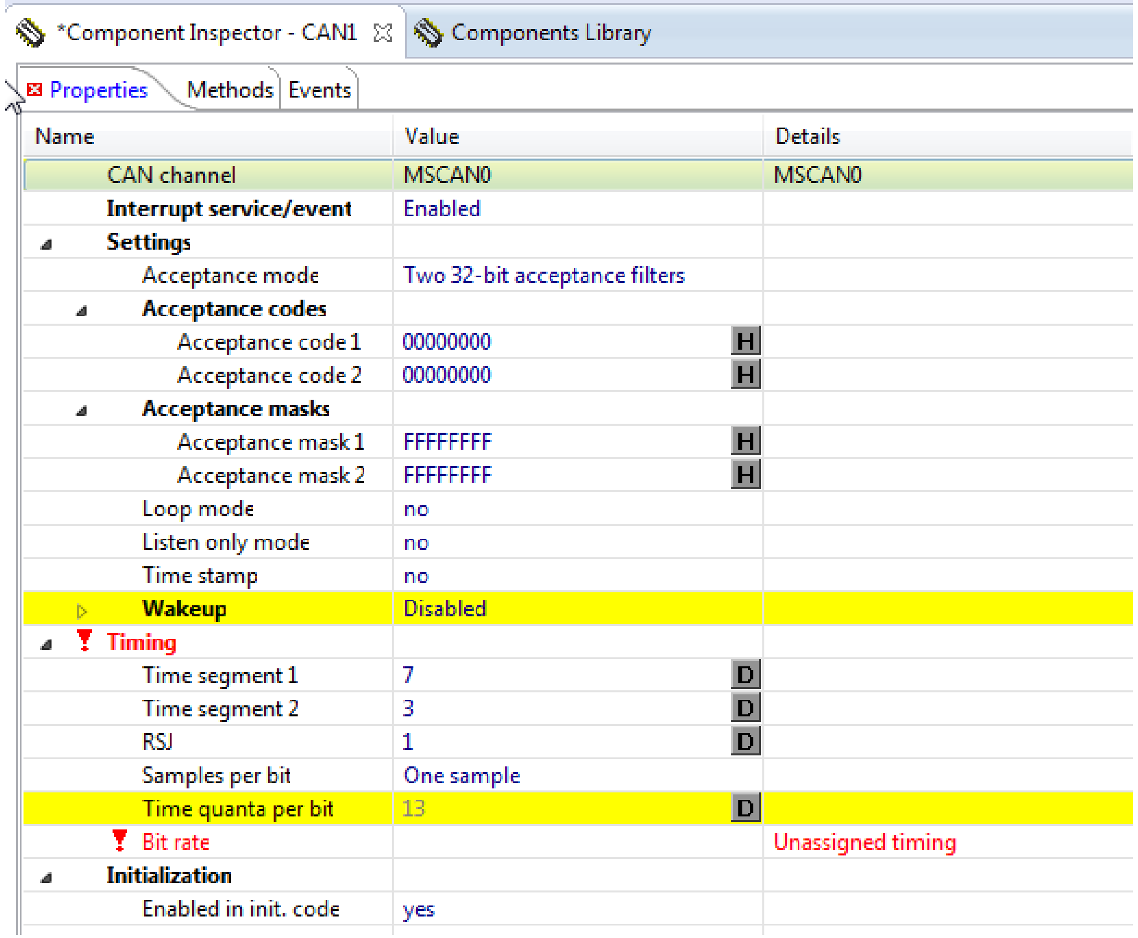


Component	Component Level
CPU External Devices	
CPU Internal Peripherals	
Communication	
AsynchroMaster	High
AsynchroSerial	High
AsynchroSlave	High
FreescaleCAN	High
InternalI2C	High
SynchroMaster	High
SynchroSlave	High
Converter	



- Generator_Configurations
  - ZVCA128MKH
- OSs
- Processors
  - Cpu:MC9S12ZVCA128MKH
- Components
  - CAN1:FreescaleCAN

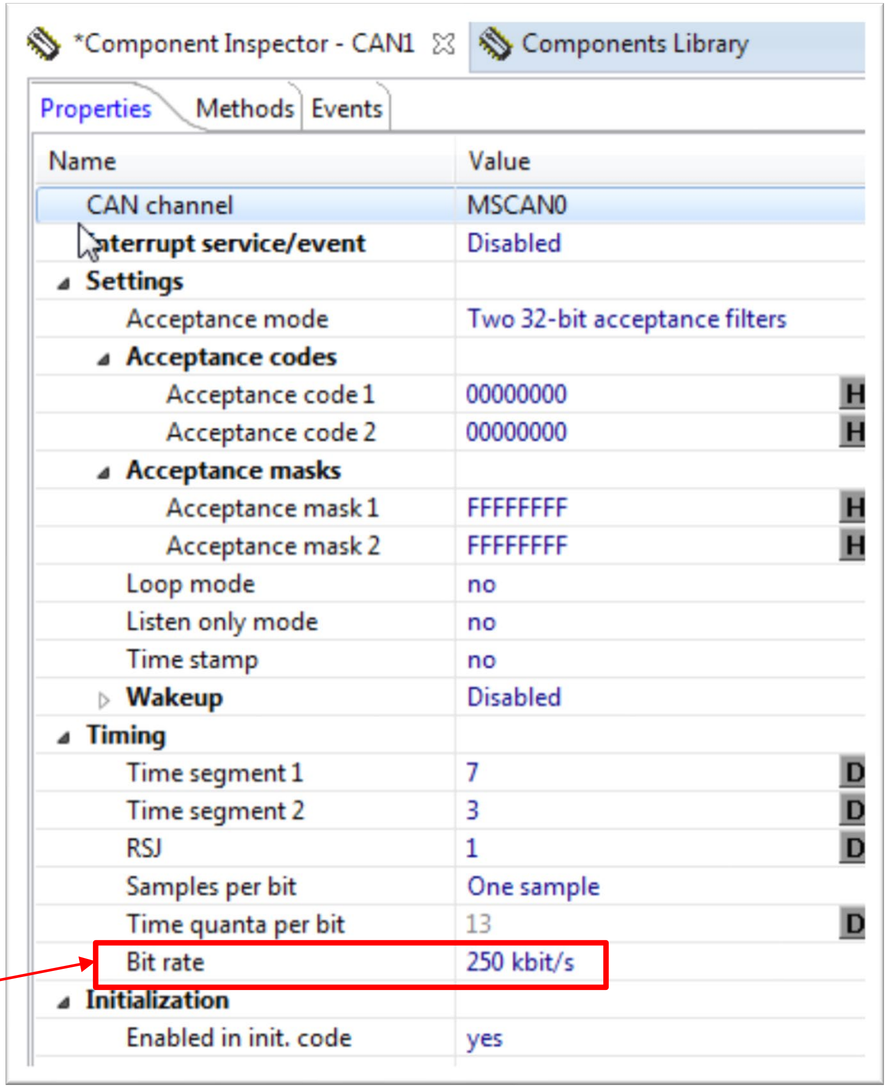
3b) CAN Protocol Engine Configuration



*Component Inspector - CAN1 | Components Library

Properties | Methods | Events

Name	Value		Details
CAN channel	MSCAN0		MSCAN0
Interrupt service/event	Enabled		
Settings			
Acceptance mode	Two 32-bit acceptance filters		
Acceptance codes			
Acceptance code 1	00000000	H	
Acceptance code 2	00000000	H	
Acceptance masks			
Acceptance mask 1	FFFFFFFF	H	
Acceptance mask 2	FFFFFFFF	H	
Loop mode	no		
Listen only mode	no		
Time stamp	no		
Wakeup	Disabled		
Timing			
Time segment 1	7	D	
Time segment 2	3	D	
RSJ	1	D	
Samples per bit	One sample		
Time quanta per bit	13	D	
Bit rate			Unassigned timing
Initialization			
Enabled in init. code	yes		



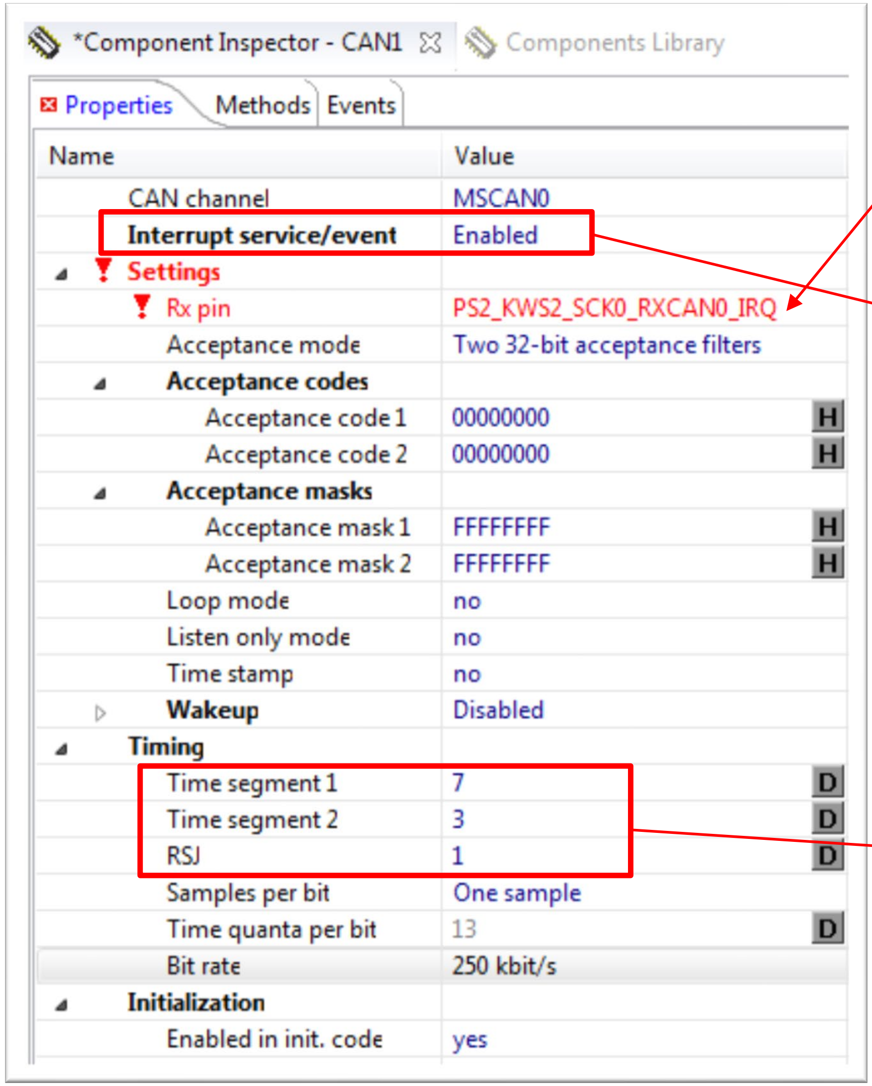
*Component Inspector - CAN1 | Components Library

Properties | Methods | Events

Name	Value	
CAN channel	MSCAN0	
Interrupt service/event	Disabled	
Settings		
Acceptance mode	Two 32-bit acceptance filters	
Acceptance codes		
Acceptance code 1	00000000	H
Acceptance code 2	00000000	H
Acceptance masks		
Acceptance mask 1	FFFFFFFF	H
Acceptance mask 2	FFFFFFFF	H
Loop mode	no	
Listen only mode	no	
Time stamp	no	
Wakeup	Disabled	
Timing		
Time segment 1	7	D
Time segment 2	3	D
RSJ	1	D
Samples per bit	One sample	
Time quanta per bit	13	D
Bit rate	250 kbit/s	
Initialization		
Enabled in init. code	yes	

1. Input “250” to set the bit rate

3c) CAN Protocol Engine Configuration

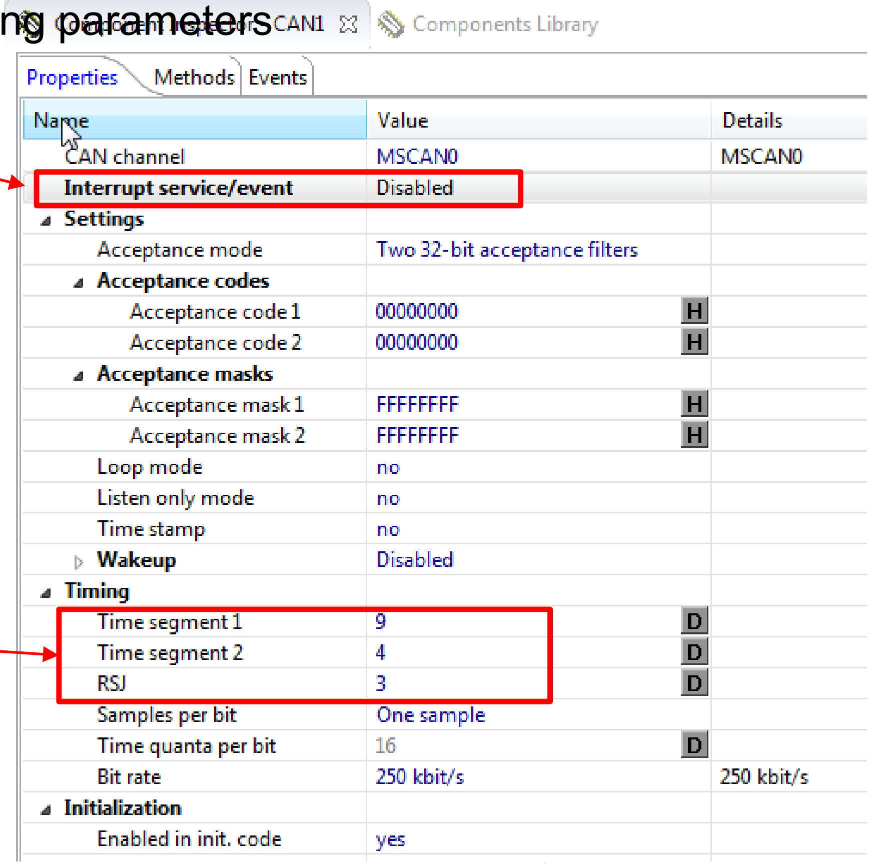


*Component Inspector - CAN1 | Components Library

Properties | Methods | Events

Name	Value	
CAN channel	MSCAN0	
Interrupt service/event	Enabled	
Settings		
Rx pin	PS2_KWS2_SCK0_RXCAN0_IRQ	
Acceptance mode	Two 32-bit acceptance filters	
Acceptance codes		
Acceptance code 1	00000000	H
Acceptance code 2	00000000	H
Acceptance masks		
Acceptance mask 1	FFFFFFFF	H
Acceptance mask 2	FFFFFFFF	H
Loop mode	no	
Listen only mode	no	
Time stamp	no	
Wakeup	Disabled	
Timing		
Time segment 1	7	D
Time segment 2	3	D
RSJ	1	D
Samples per bit	One sample	
Time quanta per bit	13	D
Bit rate	250 kbit/s	
Initialization		
Enabled in init. code	yes	

1. Correct the pin assignment issue, and it disappears from view…
2. Disable interrupts
3. Set bit timing parameters

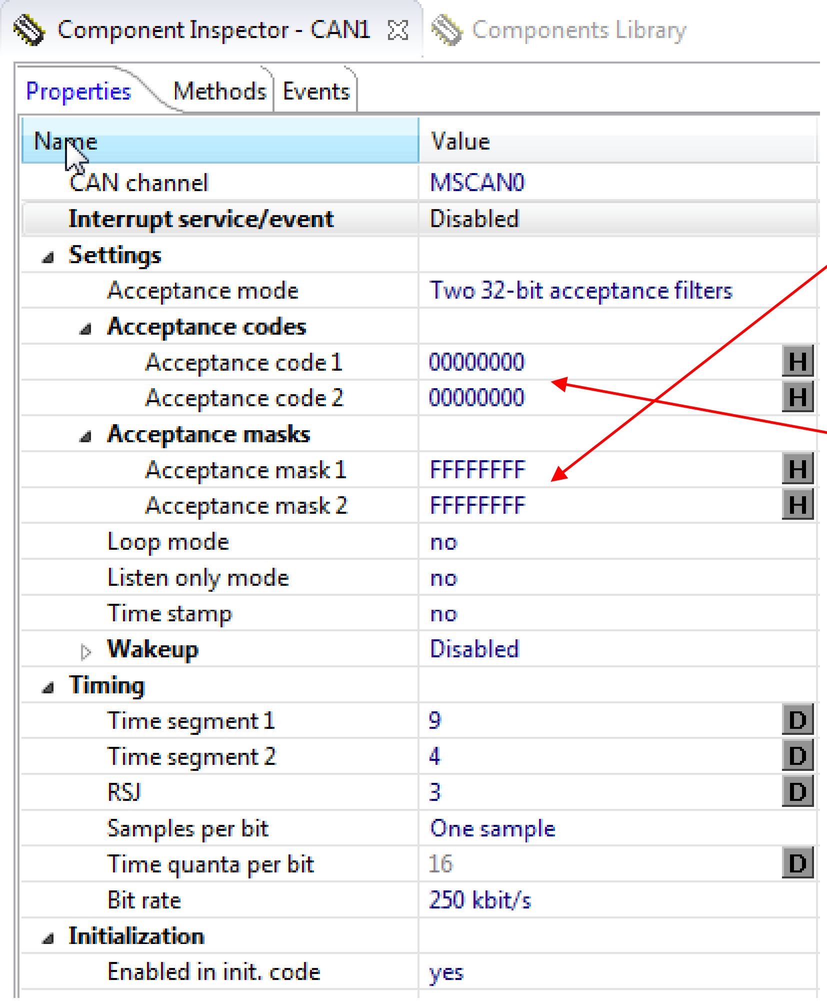


Component Inspector - CAN1 | Components Library

Properties | Methods | Events

Name	Value		Details
CAN channel	MSCAN0		MSCAN0
Interrupt service/event	Disabled		
Settings			
Acceptance mode	Two 32-bit acceptance filters		
Acceptance codes			
Acceptance code 1	00000000	H	
Acceptance code 2	00000000	H	
Acceptance masks			
Acceptance mask 1	FFFFFFFF	H	
Acceptance mask 2	FFFFFFFF	H	
Loop mode	no		
Listen only mode	no		
Time stamp	no		
Wakeup	Disabled		
Timing			
Time segment 1	9	D	
Time segment 2	4	D	
RSJ	3	D	
Samples per bit	One sample		
Time quanta per bit	16	D	
Bit rate	250 kbit/s		250 kbit/s
Initialization			
Enabled in init. code	yes		

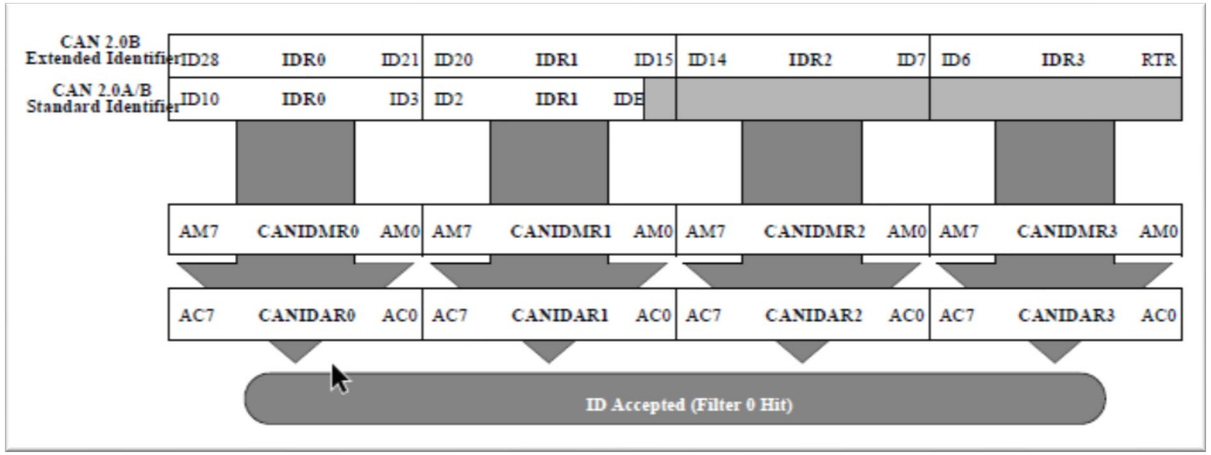
4) Message Filter Configuration



Name	Value	
CAN channel	MSCAN0	
Interrupt service/event	Disabled	
Settings		
Acceptance mode	Two 32-bit acceptance filters	
Acceptance codes		
Acceptance code 1	00000000	H
Acceptance code 2	00000000	H
Acceptance masks		
Acceptance mask 1	FFFFFFFF	H
Acceptance mask 2	FFFFFFFF	H
Loop mode	no	
Listen only mode	no	
Time stamp	no	
Wakeup	Disabled	
Timing		
Time segment 1	9	D
Time segment 2	4	D
RSJ	3	D
Samples per bit	One sample	
Time quanta per bit	16	D
Bit rate	250 kbit/s	
Initialization		
Enabled in init. code	yes	

- For easy prototyping, use two 32-bit filters and open acceptance code & mask.
- Acceptance mask:
  - ‘0’ means we must match the received ID with the acceptance code bit field
  - ‘1’ means we don’t care
- Acceptance code: the received ID bit must match a ‘1’ or ‘0’

→ With the default setting, all messages will be received.



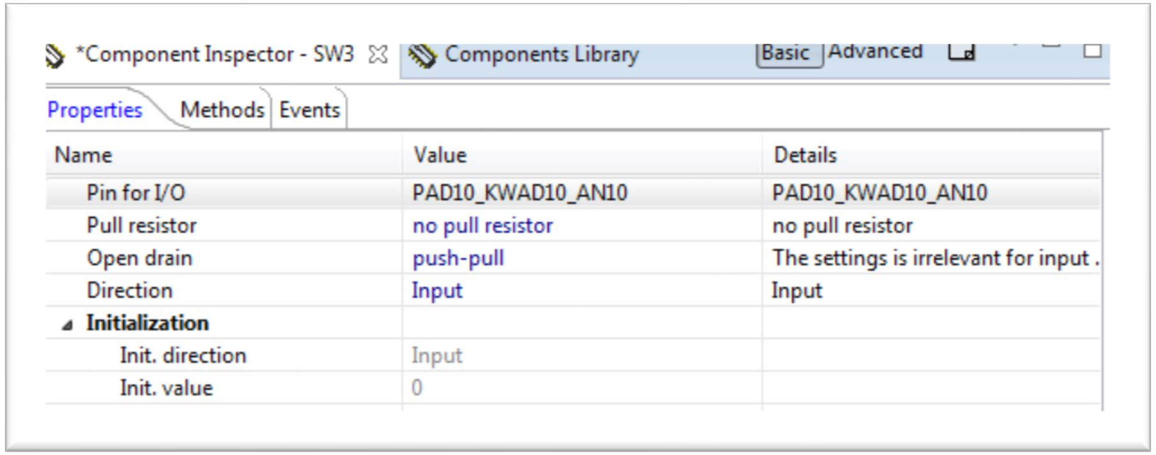
Now… Re-create SW3 Input and Red LED Output

1) Create a new BitIO component

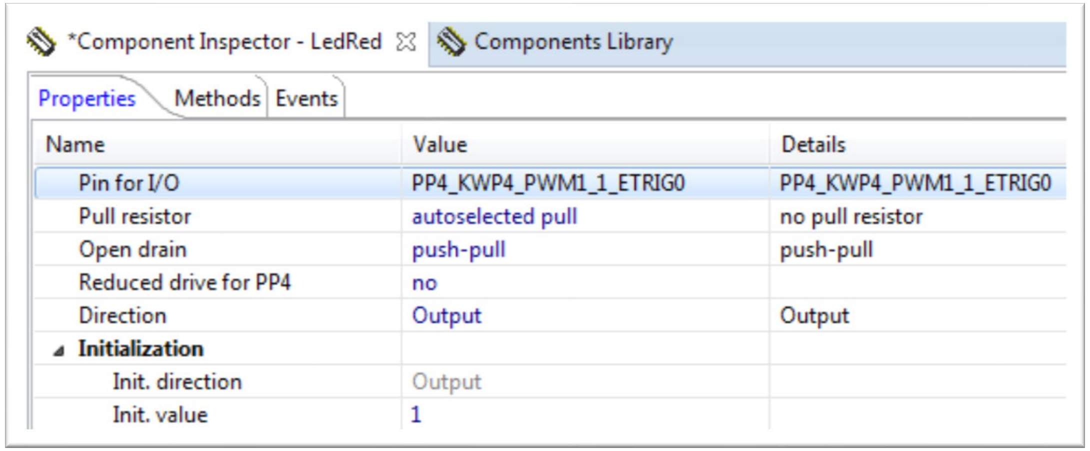
- Name: “SW3”
- Pin: PAD10
- Direction: Input

2) Create a new BitIO component

- Name: “LedRed”
- Pin: PP4
- Direction: Output
- Init. Value: 1



Name	Value	Details
Pin for I/O	PAD10_KWAD10_AN10	PAD10_KWAD10_AN10
Pull resistor	no pull resistor	no pull resistor
Open drain	push-pull	The settings is irrelevant for input .
Direction	Input	Input
Initialization		
Init. direction	Input	
Init. value	0	

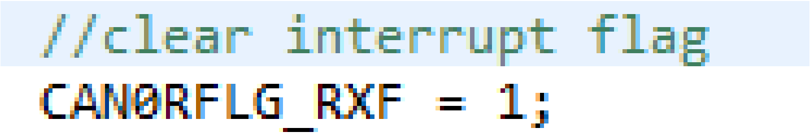


Name	Value	Details
Pin for I/O	PP4_KWP4_PWM1_1_ETRIG0	PP4_KWP4_PWM1_1_ETRIG0
Pull resistor	autoselected pull	no pull resistor
Open drain	push-pull	push-pull
Reduced drive for PP4	no	
Direction	Output	Output
Initialization		
Init. direction	Output	
Init. value	1	

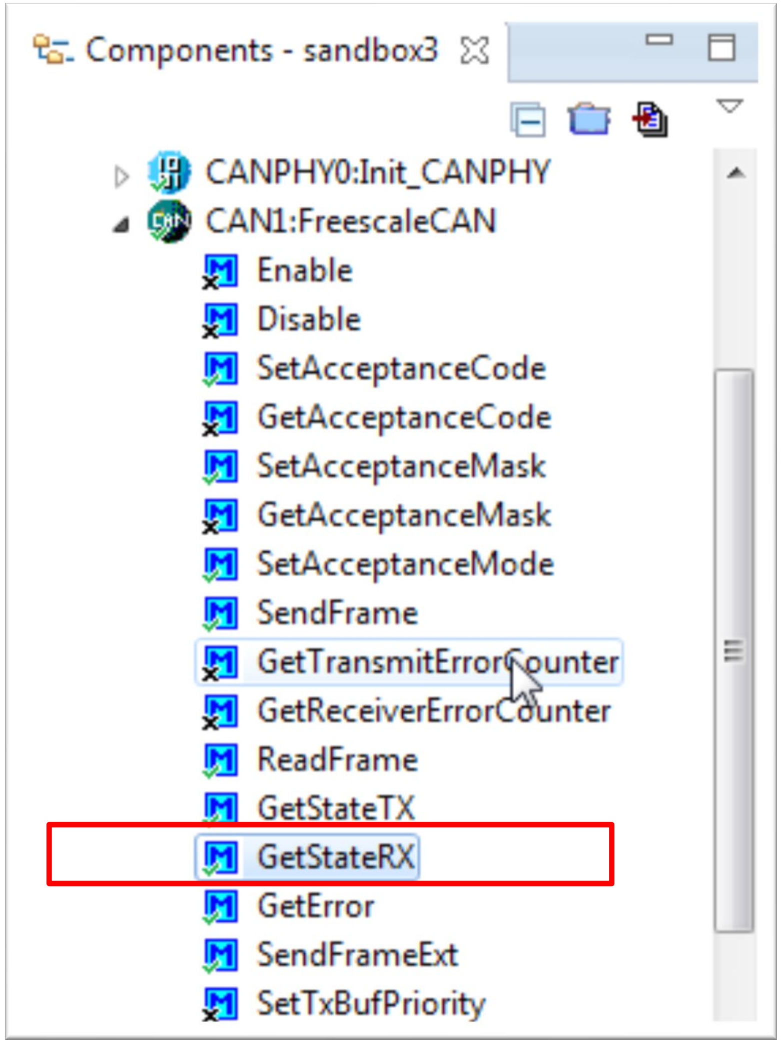
Exercise (1)

Pseudocode

- Inside of main(), create a continuous loop.
- In the loop, check for the arrival of an RX message using the GetStateRX() method
- If a message is received, light the red LED and delay for a while, then clear the red LED and continue
  - **GetStateRX** - Returns a value of the reception complete flag.
    - *ANSIC prototype:* bool GetStateRX(void)
      Return value:bool - The value of the receiver complete flag of the given buffer. Possible values: false - message buffer is empty true - message buffer isn't empty
- **Omission!** Processor Expert forgot to clear the RX buffer flag “CAN0RFLG_RXF”, so add the following:



```
//clear interrupt flag
CAN0RFLG_RXF = 1;
```



Get this RX handler working before you continue.

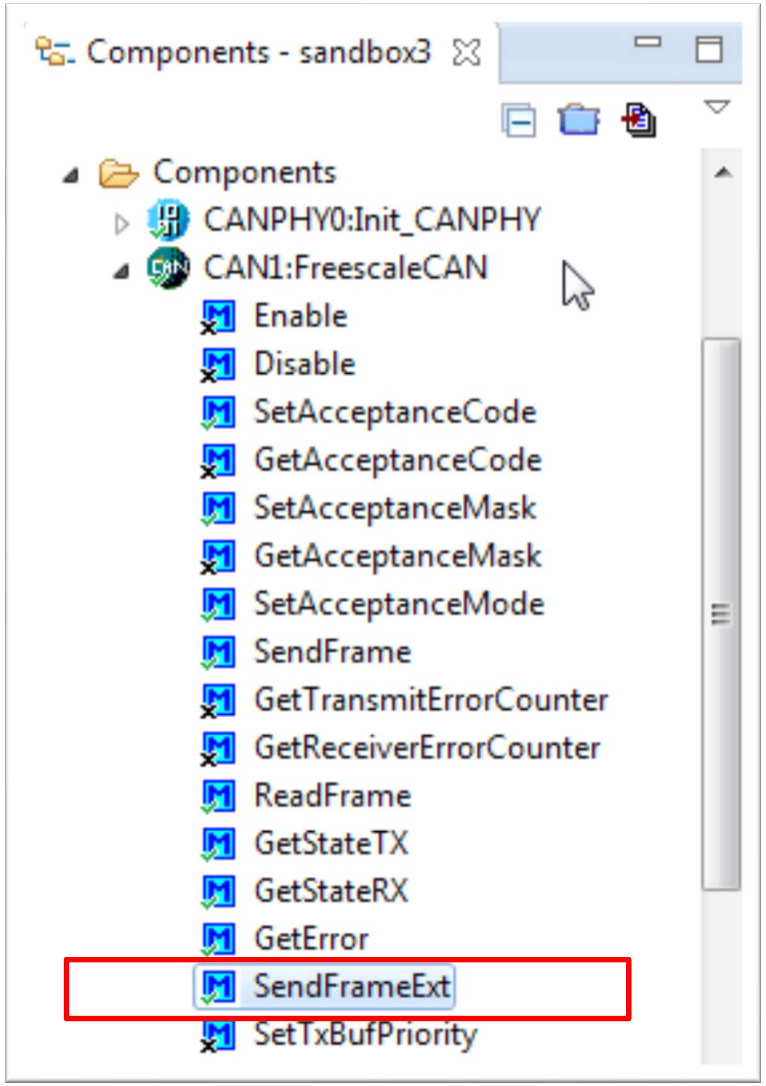
Exercise (2)

Pseudocode

- (within the same continuous loop…)
- If SW3 is pressed, a CAN frame is transmitted, followed be short delay.

1. Use the SW3 GetVal() method to check for a button press
2. When the SW3 is true, use the CAN1 SendFrameExt() method to send a message frame

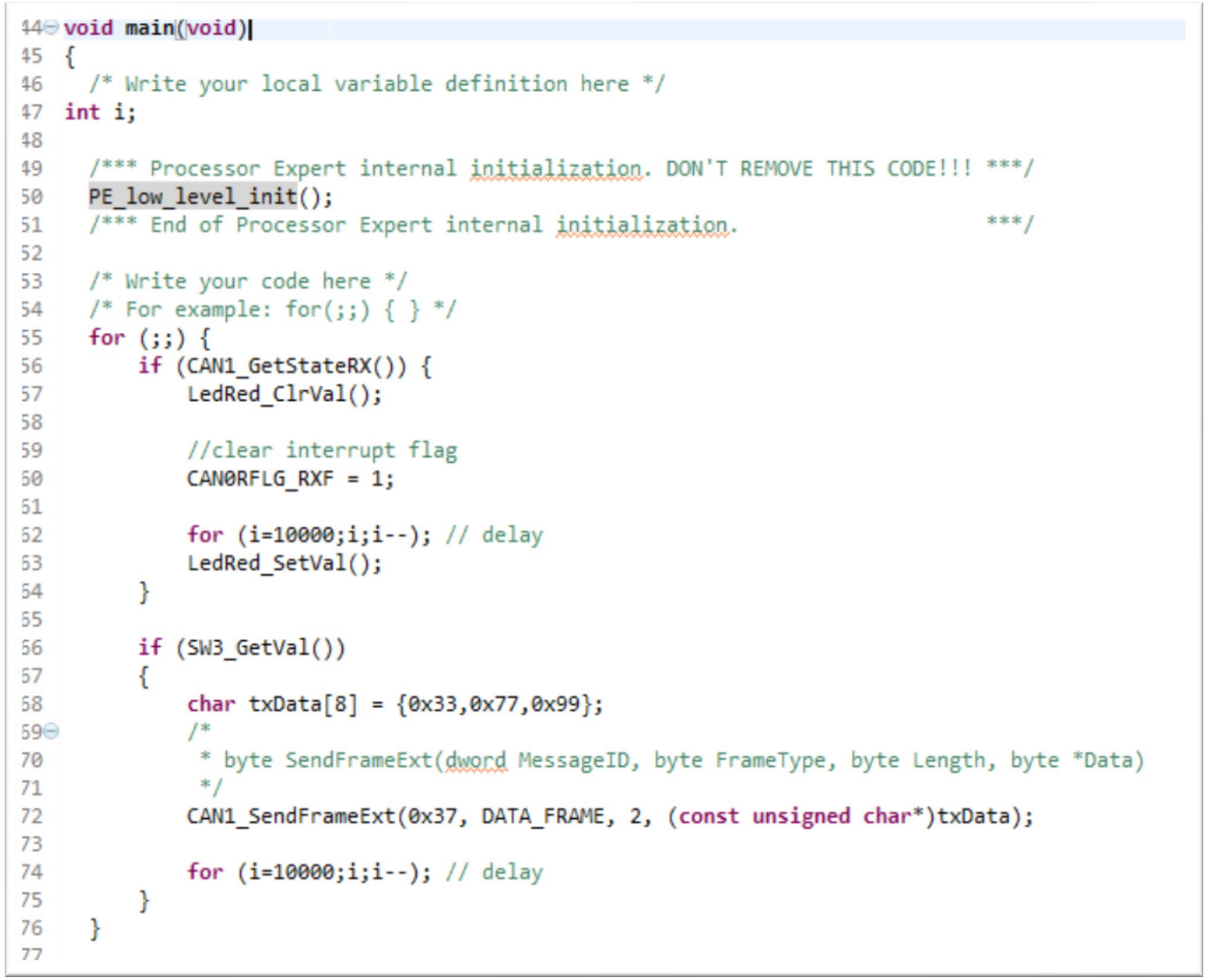
   - **SendFrameExt** - Sends a frame. This method automatically selects a free transmit buffer for data transmission. The user cannot specify a transmit buffer.
   - *ANSIC prototype:* byte SendFrameExt(dword MessageID, byte FrameType, byte Length, byte *Data)



Components - sandbox3

- Components
  - CANPHY0:Init_CANPHY
  - CAN1:FreescaleCAN
    - Enable
    - Disable
    - SetAcceptanceCode
    - GetAcceptanceCode
    - SetAcceptanceMask
    - GetAcceptanceMask
    - SetAcceptanceMode
    - SendFrame
    - GetTransmitErrorCounter
    - GetReceiverErrorCounter
    - ReadFrame
    - GetStateTX
    - GetStateRX
    - GetError
    - SendFrameExt
    - SetTxBufPriority

Coding Solution



```c
void main(void)
{
  /* Write your local variable definition here */
int i;

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  /* Write your code here */
  /* For example: for(;;) { } */
  for (;;) {
      if (CAN1_GetStateRX()) {
          LedRed_ClrVal();

          //clear interrupt flag
          CAN0RFLG_RXF = 1;

          for (i=10000;i;i--); // delay
          LedRed_SetVal();
      }

      if (SW3_GetVal())
      {
          char txData[8] = {0x33,0x77,0x99};
          /*
           * byte SendFrameExt(dword MessageID, byte FrameType, byte Length, byte *Data)
           */
          CAN1_SendFrameExt(0x37, DATA_FRAME, 2, (const unsigned char*)txData);

          for (i=10000;i;i--); // delay
      }
  }
```

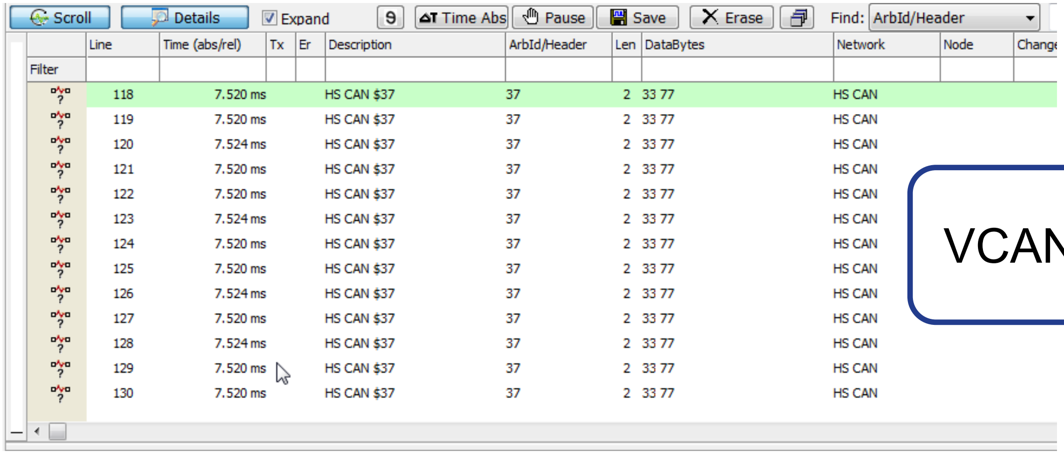
CAN Communication Project 1

Transmission:

1. Press Push button 2 to send the message with ID: 0xAA and data: 0x33, 0x44
2. Press Push button 3 to send the message with ID: 0x55 and data: 0x11, 0x22

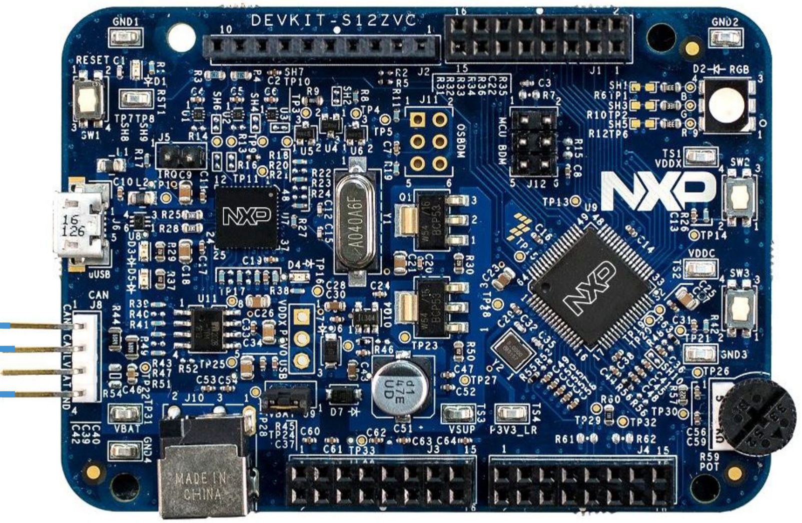
Reception:

1. If any message with ID = 0x33 is received green led will turn on
2. If any message with ID = 0x11 is received blue led will turn on



	Line	Time (abs/rel)	Tx	Er	Description	ArbId/Header	Len	DataBytes	Network	Node	Chang
Filter											
	118	7.520 ms			HS CAN $37	37	2	33 77	HS CAN		
	119	7.520 ms			HS CAN $37	37	2	33 77	HS CAN		
	120	7.524 ms			HS CAN $37	37	2	33 77	HS CAN		
	121	7.520 ms			HS CAN $37	37	2	33 77	HS CAN		
	122	7.520 ms			HS CAN $37	37	2	33 77	HS CAN		
	123	7.524 ms			HS CAN $37	37	2	33 77	HS CAN		
	124	7.520 ms			HS CAN $37	37	2	33 77	HS CAN		
	125	7.520 ms			HS CAN $37	37	2	33 77	HS CAN		
	126	7.524 ms			HS CAN $37	37	2	33 77	HS CAN		
	127	7.520 ms			HS CAN $37	37	2	33 77	HS CAN		
	128	7.524 ms			HS CAN $37	37	2	33 77	HS CAN		
	129	7.520 ms			HS CAN $37	37	2	33 77	HS CAN		
	130	7.520 ms			HS CAN $37	37	2	33 77	HS CAN		

VCAN4



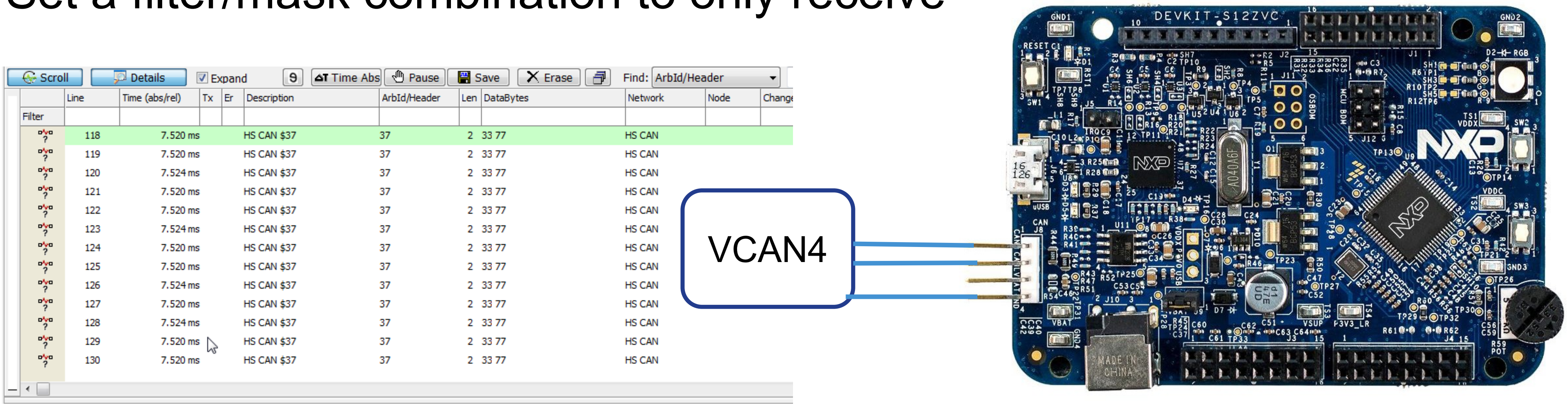
CAN Communication Project 2

Transmission:

1. Create an ADC peripheral to read pin AN1 (potentiometer)
2. Create a Timer peripheral with periodic interrupt
3. On timer interrupt, convert the voltage on AN1 and transmit by CAN message

Reception:

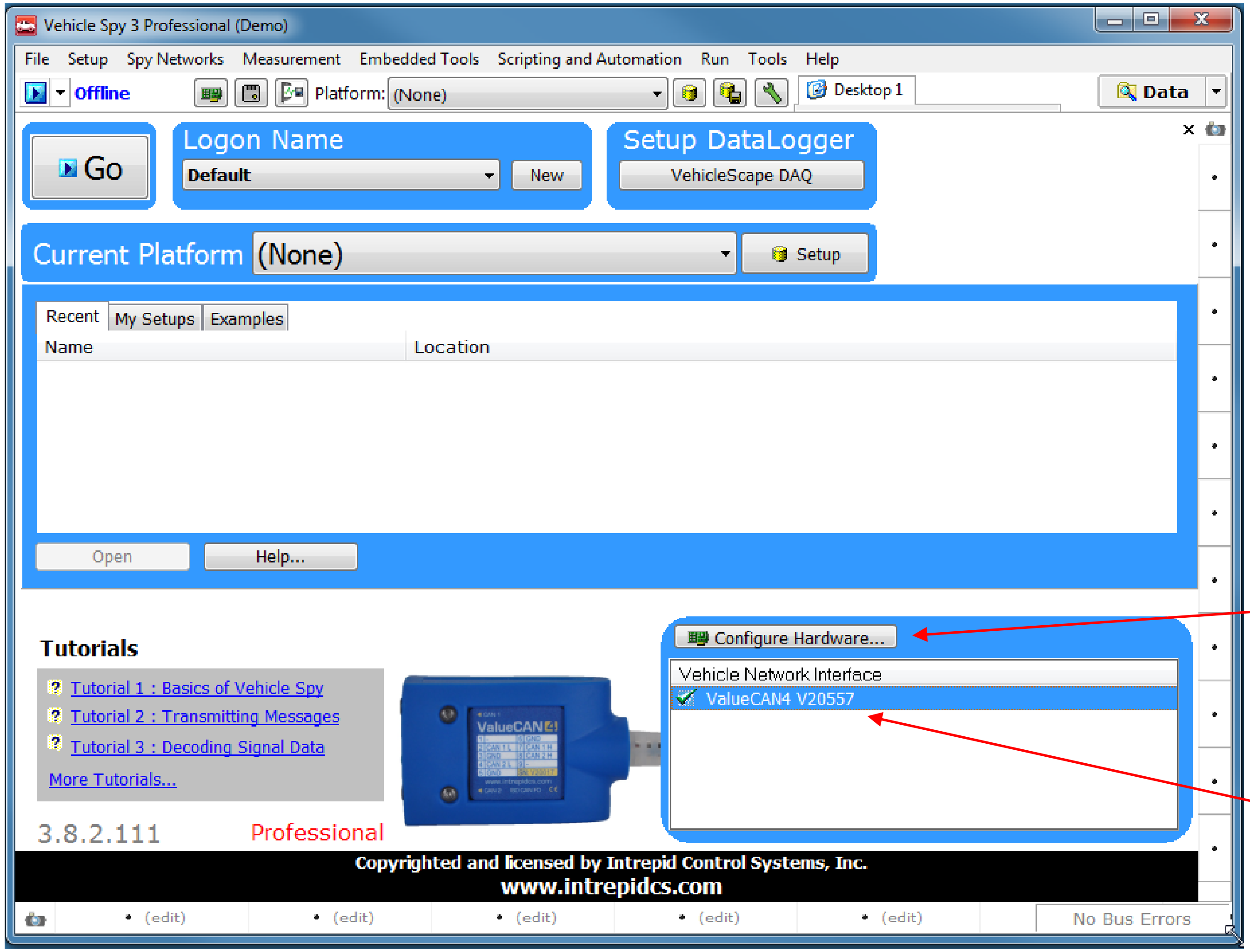
1. Set a filter/mask combination to only receive



	Line	Time (abs/rel)	Tx	Er	Description	ArbId/Header	Len	DataBytes	Network	Node	Change
Filter											
	118	7.520 ms			HS CAN $37	37	2	33 77	HS CAN		
	119	7.520 ms			HS CAN $37	37	2	33 77	HS CAN		
	120	7.524 ms			HS CAN $37	37	2	33 77	HS CAN		
	121	7.520 ms			HS CAN $37	37	2	33 77	HS CAN		
	122	7.520 ms			HS CAN $37	37	2	33 77	HS CAN		
	123	7.524 ms			HS CAN $37	37	2	33 77	HS CAN		
	124	7.520 ms			HS CAN $37	37	2	33 77	HS CAN		
	125	7.520 ms			HS CAN $37	37	2	33 77	HS CAN		
	126	7.524 ms			HS CAN $37	37	2	33 77	HS CAN		
	127	7.520 ms			HS CAN $37	37	2	33 77	HS CAN		
	128	7.524 ms			HS CAN $37	37	2	33 77	HS CAN		
	129	7.520 ms			HS CAN $37	37	2	33 77	HS CAN		
	130	7.520 ms			HS CAN $37	37	2	33 77	HS CAN		

ValueCAN4 & Vehicle Spy

VSPY3

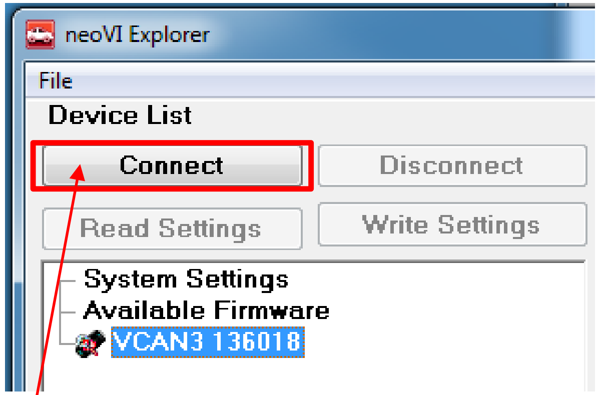


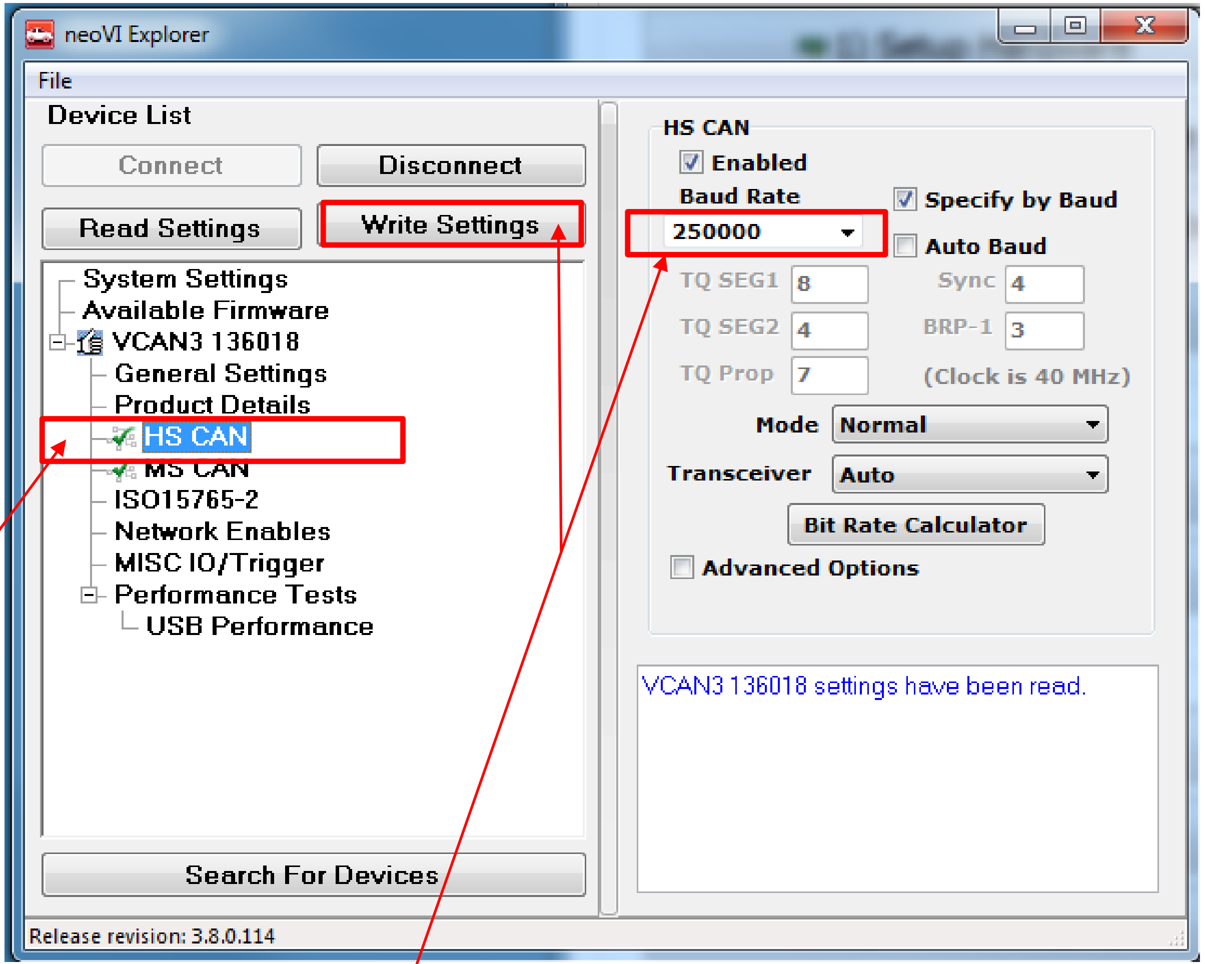
Vehicle Spy 3 Professional (Demo)
File Setup Spy Networks Measurement Embedded Tools Scripting and Automation Run Tools Help
Offline
Platform: (None)
Desktop 1
Data
Go
Logon Name
Default
New
Setup DataLogger
VehicleScape DAQ
Current Platform (None)
Setup
Recent My Setups Examples
Name Location
Open
Help...
Tutorials
Tutorial 1 : Basics of Vehicle Spy
Tutorial 2 : Transmitting Messages
Tutorial 3 : Decoding Signal Data
More Tutorials...
3.8.2.111
Professional
Configure Hardware...
Vehicle Network Interface
ValueCAN4 V20557
Copyrighted and licensed by Intrepid Control Systems, Inc.
www.intrepidcs.com
(edit)
(edit)
(edit)
(edit)
(edit)
No Bus Errors

Configure Hardware (click here)

Detected Hardware

Hardware Configuration





HS CAN

☑ Enabled

Baud Rate: 250000

☑ Specify by Baud

□ Auto Baud

TQ SEG1: 8 Sync: 4

TQ SEG2: 4 BRP-1: 3

TQ Prop: 7 (Clock is 40 MHz)

Mode: Normal

Transceiver: Auto

□ Advanced Options

VCAN3 136018 settings have been read.

Release revision: 3.8.0.114

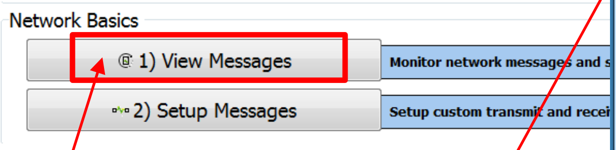
1) Connect to HW

2) Select HS CAN

3) Choose baud rate 250,000 then write-in

4) Close the window

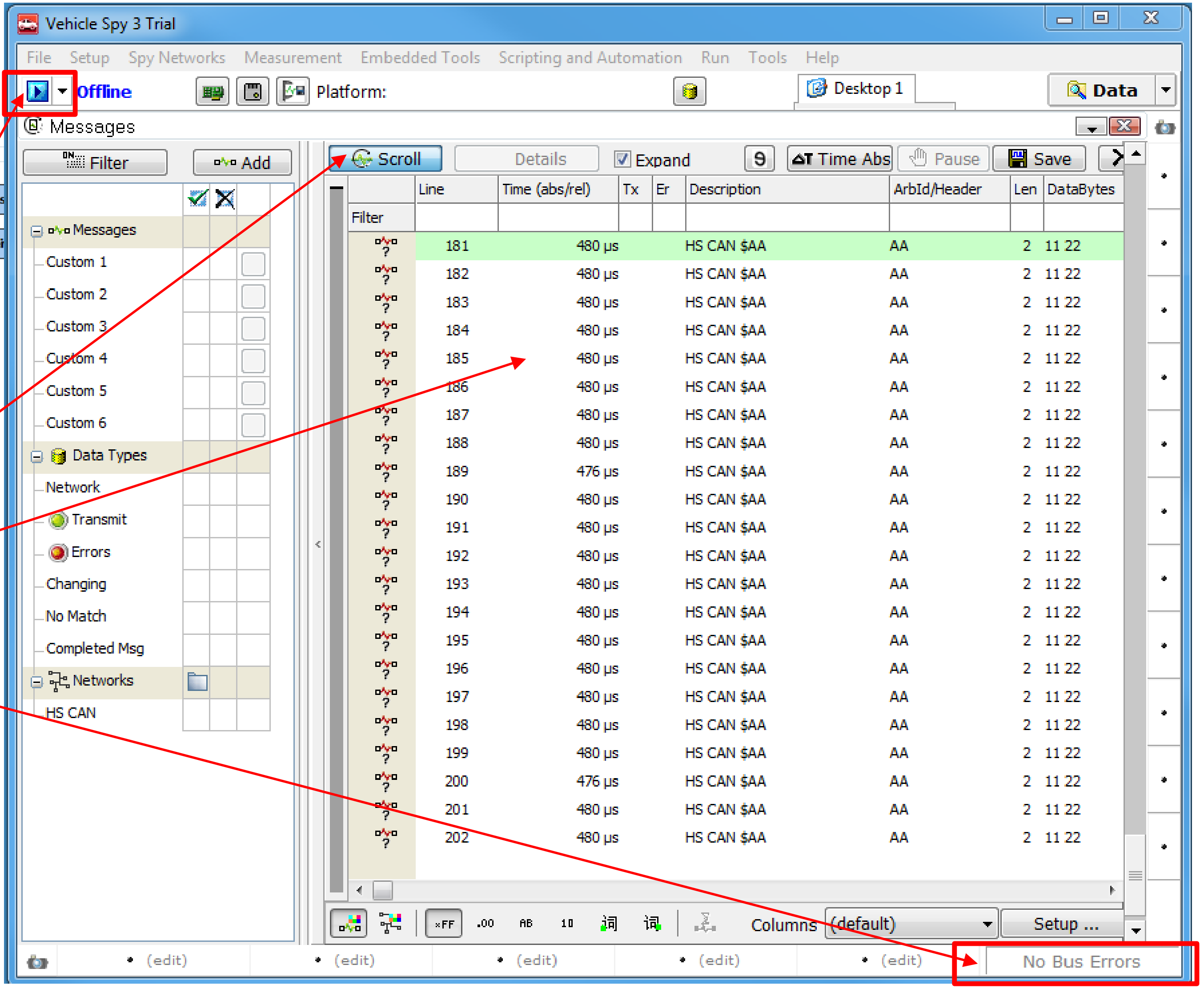
RX Messages



Network Basics

@ 1) View Messages — Monitor network messages and s

2) Setup Messages — Setup custom transmit and recei



Vehicle Spy 3 Trial

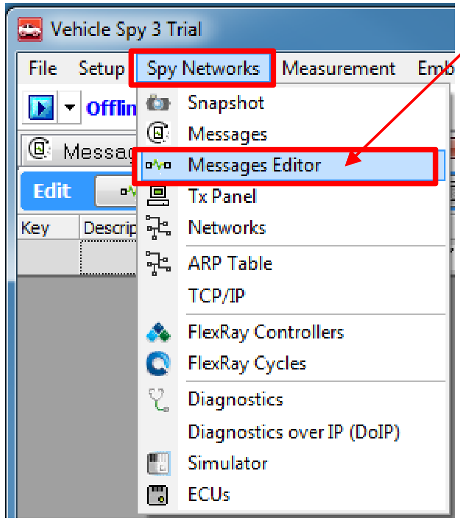
Line	Time (abs/rel)	Tx	Er	Description	ArbId/Header	Len	DataBytes
Filter							
181	480 µs			HS CAN $AA	AA	2	11 22
182	480 µs			HS CAN $AA	AA	2	11 22
183	480 µs			HS CAN $AA	AA	2	11 22
184	480 µs			HS CAN $AA	AA	2	11 22
185	480 µs			HS CAN $AA	AA	2	11 22
186	480 µs			HS CAN $AA	AA	2	11 22
187	480 µs			HS CAN $AA	AA	2	11 22
188	480 µs			HS CAN $AA	AA	2	11 22
189	476 µs			HS CAN $AA	AA	2	11 22
190	480 µs			HS CAN $AA	AA	2	11 22
191	480 µs			HS CAN $AA	AA	2	11 22
192	480 µs			HS CAN $AA	AA	2	11 22
193	480 µs			HS CAN $AA	AA	2	11 22
194	480 µs			HS CAN $AA	AA	2	11 22
195	480 µs			HS CAN $AA	AA	2	11 22
196	480 µs			HS CAN $AA	AA	2	11 22
197	480 µs			HS CAN $AA	AA	2	11 22
198	480 µs			HS CAN $AA	AA	2	11 22
199	480 µs			HS CAN $AA	AA	2	11 22
200	476 µs			HS CAN $AA	AA	2	11 22
201	480 µs			HS CAN $AA	AA	2	11 22
202	480 µs			HS CAN $AA	AA	2	11 22

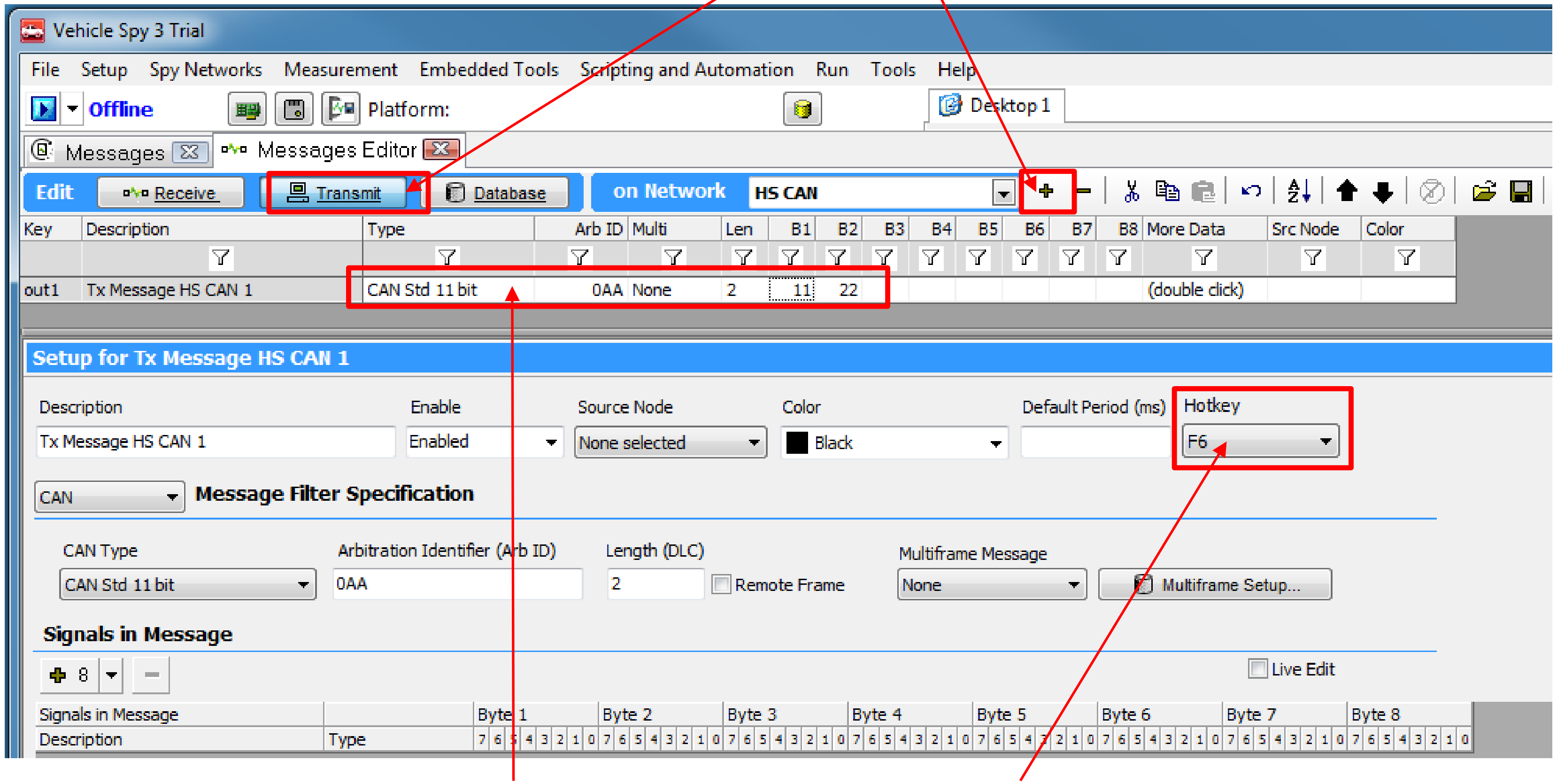
1) Connect to HW
2) Start analyzer
3) Scroll / no-scroll
4) Message view
5) No bus errors

Define TX Messages

1) Launch Message Editor

2) Add a Transmit message





Key	Description	Type	Arb ID	Multi	Len	B1	B2	B3	B4	B5	B6	B7	B8	More Data	Src Node	Color
out1	Tx Message HS CAN 1	CAN Std 11 bit	0AA	None	2	11	22							(double click)		

Description: Tx Message HS CAN 1
Enable: Enabled
Source Node: None selected
Color: Black
Default Period (ms):
Hotkey: F6

CAN

CAN Type: CAN Std 11 bit
Arbitration Identifier (Arb ID): 0AA
Length (DLC): 2
□ Remote Frame
Multiframe Message: None

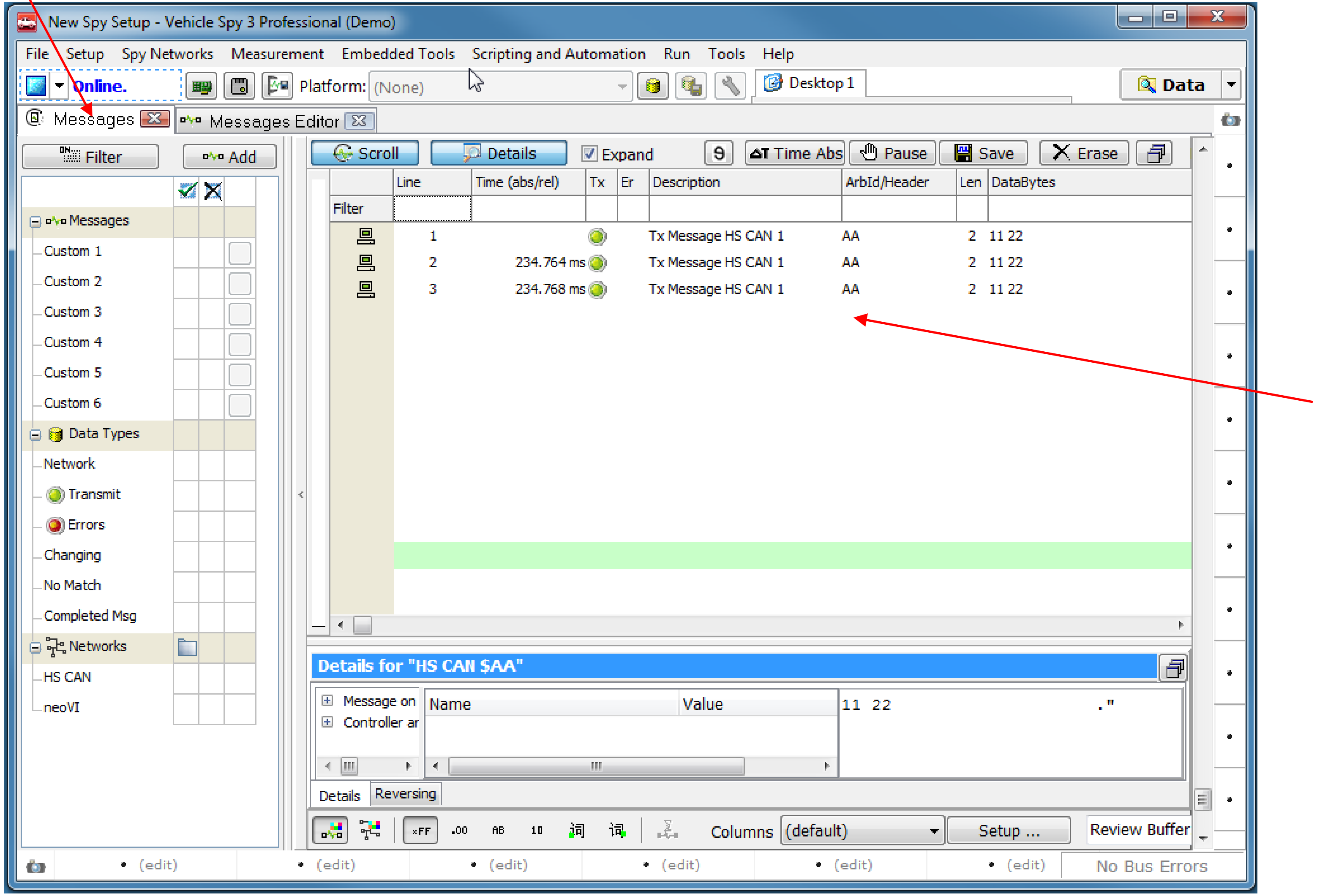
☐ Live Edit

Signals in Message Description	Type	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
		7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0

3) Set message parameters
Type, ID, Len, data bytes

4) F6 hotkey

Return to Messages Tab



Line	Time (abs/rel)	Tx	Er	Description	ArbId/Header	Len	DataBytes
1		●		Tx Message HS CAN 1	AA	2	11 22
2	234.764 ms	●		Tx Message HS CAN 1	AA	2	11 22
3	234.768 ms	●		Tx Message HS CAN 1	AA	2	11 22

Details for "HS CAN $AA"

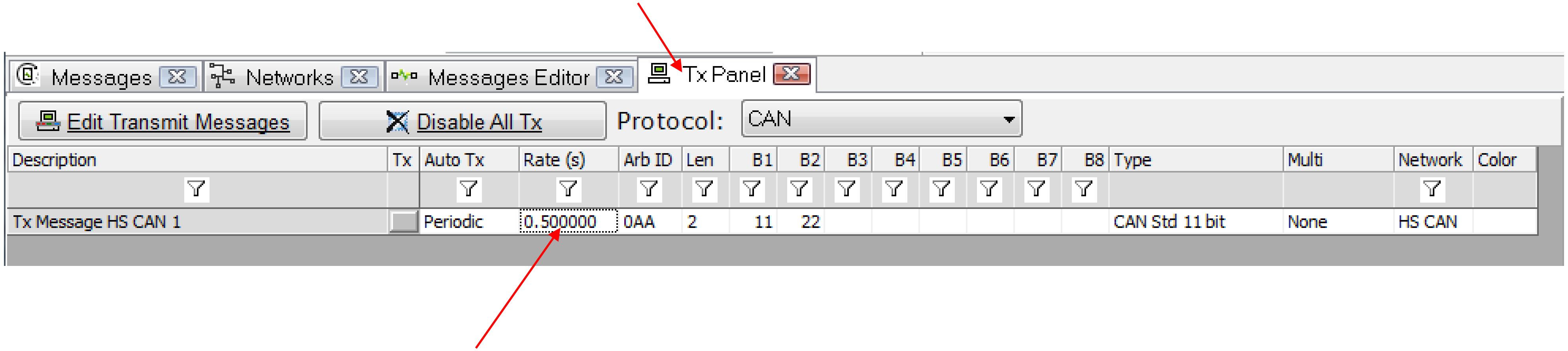
Name	Value

11 22 ."

Columns: (default)

Press F6 on keyboard and see LED blink on the EVB

VSPY3 – Periodic Transmission



Protocol: CAN

Description	Tx	Auto Tx	Rate (s)	Arb ID	Len	B1	B2	B3	B4	B5	B6	B7	B8	Type	Multi	Network	Color
Tx Message HS CAN 1	□	Periodic	0.500000	0AA	2	11	22							CAN Std 11 bit	None	HS CAN	

1. Open menu item [Spy Networks → TX Panel]
2. Set a periodic rate (slow enough that you can see the LED blink) and hit ENTER
3. Return to Message tab and view CAN traffic…

Thank You for Participating!

SECURE CONNECTIONS
FOR A SMARTER WORLD