Getting Started with S32K Hands-on Workshop

Allen Willson

FAE

Automotive

October 2019 | Session #AMF-AUT-T3875

SECURE CONNECTIONS FOR A SMARTER WORLD

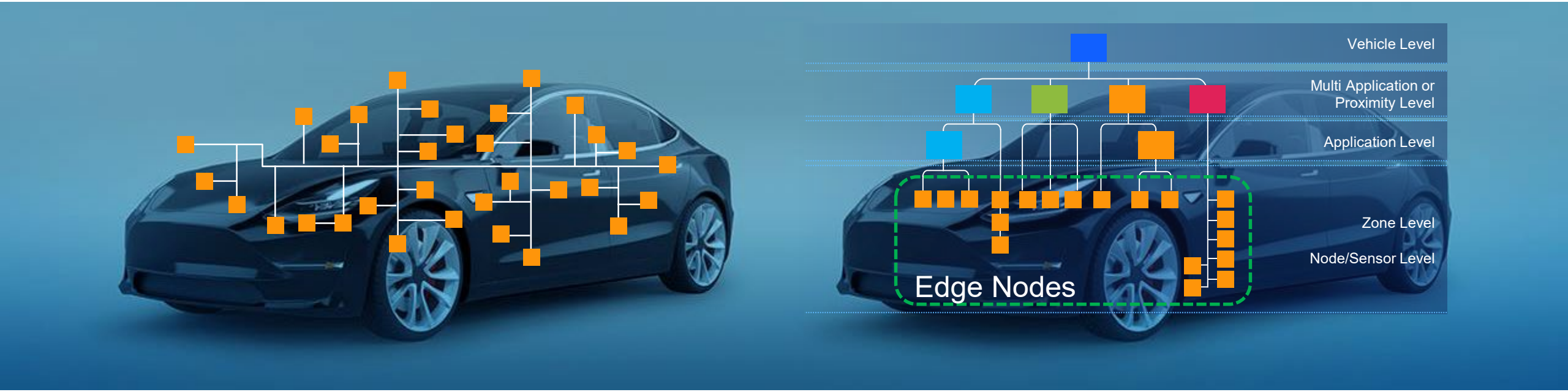
Agenda

- Brief S32K Product & Roadmap Overview
- Introduction (“Cookbook App Note”)
  1. Hello World (GPIO) – Hands-On
  2. Hello World + Clocks
  3. Hello World + Interrupts – Hands-On
  4. DMA
  5. Timed I/O (FTM)
  6. ADC
  7. UART
  8. SPI
  9. Classic CAN – Hands-On
  10. CAN FD
- Appendix: S32 Design Studio Blank Project Steps

Product Overview & Roadmap

Introduction

Solutions for Edge Nodes – Today and Tomorrow



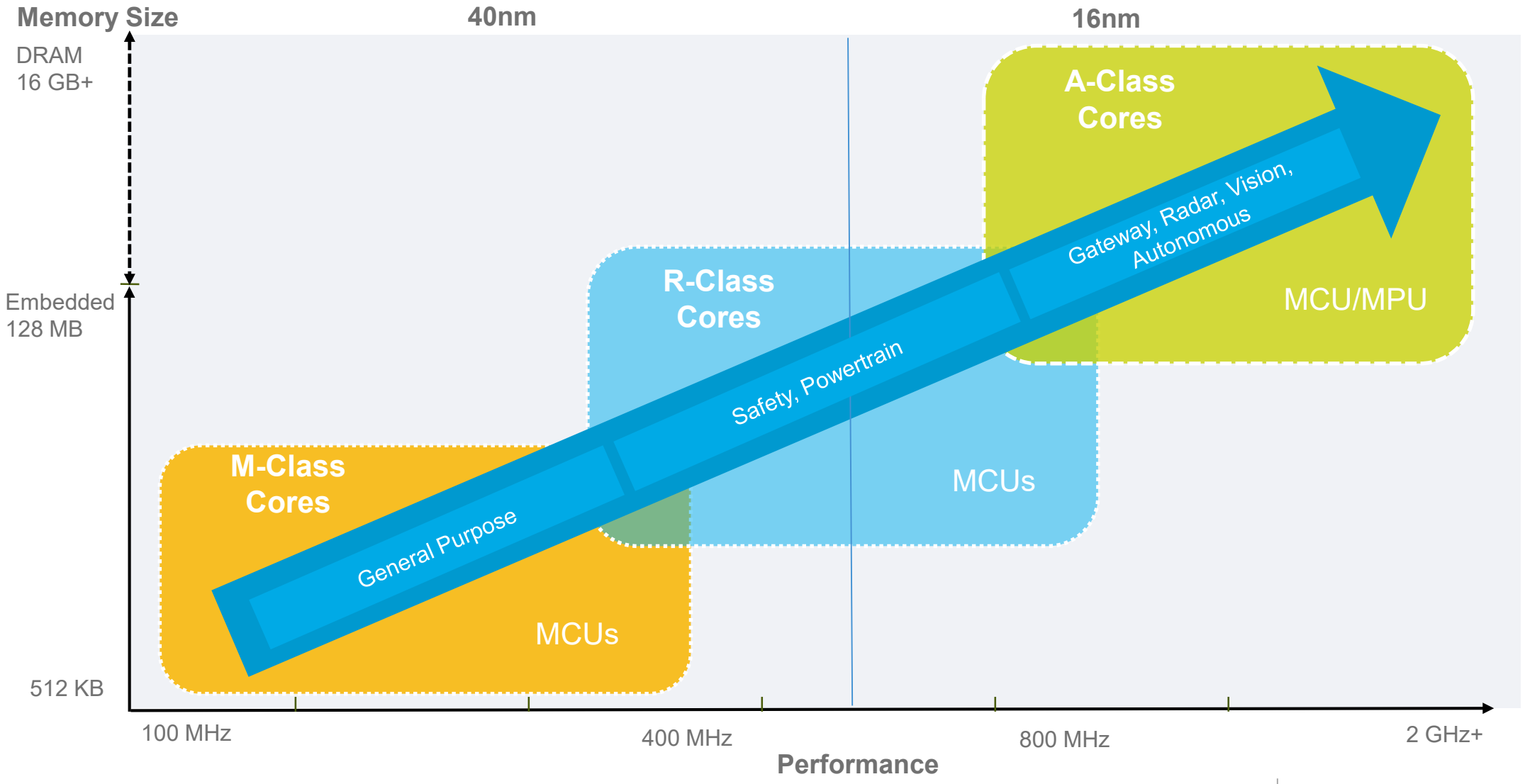
Today

Distributed vehicle architectures
Incompatible silicon and software
Security and over-the-air update challenges
Inefficient development
Not easily upgraded or scaled

Tomorrow

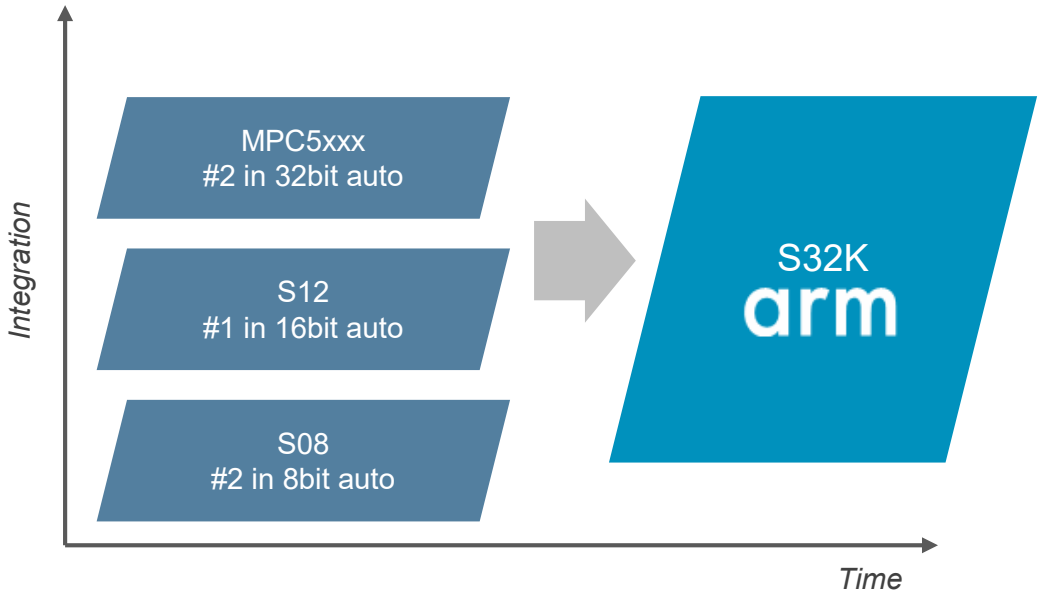
High performance domain architectures
Greater network capacity
Secure, safe over-the-air updates
Efficient to develop
Upgradable and scalable platform, future proof

Microcontroller Roadmap & Scalability



S32K Value Proposition

ARM Cortex for Scalability



Future-proof Features

S32K1 / KEA Compatibility

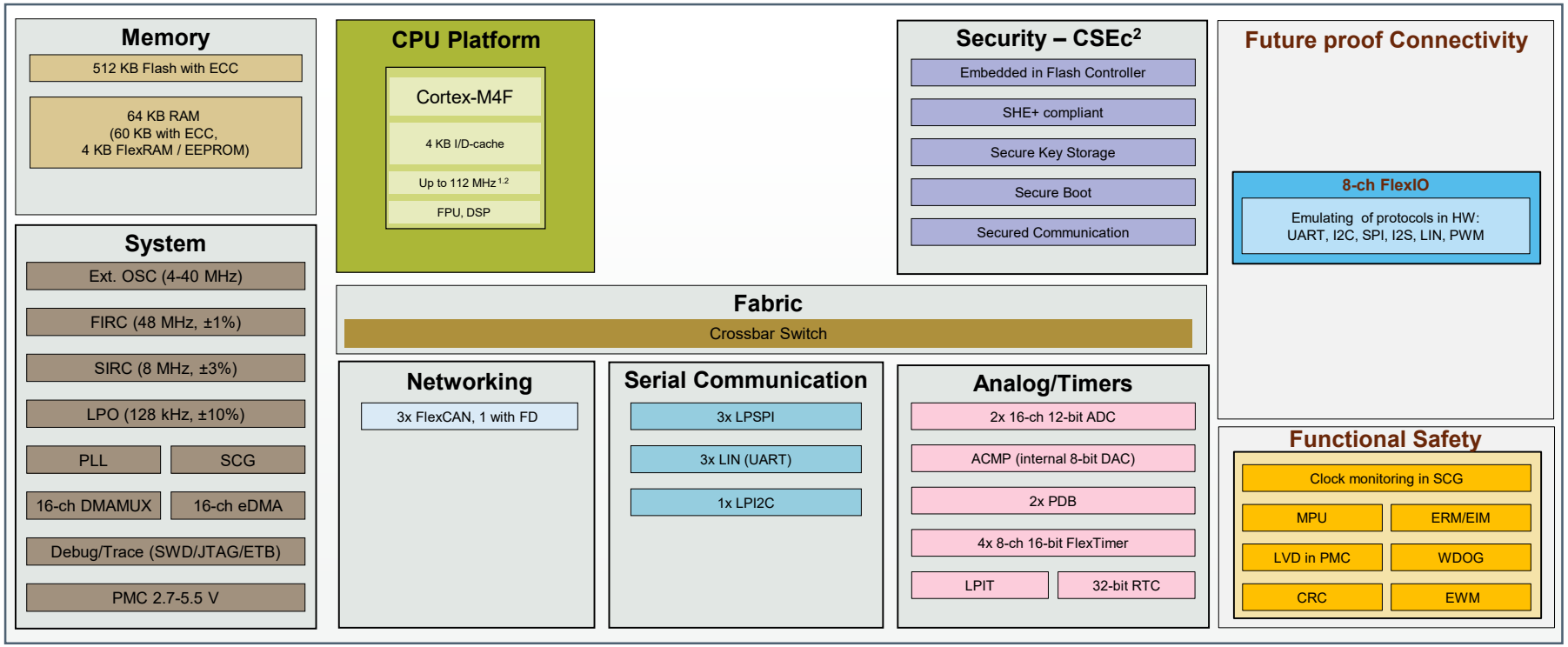
Pin Compatibility: ✓
- Within S32K1xx product series
- Similar pinout as **KEA** products

IP Compatibility: ✓
- With MPC55xx/MPC56xxx/MPC57xxx product series: FlexCAN, ACMP, eDMA, QuadSPI
- With KEA products: FlexTimer, IIC, LSPI, UART, CRC, FlexIO

Flash	Pin Count								
	16/24	32	48	64	80	100 LQFP	100 BGA	144	176
2M							S32K148	S32K148	S32K148
1M				S32K146		S32K146	S32K146	S32K146	
512K				S32K144		S32K144	S32K144		
256K			S32K142 * S32K118	S32K142 S32K118		S32K142			
128K		S32K116	S32K116	KEAZ128	KEA128				
64K		KEAZN64		KEAZ(N)64	KEAZ64				
32K		KEAZN32		KEAZN32					
16K		KEAZN16		KEAZN16					
8K	KEAZN8								

*: S32K142 48LQFP is for development only

S32K144: ASIL B 512K General Purpose MCU



Specifications:

- **Cores:** ARM Cortex-M4F @112 MHz max
- **Memory:** 512 KB Flash, 64 KB RAM (60 KB with ECC, 4 KB FlexRAM/EEPROM)
- **Temp Range:** Ta -40 to 125°C (Tj=135°C)
- **Power Supplies:** 2.7-5.5 V
- **Packaging:**10 x 10 mm, 0.5 mm pitch 64 LQFP (up to 58 usable pins). 11 x 11 mm, 1 mm pitch 100 MapBGA.(up to 89 usable pins). 14 x 14 mm, 0.5 mm pitch 100 LQFP (up to 89 usable pins)

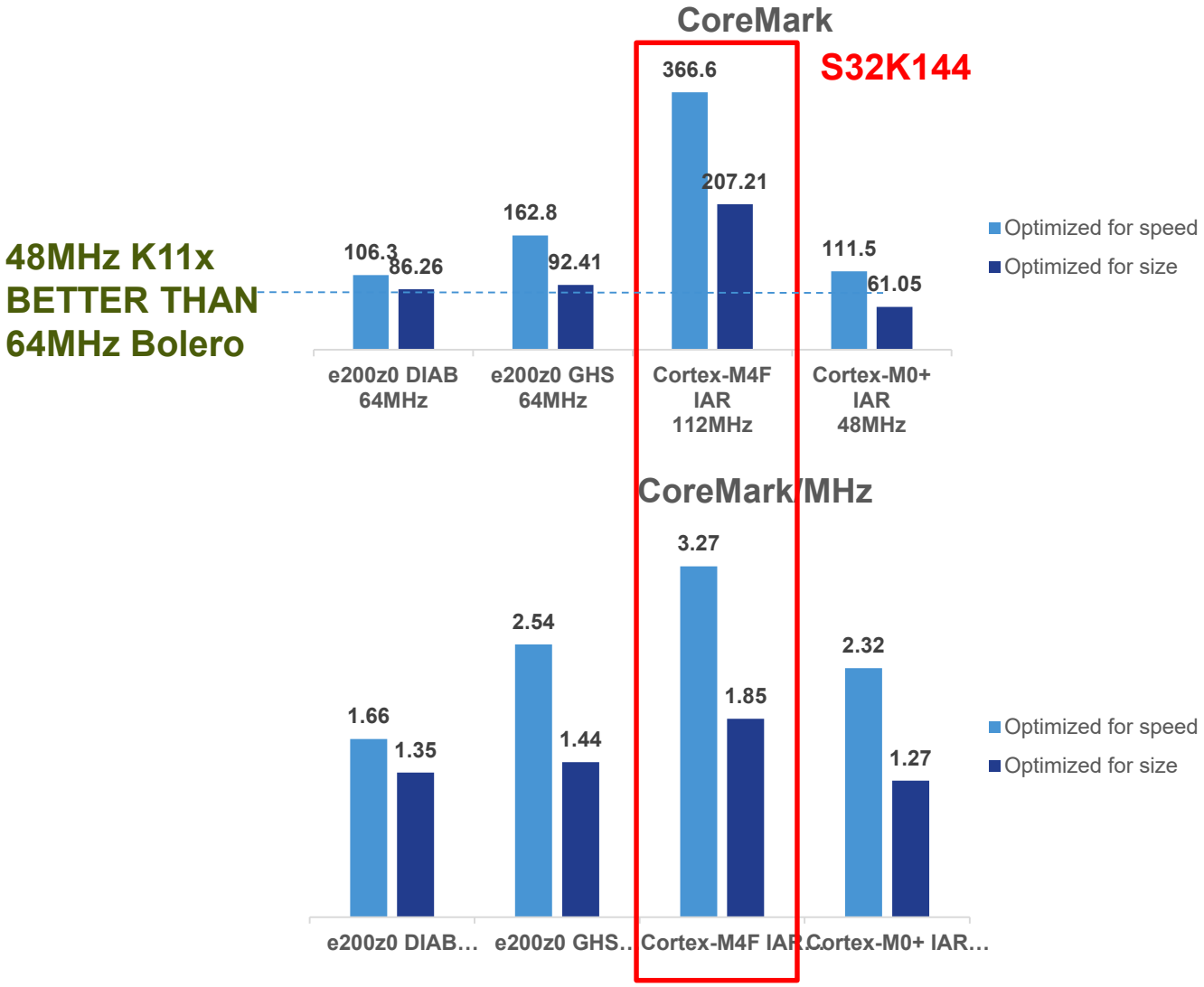
Key Features:

- **High Performance:** Powerful ARM Cortex-M4F core
- **Advanced Automotive Communication:** CAN FD
- **Functional Safety:** Developed as per ISO 26262 with target ASIL B
- **Security:** HW security engine (SHE+ compliant)
- **Low Power**: Low leakage tech. Best in class STOP current: 25-40 uA (device dependent)
- **Full solution offering:** AUTOSAR, SDK, Design Studio IDE

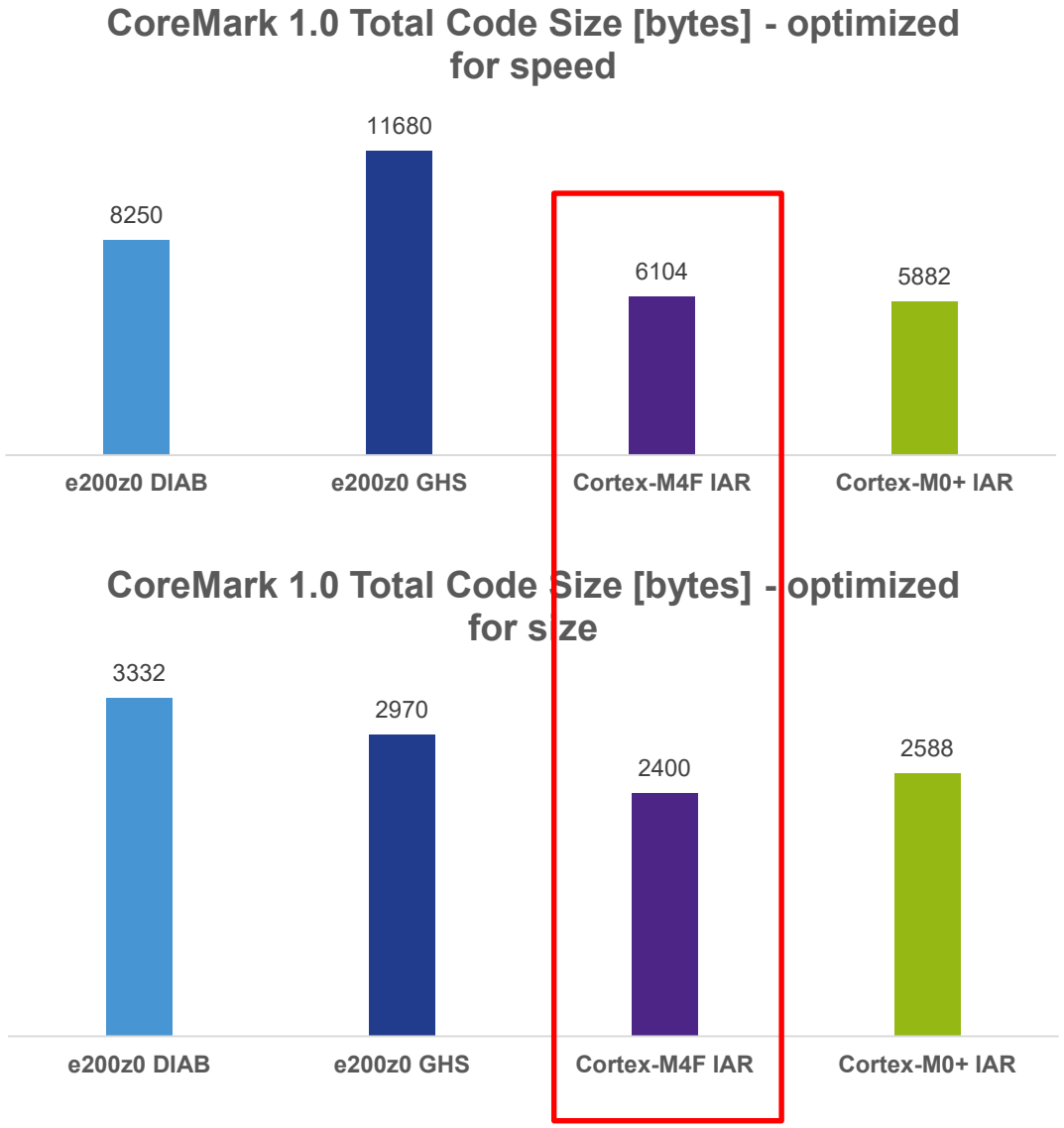
Footnote:

1. 112MHz not valid with M temp (125C).
2. Write or erase access to security (CSEc) or EEPROM is allowed only when device operating in RUN mode (up to 80MHz). No write or erase access to security and EEPROM allowed when device running at HSRUN mode (112MHz).

S32K Superior Performance



S32K Superior Code Density

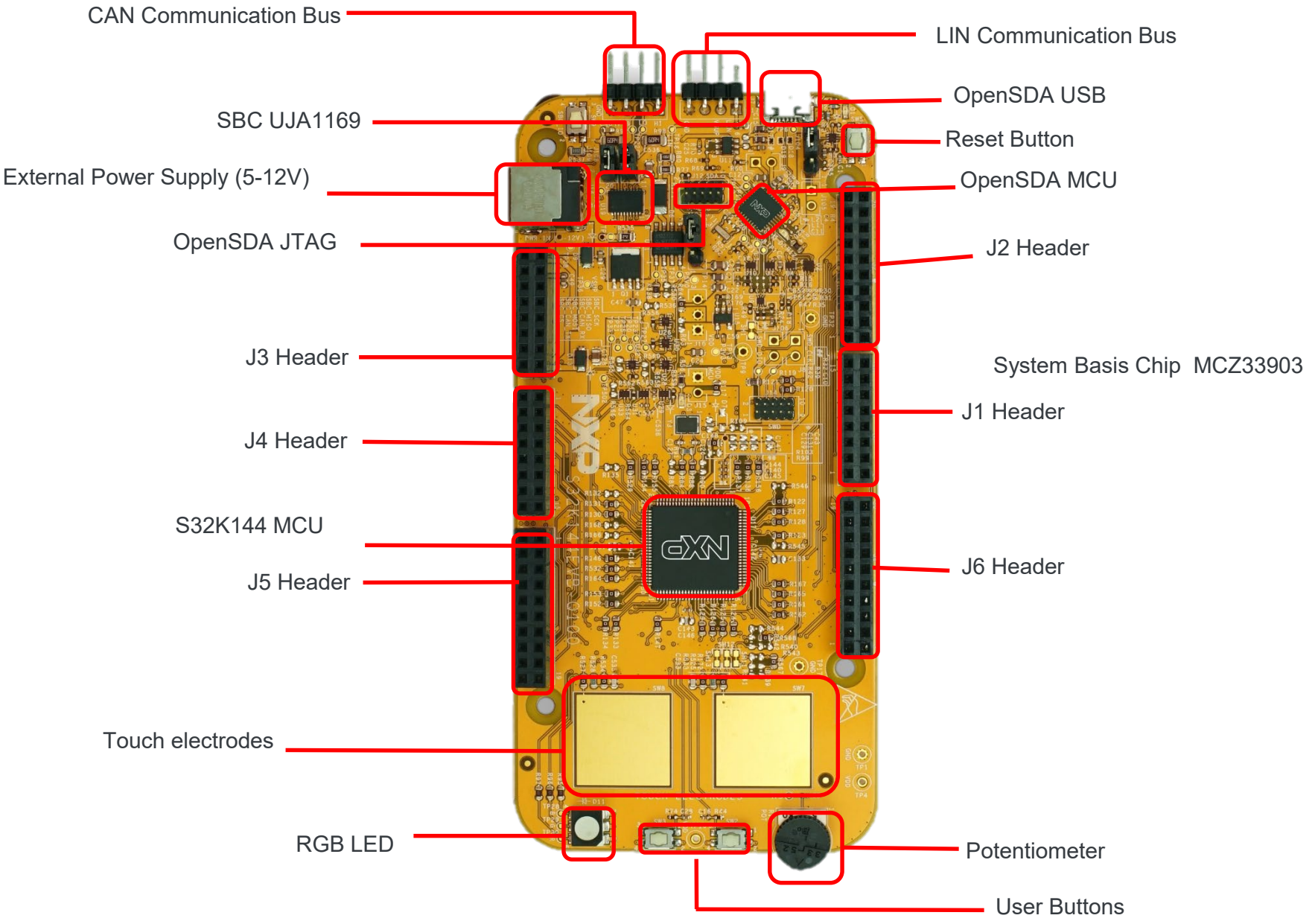


- Higher speed leads to better cache efficency
- More space for application code

Industry Leading Low Power Performance

	Ta (C)	VLPS (uA)	VLPR (mA)	Stop 1 (mA)	Run (mA)
S32K116	25 (typ)	26	1.9	7	tbd
S32K118	25 (typ)	26	1.9	7	tbd
S32K142	25 (typ)	29	1.17	6.4	37.5
S32K144	25 (typ)	29.8	1.48	7	39.6
	105 (typ)	256.3	1.8	7.8	40.5
	125 (max)	1492	3.18	12.2	46
S32K146	25 (typ)	40	5	15	tbd
S32K148	25 (typ)	38	2.17	8.5	57.7

S32K144 EVB


CAN Communication Bus
LIN Communication Bus
OpenSDA USB
SBC UJA1169
Reset Button
External Power Supply (5-12V)
OpenSDA MCU
OpenSDA JTAG
J2 Header
J3 Header
System Basis Chip MCZ33903
J1 Header
J4 Header
S32K144 MCU
J6 Header
J5 Header
Touch electrodes
RGB LED
Potentiometer
User Buttons

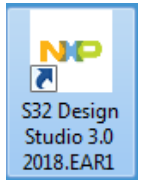
HMI Mapping

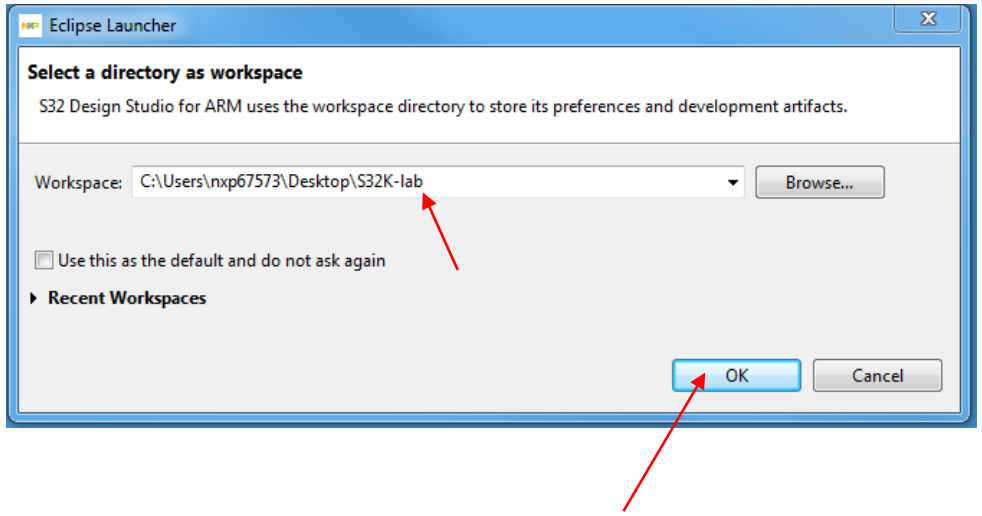
Component	S32K144
Red LED	PTD15 (FTM0 CH0)
Blue LED	PTD0(FTM0 CH2)
Green LED	PTD16(FTM0 CH1)
Potentiometer	PTC14 (ADC0_SE12)
SW2	PTC12
SW3	PTC13
OpenSDA UART TX	PTC7(LPUART1_TX)
OpenSDA UART RX	PTC6(LPUART1_RX)
CAN TX	PTE5(CAN0_TX)
CAN RX	PTE4 (CAN0_RX)
LIN TX	PTD7(LPUART2_TX)
LIN RX	PTD6 (LPUART2_RX)
SBC_SCK	PTB14 (LPSPI1_SCK)
SBC_MISO	PTB15(LPSPI1_SIN)
SBC_MOSI	PTB16(LPSPI1_SOUT)
SBC_CS	PTB17(LPSPI1_PCS3)

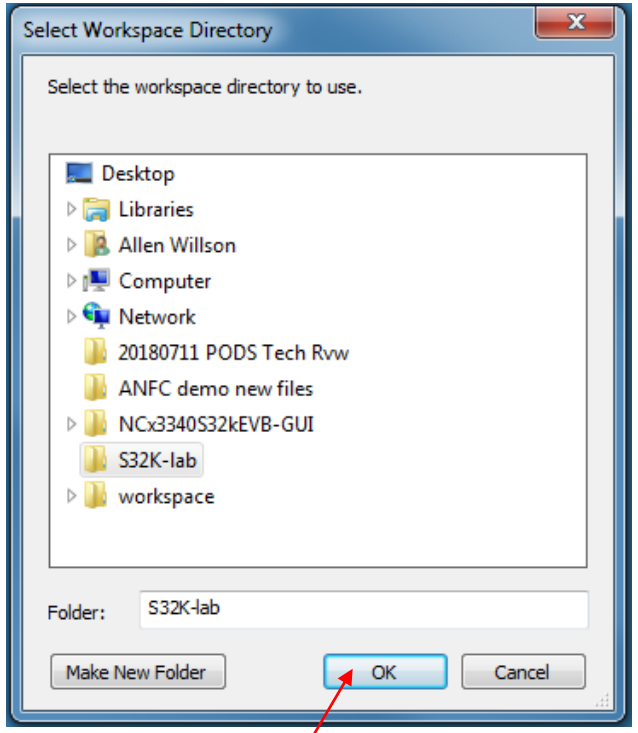
0. Start S32 Design Studio and Import the Labs

Open S32 Design Studio & Select a Workspace

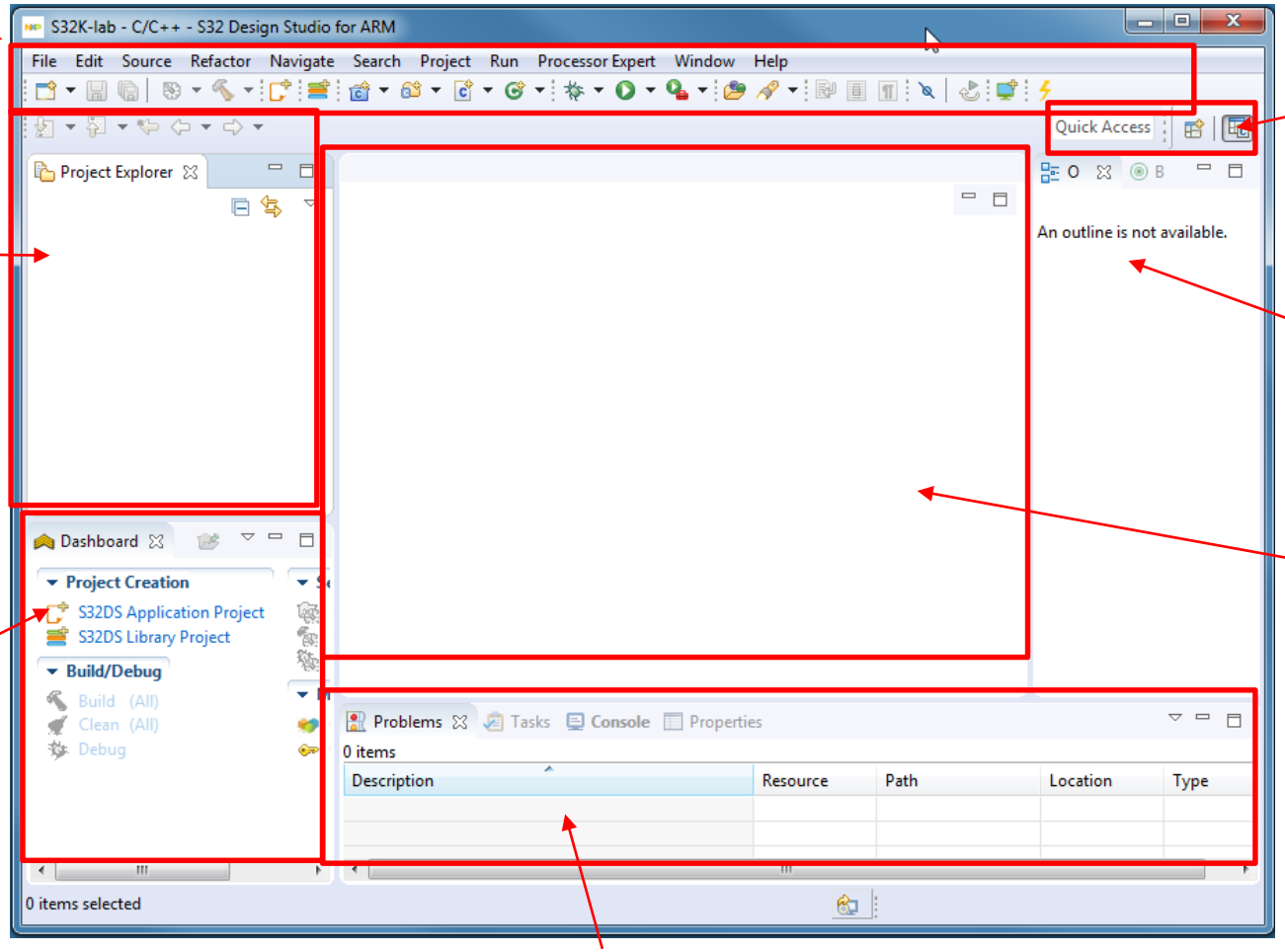
1. Double click the S32 Design Studio 3.0 icon on the desktop
2. Create a folder on the desktop, to use as a workspace
3. Click “OK”





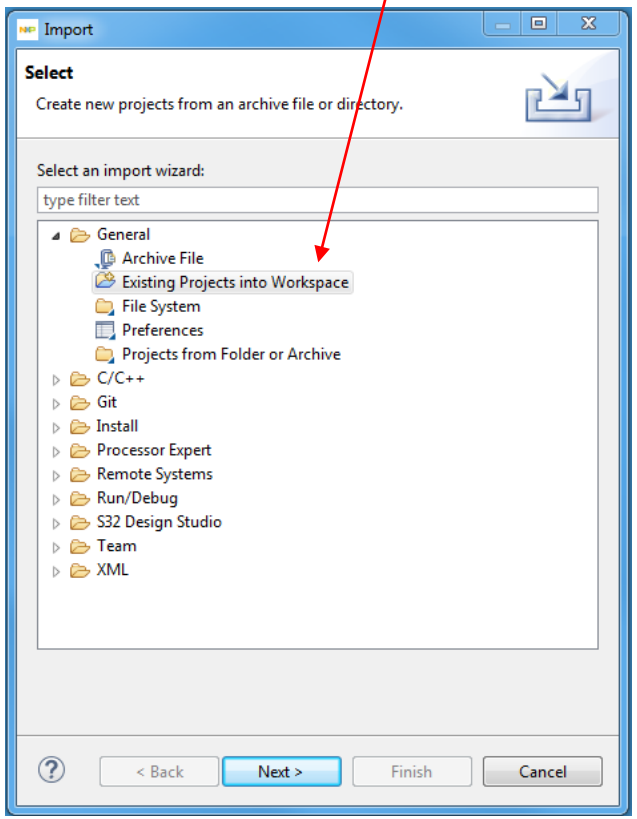


Default View

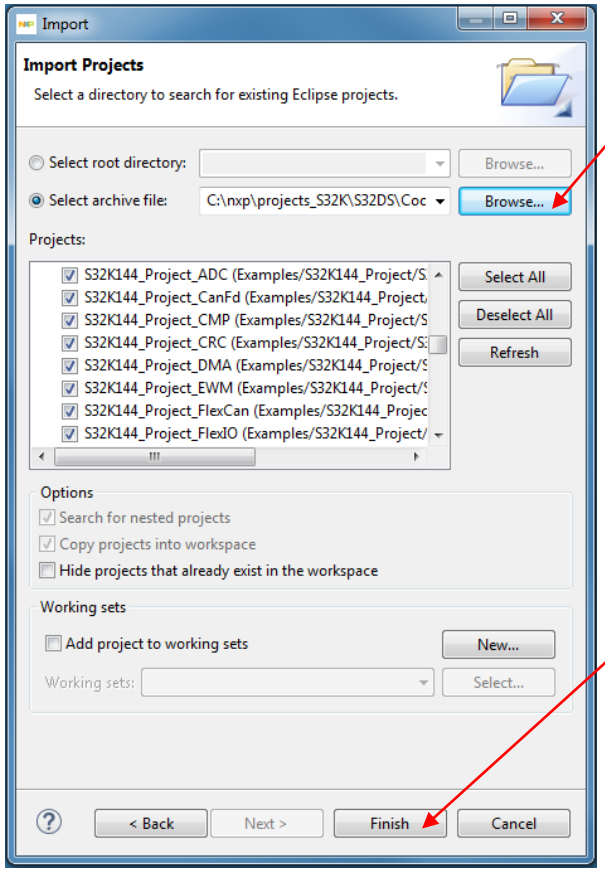


Import the Labs

1. File→ Import→ General→ Existing Projects into Workspace→ Next

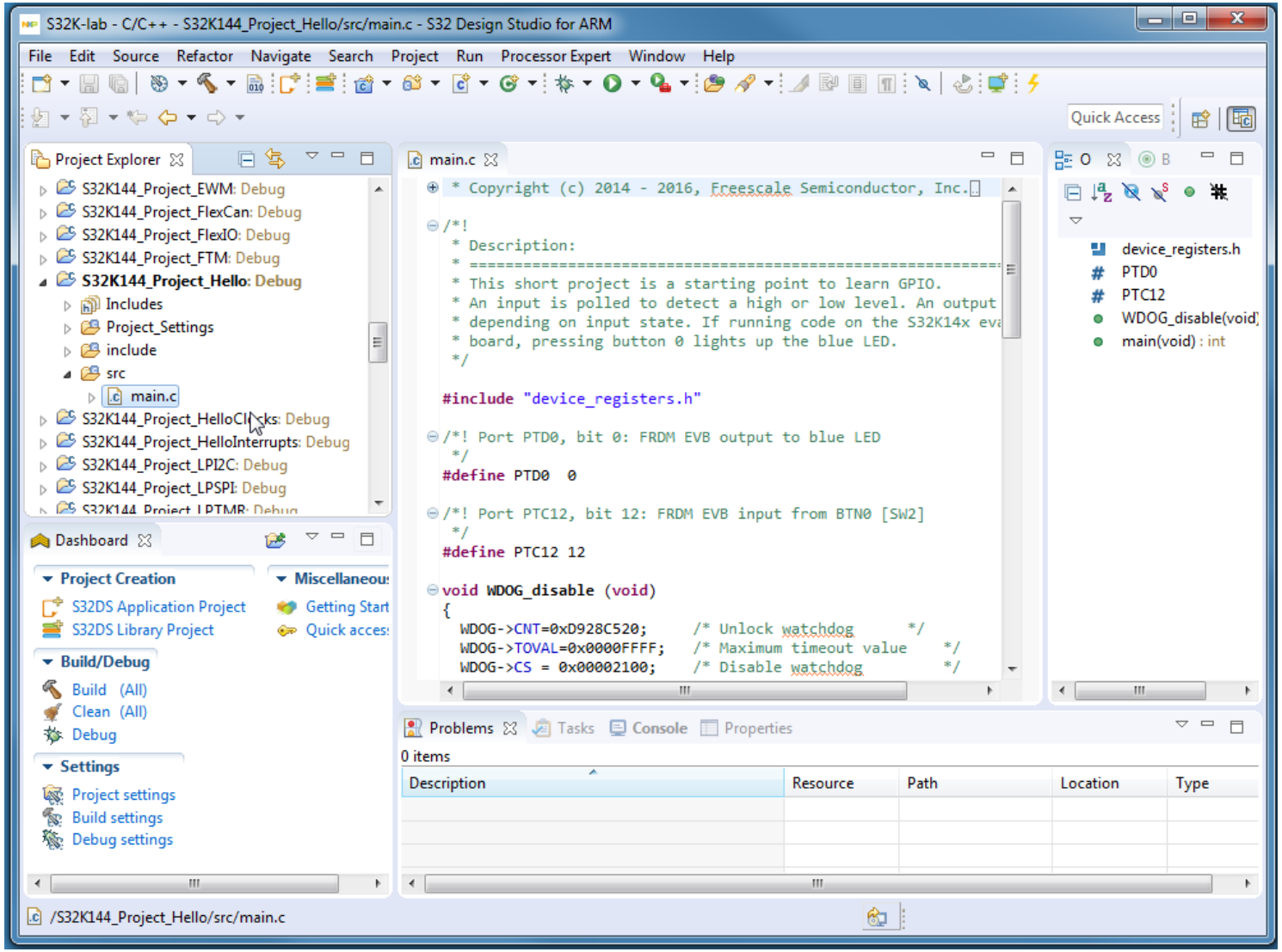


2. Select Archive File→ Browse and select file



3. Finish

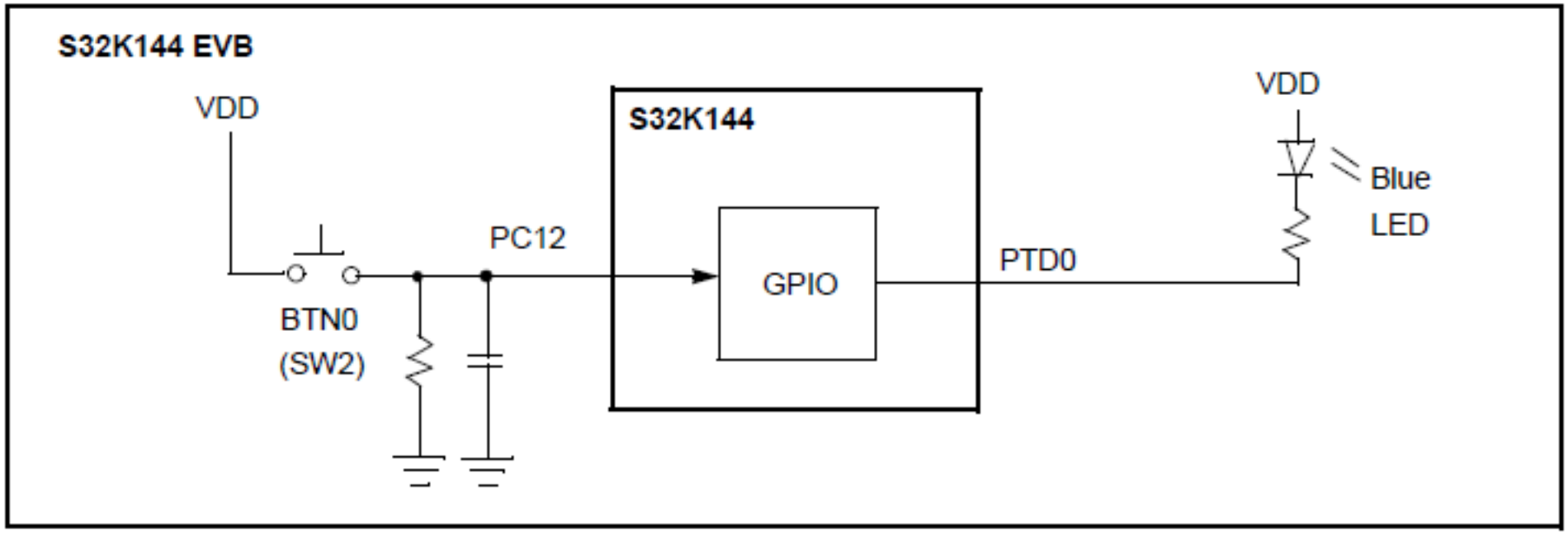
Take a Tour of S32 Design Studio



1. Hello World (GPIO)

1. Hello World: Introduction

Summary: A GPIO input is continuously polled to detect a high or low level. A GPIO output is set corresponding to the level.



1. Hello World: Key Points

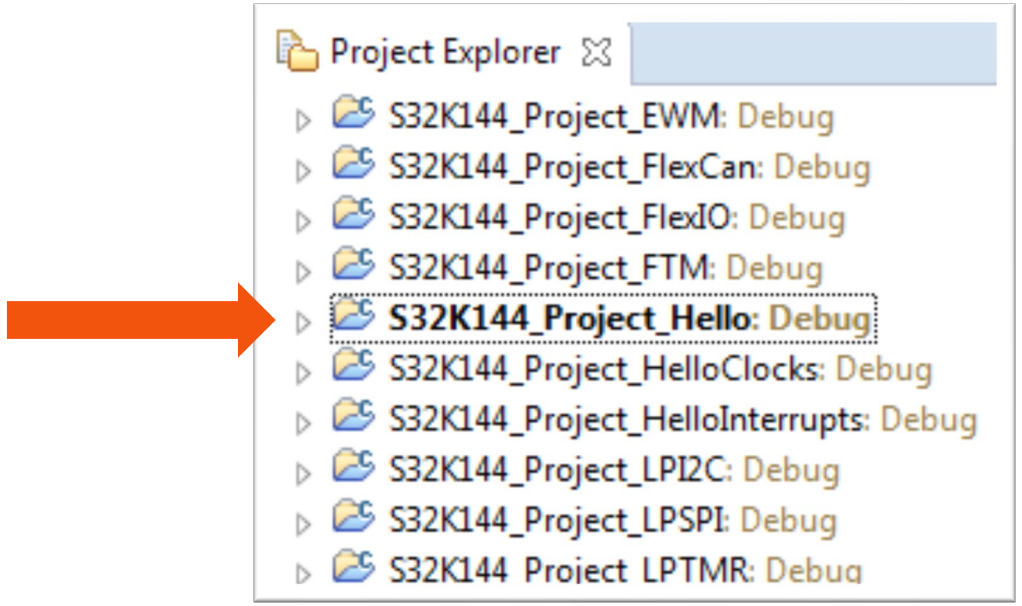
- Out of reset, clocks are not enabled to peripherals. (Done in PCC, Peripheral Clock Controller module)

- GPIO requires configuring
  - Input or output direction (Done in PTx, GPIO modules)
  - GPIO function (Done in PORTx, Port Control and Interrupts module)

- Hands on: Using next slides:
  - browse a project from a bare metal code example that exists in S32 Design Studio,
  - build, download to flash and run.

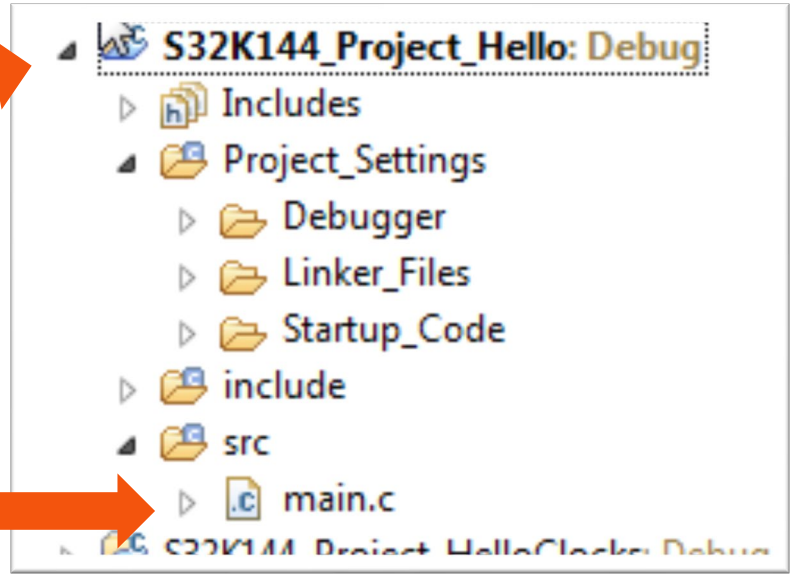
1. Hello World: Build 1 of 2

1. Select project “Hello”



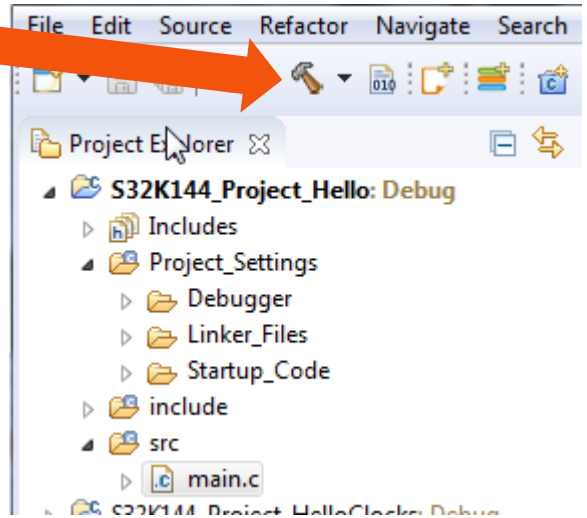
1. Hello World: Build 2 of 2

1. Expand Hello, then src folders

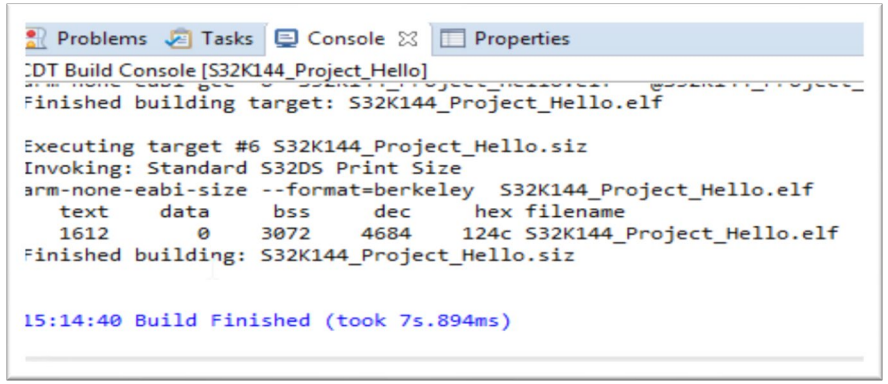


2. Double click on “main.c” to examine code

3. Click on hammer icon to build



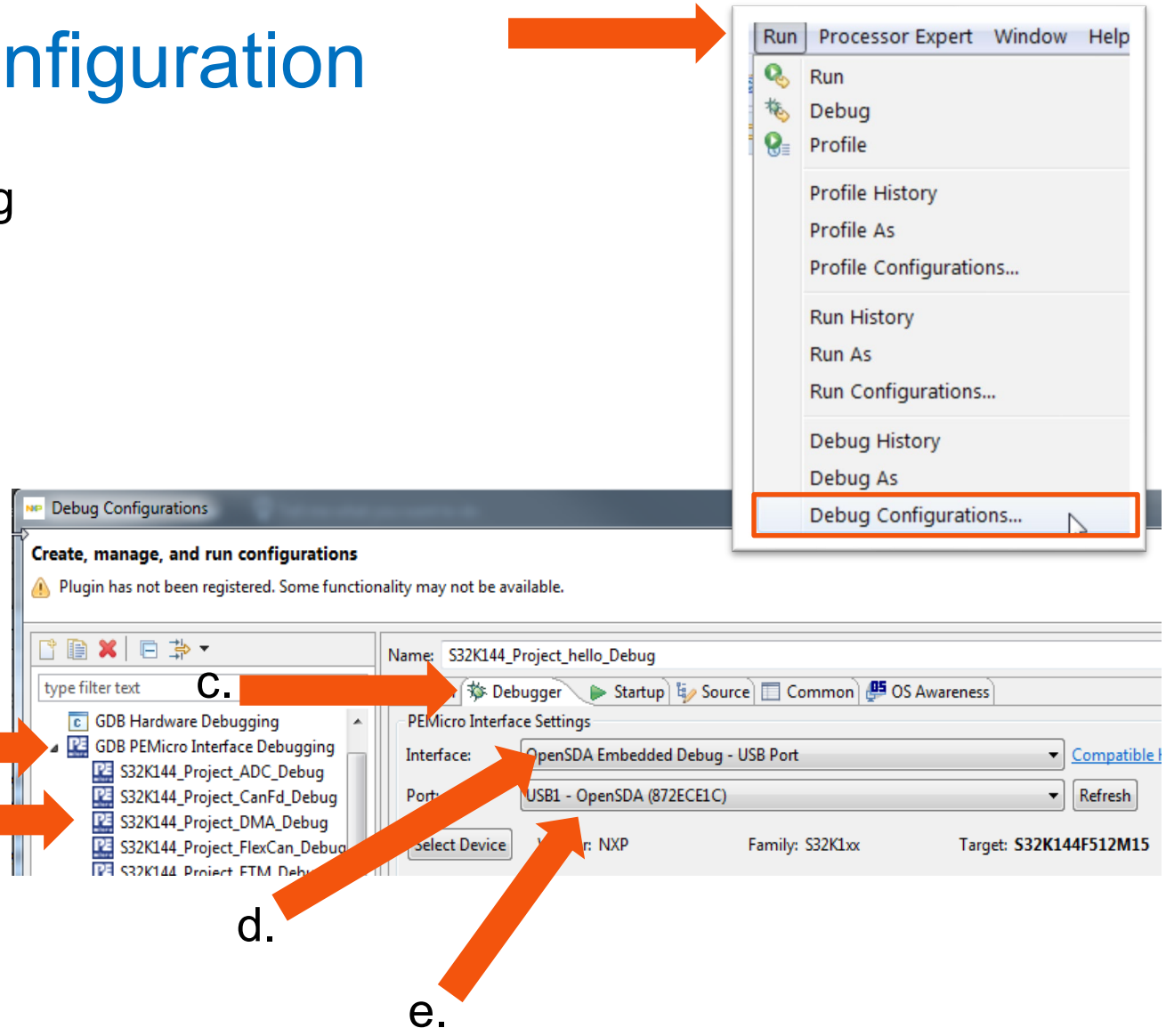
4. Click on Console tab to ensure the build finished



1. Hello World: Debug Configuration

1. The first time a project is built, a debug configuration is needed. Click Run→ Debug Configurations

2. Set up debug configuration:
   a. expand GDB PEMicro Interface,
   b. select hello_Debug,
   c. select Debugger tab,
   d. verify OpenSDA selected &
   e. Port is filled.



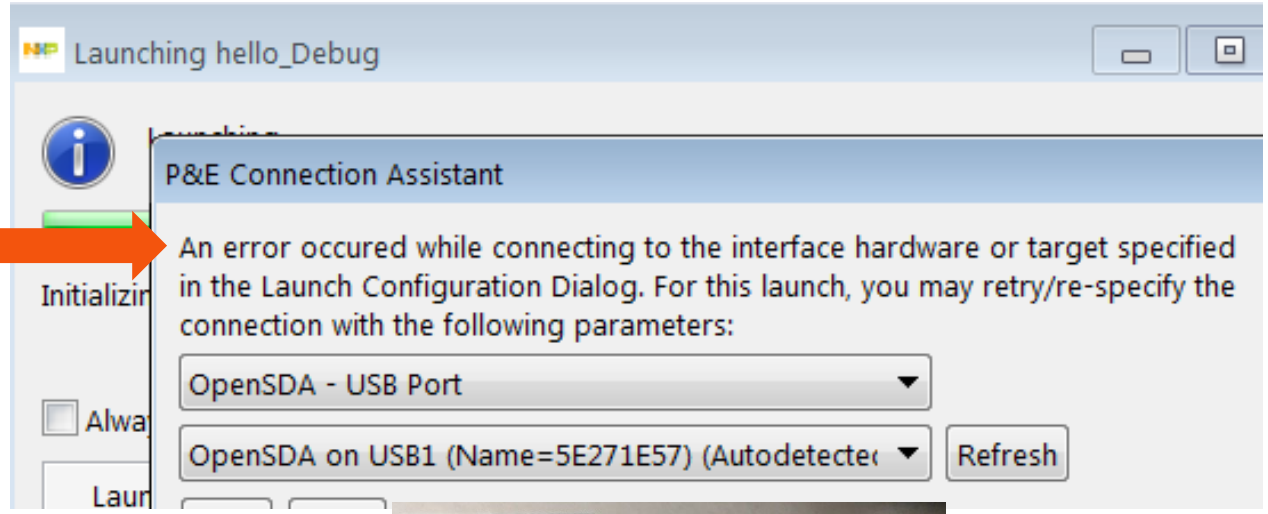
3. Click Debug on bottom right corner

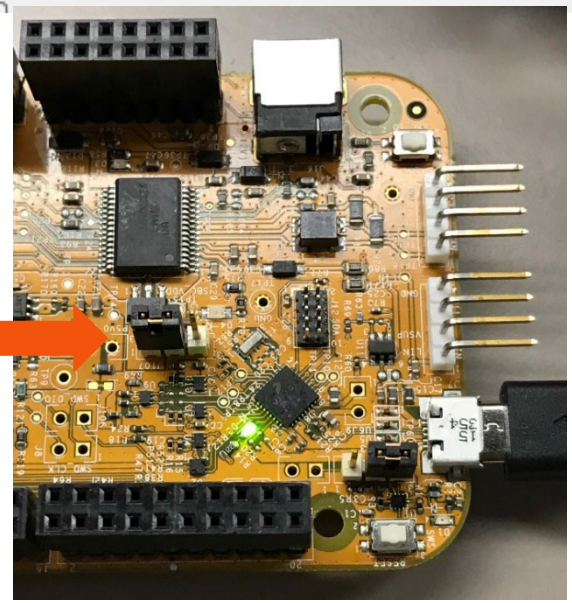
1. Hello World: Debug Tip

If you get this error message at Debug, you likely need to fix your power configuration.

In this case, J107 Power Source Selection jumper (center of board) was configured to provide power from an external 12V source, but none was connected.

Solution: connect J107 pins 2-3 (default) so power is routed from USB

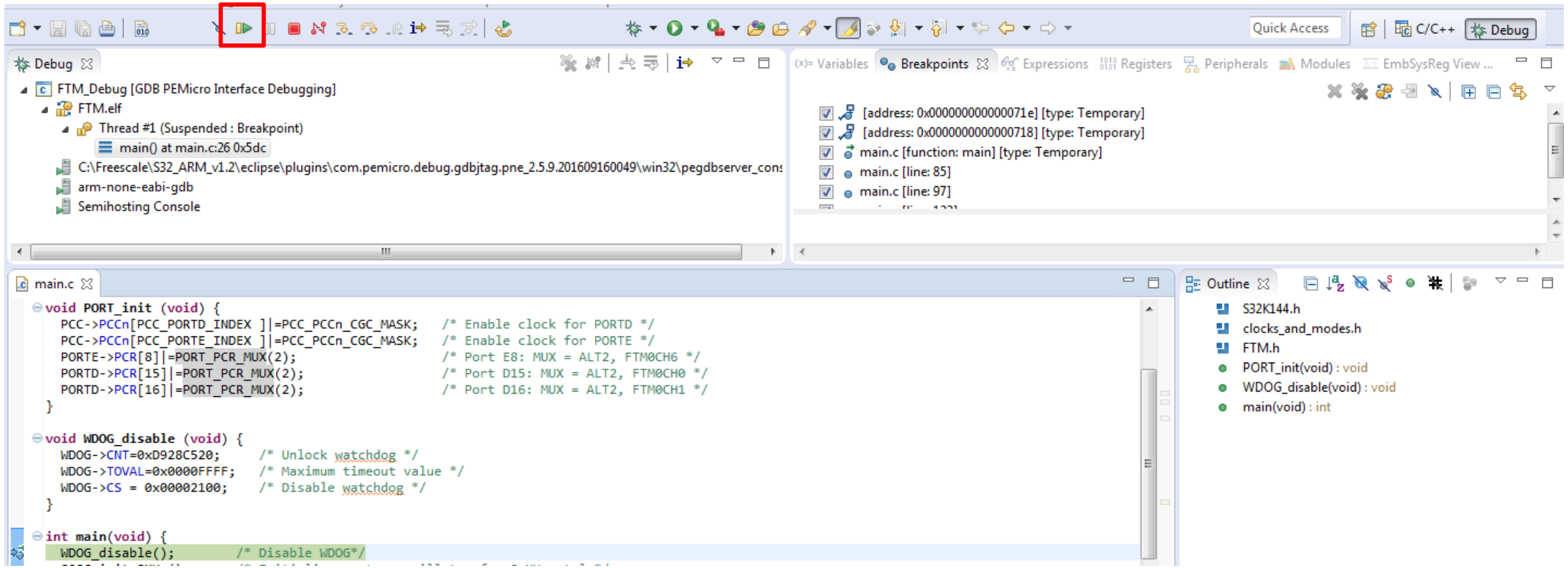




J107: move jumper to right if external 12V is not connected

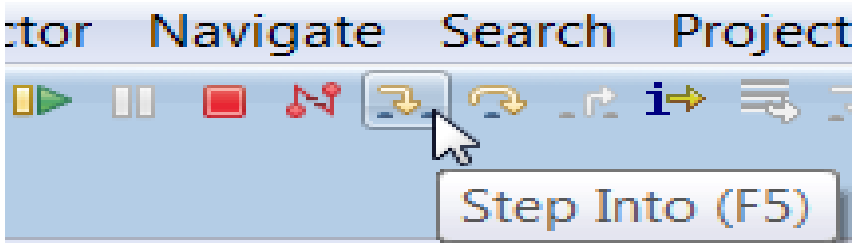
1. Hello World: Run a Project

- Debug perspective: click “Resume” icon to execute code

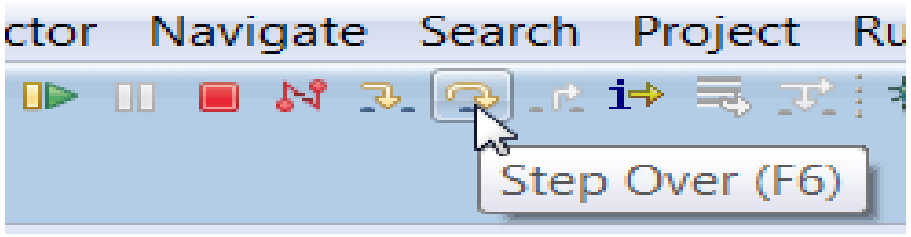


1. Hello World: Hands on: Debug Basics

- Step



- Step Over



- Step Return

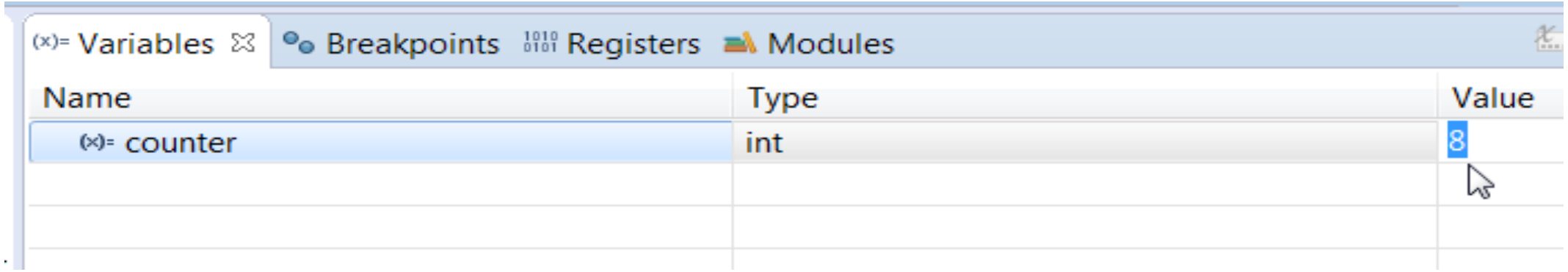
- Resume ["go", "run"]

- Suspend ["stop", "halt"]

- Terminate [exit debugger]

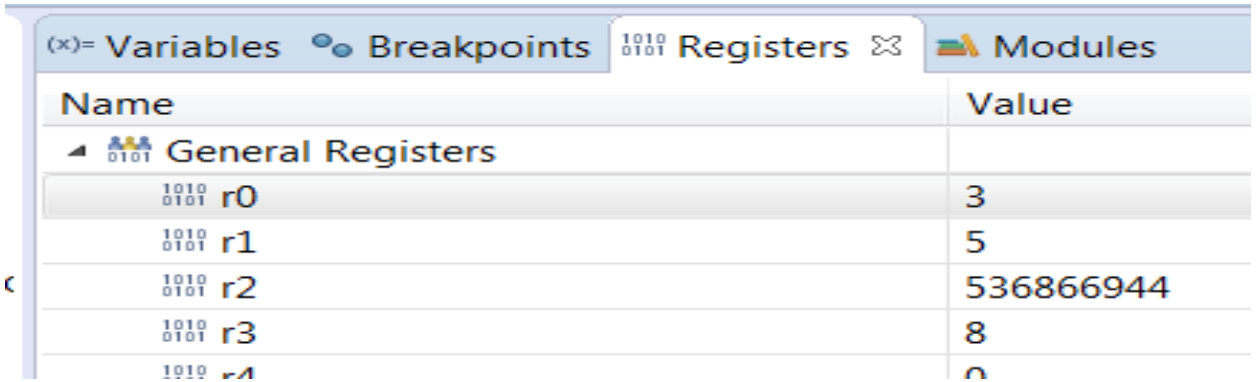
Debug Basics: View & Alter Variables

- View in “Variables” tab in upper right of debug perspective.
- Click on a value to allow typing in a different value.

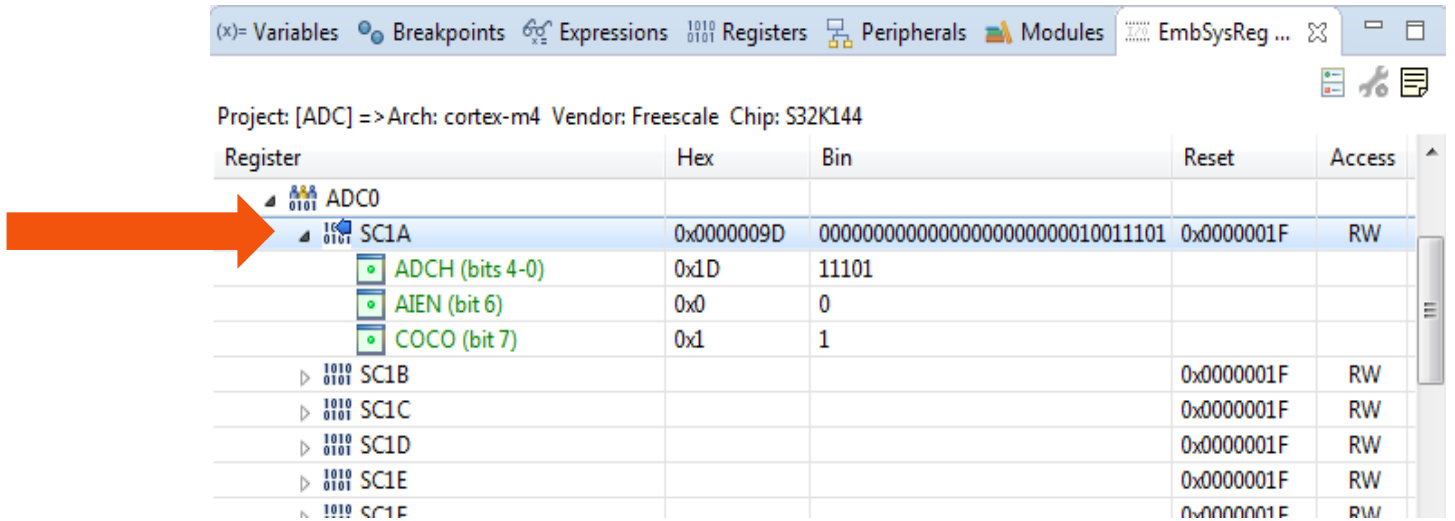


Debug Basics: View & Alter Registers

- View CPU registers in the “Registers” tab
- Click on a value to allow typing in a different value

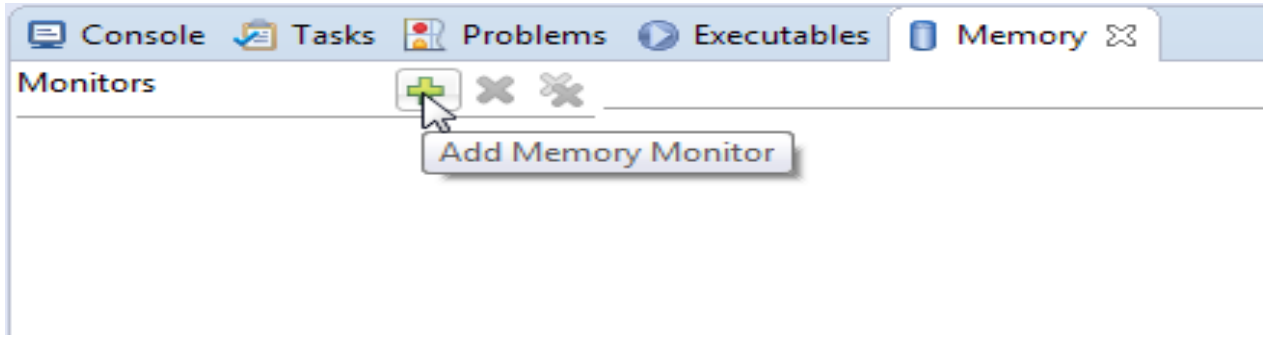


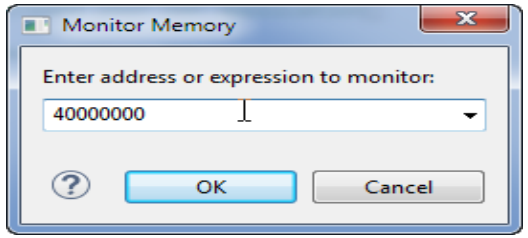
- View peripheral registers in the EmbSys Registers tab
- Double-click the targeted register to load contents

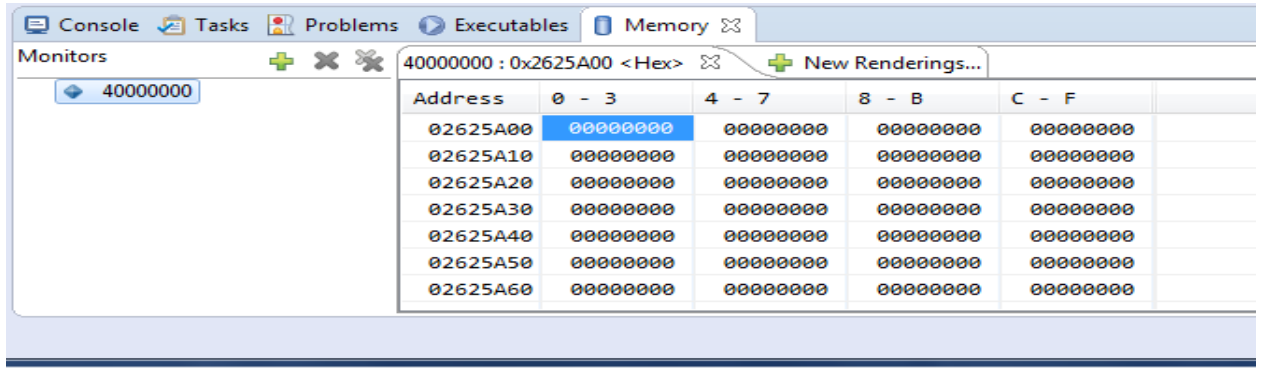


Debug Basics: View & Alter Memory

- Add Memory Monitor
- Select Base Address to Start at : 40000000
- View Memory



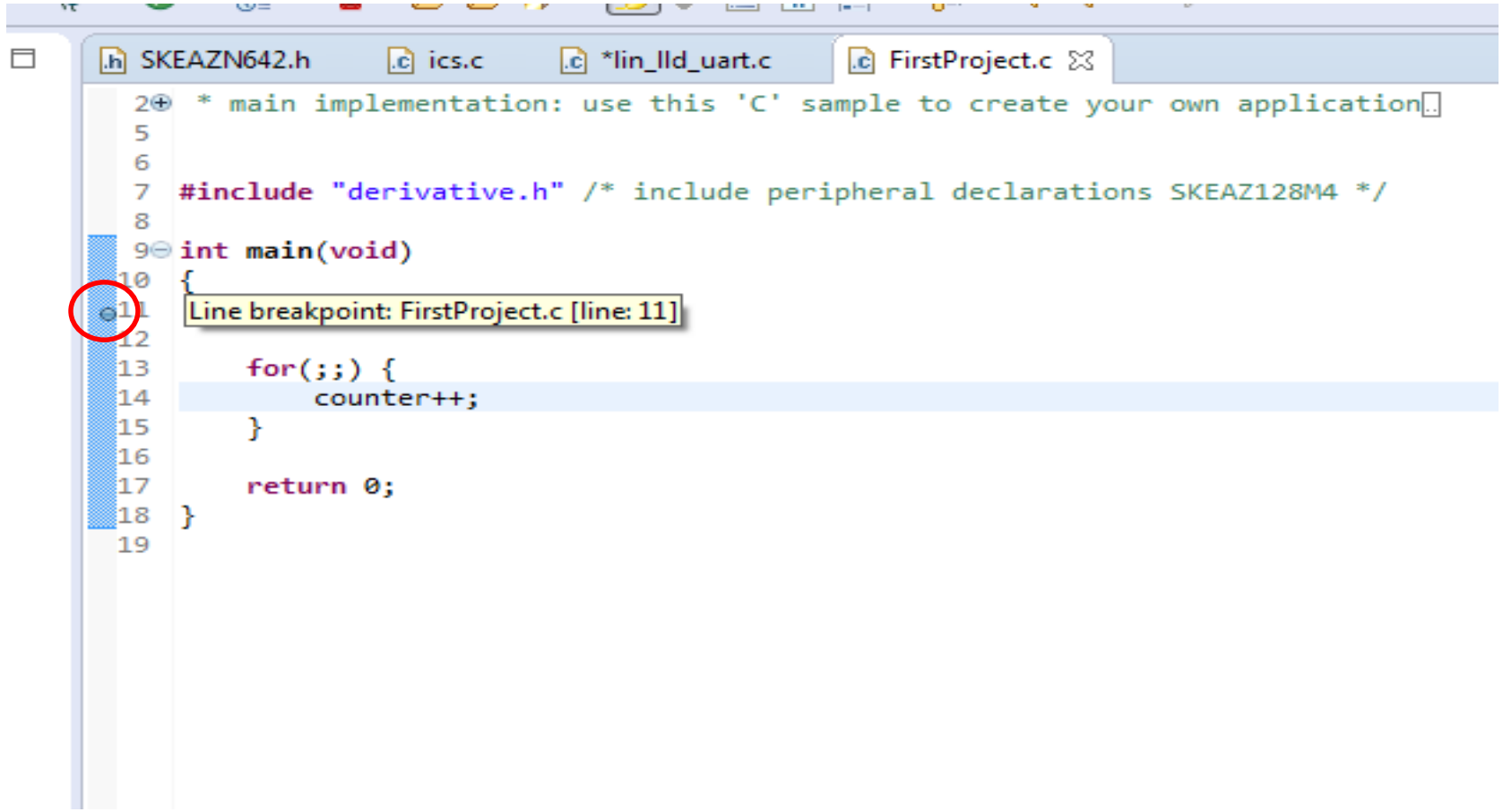




Debug Basics: Breakpoints

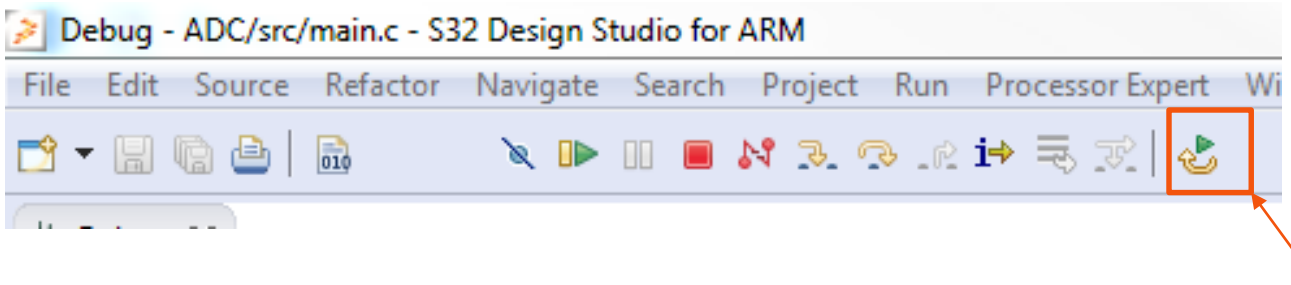
Add Breakpoint: Point and Click

- Light blue dot represents debugger breakpoint

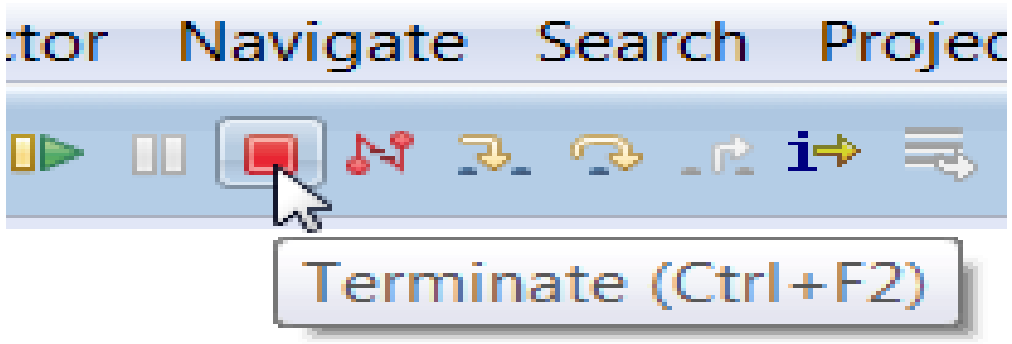


Debug Basics: Reset & Terminate Debug Session

- Reset program counter



- Terminate Ctl+F2



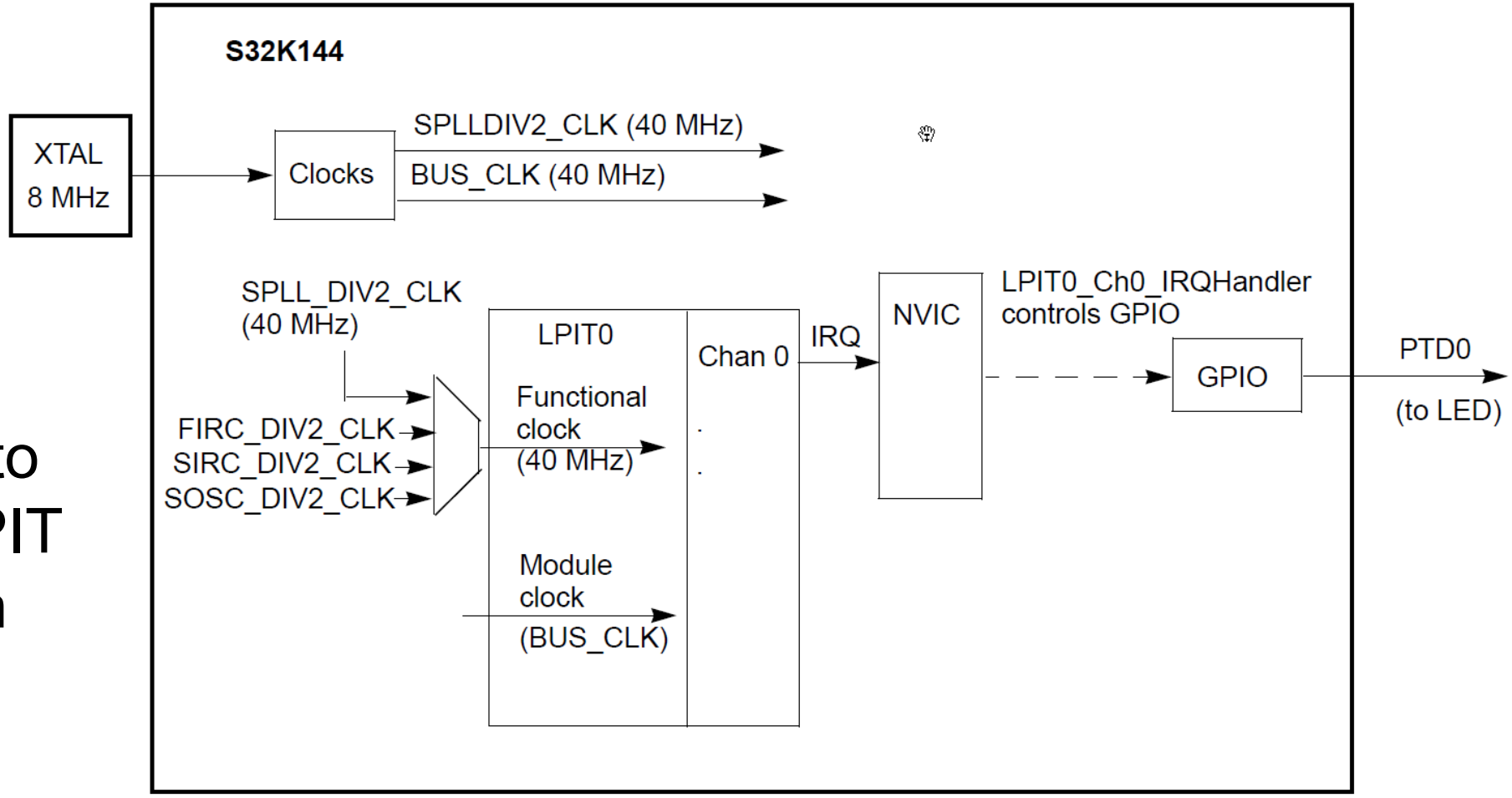
Always terminate a debug session when finished.

3. Hello World + Interrupts

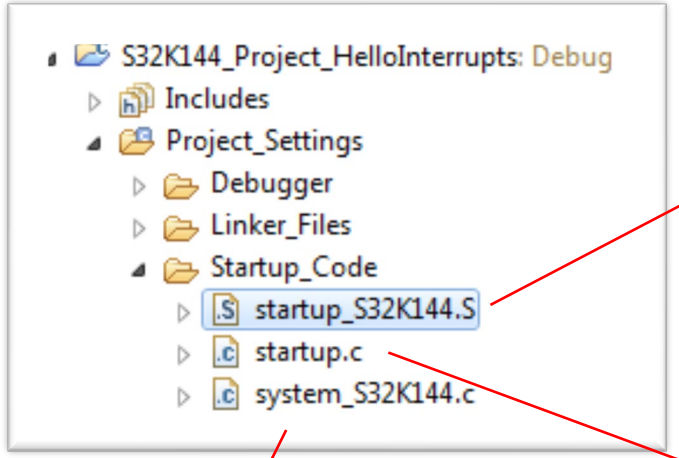
3. Hello World + Interrupts: Introduction

Summary:

An interrupt is implemented to service the LPIT counter match function.



3. Hello World + Interrupts: Startup code



- startup_S32K144.S
  - interrupt table
  - flash configuration field
  - reset handler
  - CPU initialization
  - jump to main()
- startup.c
  - RAM initialization
  - DATA table copying
- system_S32K144.c
  - Watchdog handling
  - Clock handling
  - Reset source handling

3. Hello World + Interrupts: Vector # vs Interrupt

Address	Contents in Big Endian memory view	Contents as Little Endian format addresses[1]	Vector #	IRQ # (NVIC interrupt source)	Symbol in file startup_S32K144.S, section .isr_vector, __isr_vector	Description
0x0000 0000	0070 0020	2000 7000	0		__StackTop	Initial stack pointer
0x0000 0004	1104 0000	0000 4010	1		Reset_Handler	Initial Program Counter
0x0000 0008	4D04 0000	0000 044C	2		NMI_Handler	Non-maskable IRQ (NMI) Vector
0x0000 003C	4D04 0000	0000 044C	15		SysTick_Handler	Sys tick timer (Sys Tick) Vector
0x0000 0040	4D04 0000	0000 044C	16	0	DMA0_IRQHandler	Interrupt # 0 Vector: DMA Channel 0 transfer complete
0x0000 0044	4D04 0000	0000 044C	17	1	DMA1_IRQHandler	Interrupt # 1 Vector DMA Channel
			...	...		
0X0000 0100	5906 0000	0000 0658	64	48	LPIT0_Ch0_IRQHandler	Interrupt #48 Vector: LPIT0 Ch. 0

Quick look at S32K Reference Manual (Interrupt tables, memory map, etc.)

3. Hello World + Interrupts: Interrupt Handlers

- Location of vector table in S32 Design Studio project: **startup_S32K144.s**
- Vector table is here

```
    .section .isr_vector, "a"
    .align 2
    .globl __isr_vector
__isr_vector:
    .long   __StackTop                              /* Top of Stack */
    .long   Reset_Handler                           /* Reset Handler */
    .long   NMI_Handler                             /* NMI Handler*/
```

- LPIT0 channel 0 interrupt handler definition

```
    .long   RTC_Seconds_IRQHandler                  /* RTC overflow*/
    .long   LPIT0_Ch0_IRQHandler                    /* LPIT CH0 overflow*/
```

3. Hello World + Interrupts: “Weak” Handlers

- Startup_S32K144.S

```
    .weak DefaultISR
    .type DefaultISR, %function
DefaultISR:
    b       DefaultISR
    .size DefaultISR, . - DefaultISR

/*    Macro to define default handlers. Default handler
 *    will be weak symbol and just dead loops. They can be
 *    overwritten by other handlers */
    .macro def_irq_handler  handler_name
    .weak \handler_name
    .set  \handler_name, DefaultISR
    .endm
```

```
def_irq_handler    RTC_Seconds_IRQHandler
def_irq_handler    LPIT0_Ch0_IRQHandler
def_irq_handler    LPIT0_Ch1_IRQHandler
def_irq_handler    LPIT0_Ch2_IRQHandler
def_irq_handler    LPIT0_Ch3_IRQHandler
def_irq_handler    PDB0_IRQHandler
```

- Main.c

```
void LPIT0_Ch0_IRQHandler (void)
{
    lpit0_ch0_flag_counter++;         /* Increment LPIT0 timeout counter */
    PTD->PTOR |= 1<<0;                /* Toggle output on port D0 (blue LED) */
    LPIT0->MSR |= LPIT_MSR_TIF0_MASK; /* Clear LPIT0 timer flag 0 */
}
```

- Overrides the “weak” default handler

3. Hello World + Interrupts: Interrupt initialization

The Nested Vector Interrupt Controller (NVIC) is used to initialize an interrupt:

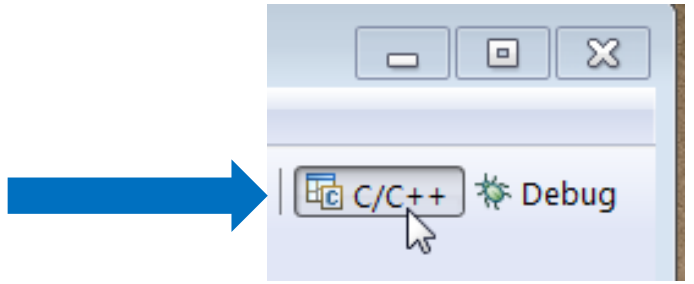
- Clear any prior pending interrupt (in case there was one)
  - — Write a 1 to the interrupt # bit in Interrupt Clear Pending Register (ICPR)
- Enable the desired interrupt
  - — Write a 1 to the interrupt # bit in the Interrupt Set Enable Register (ISER)
- Set the interrupt's priority
  - — Write a priority from 0 to 15 to the appropriate Interrupt Priority register (IP)

Example for IRQ # 48:

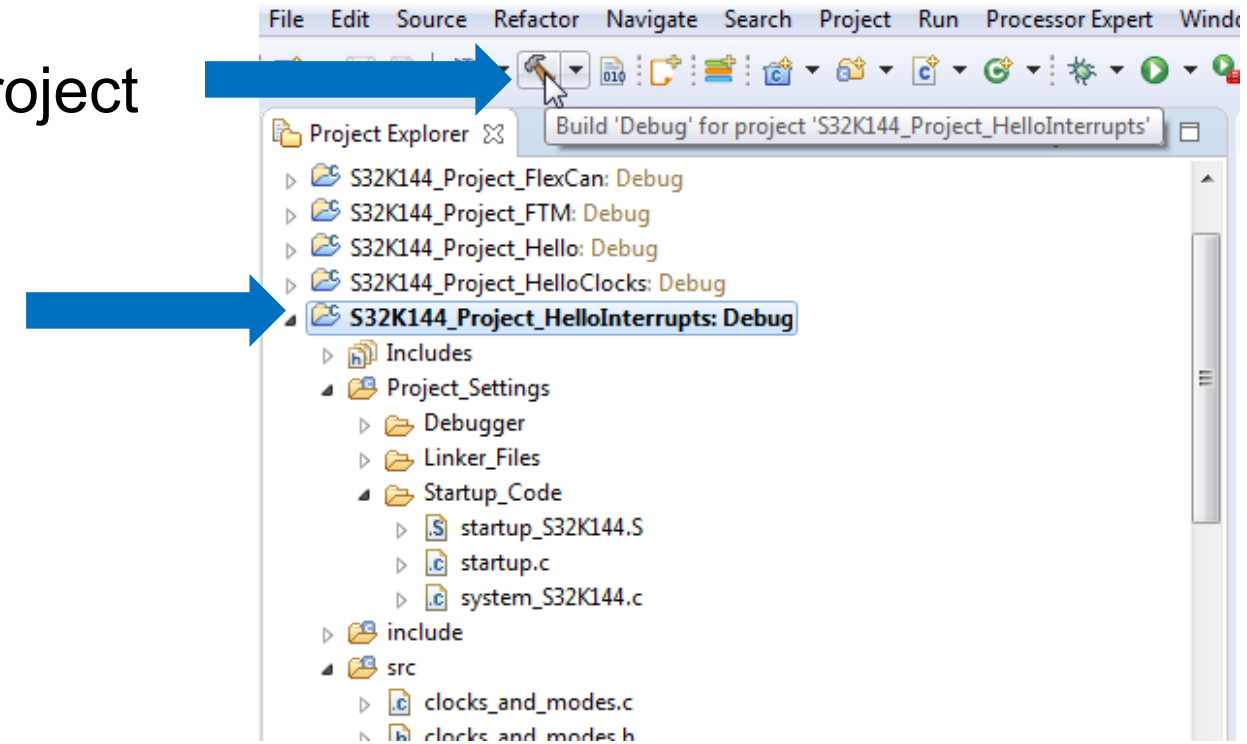
```
S32_NVIC->ICPR[1] = 1 << (48 % 32);   /* IRQ48-LPIT0 ch0: clear any pending IRQ*/

S32_NVIC->ISER[1] = 1 << (48 % 32);   /* IRQ48-LPIT0 ch0: enable IRQ */

S32_NVIC->IP[48] = 0xA0;              /* IRQ48-LPIT0 ch0: priority 10 of 0-15*/
```

3. Hello World + Interrupts

1. Be sure C/C++ perspective is selected, otherwise click on it to get it selected.



2. Double click on HelloInterrupts project
3. Click on Build icon

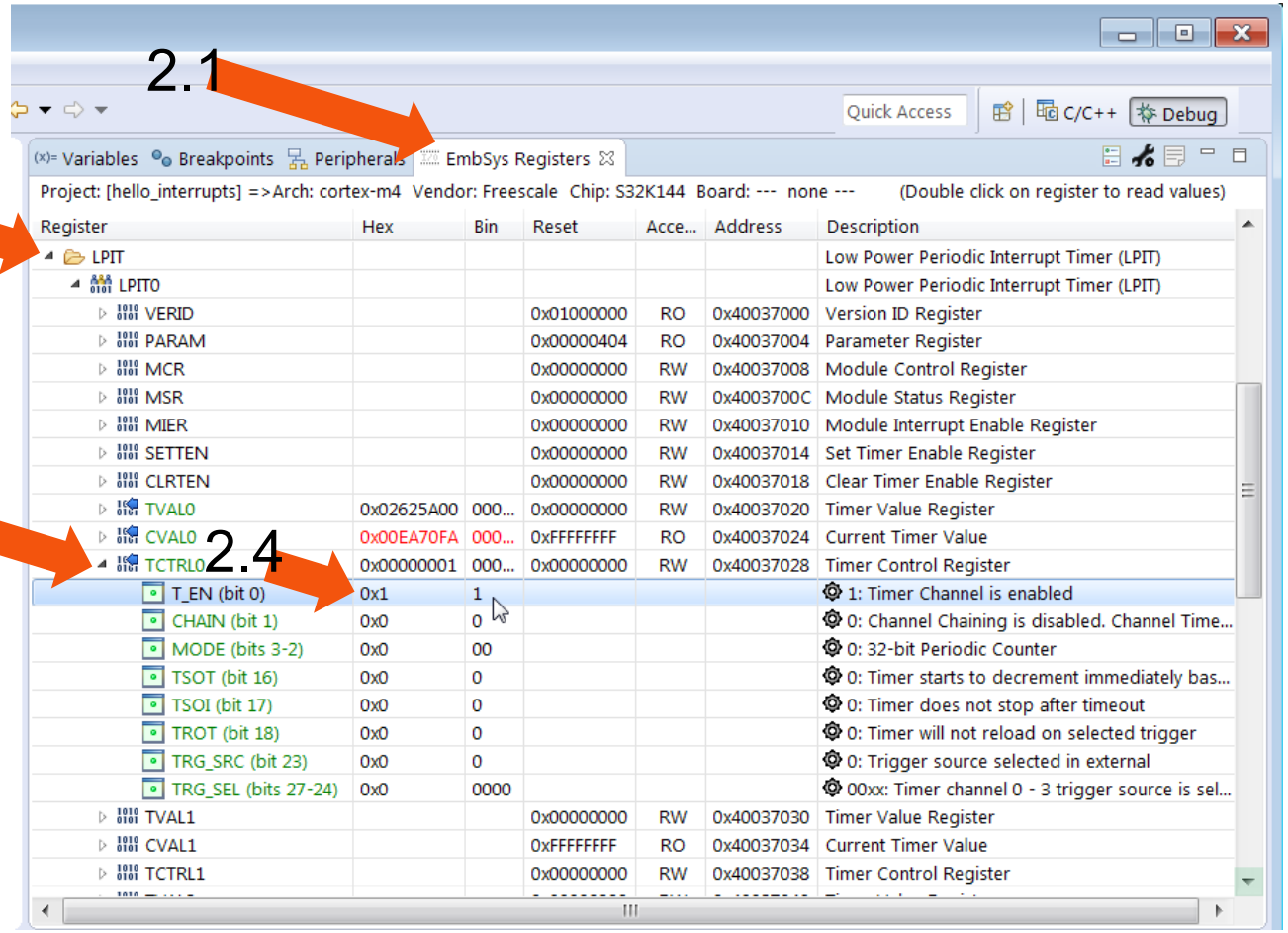


4. Click on Debug icon
   - Create debug connection… Go…

3. Hello World + Interrupts: View, Modify Registers

1. Use Resume to run, Suspend to halt

2. Disable LPIT channel 0:
   - Click EmbSys Registers tab
   - Scroll down list about half way and expand LPIT & LPIT0
   - Expand register TCTRL0
   - Click on hex value (0x1) to change to 0x0, disabling counter
3. Use Resume to run again. Note LED stopped blinking
4. Click Terminate.



How to Break an ARM (1)

- Uninitialized clock sources will result in a CPU Hard Fault
- ARM Cortex-M generally allows each peripheral to enable/disable the clock to its register file
  - Power-saving feature!

- Exercise: Hard Fault handler
  - Remove the line of code that enables port clock
  - Observe results
  - Find the offending line of code

HardFault Handler

- Change this PORT_Init() code:

PCC-> PCCn[PCC_PORTD_INDEX] = PCC_PCCn_CGC_MASK;

- To:

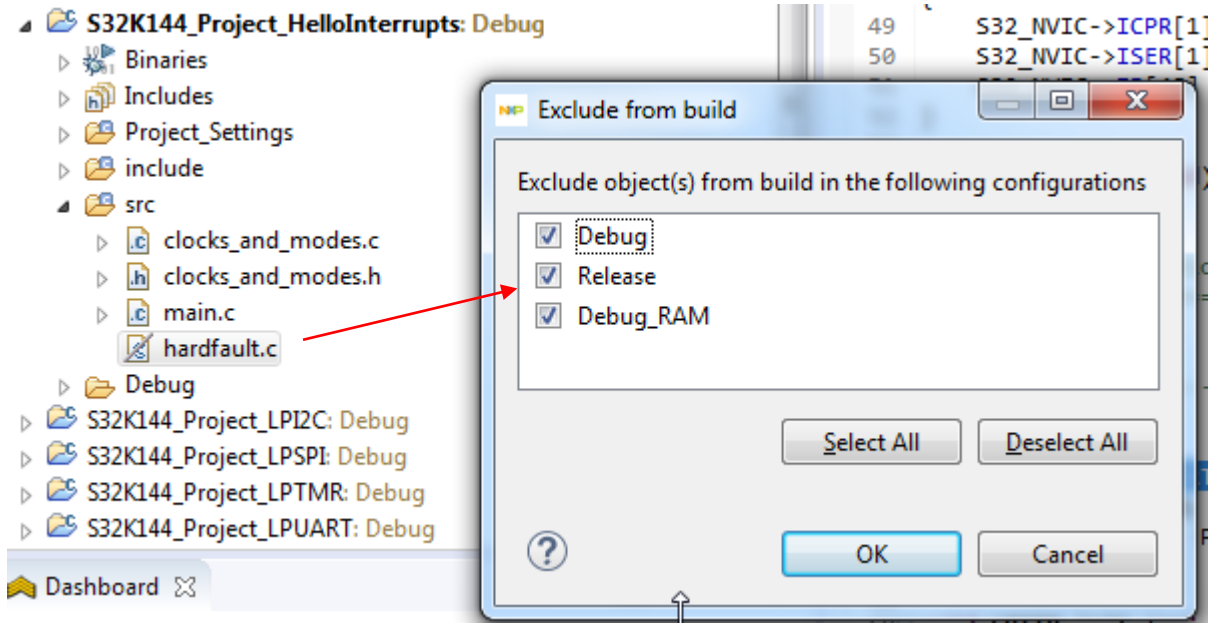
PCC-> PCCn[PCC_PORTE_INDEX] = PCC_PCCn_CGC_MASK;

- Add “hardfault.c” to the project

- Build & Debug

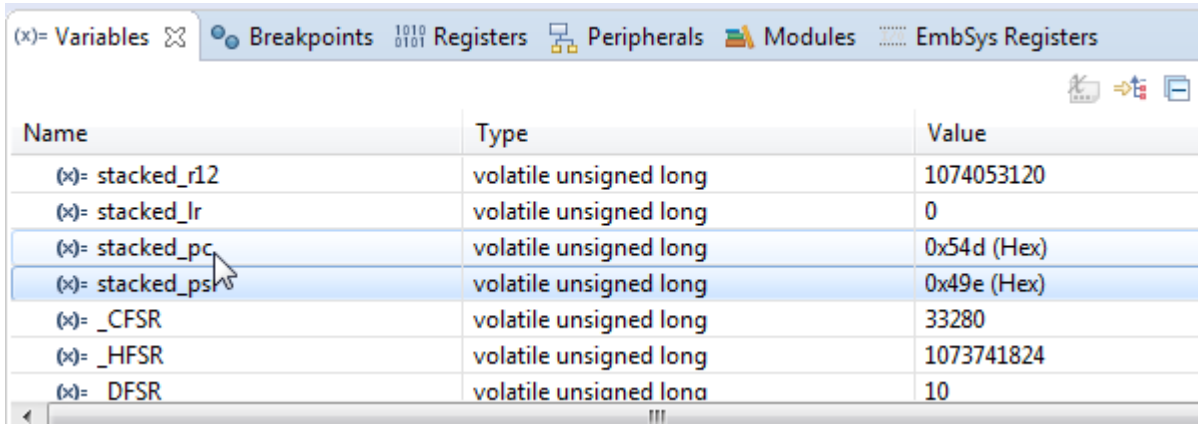
Follow instructor’s lead…

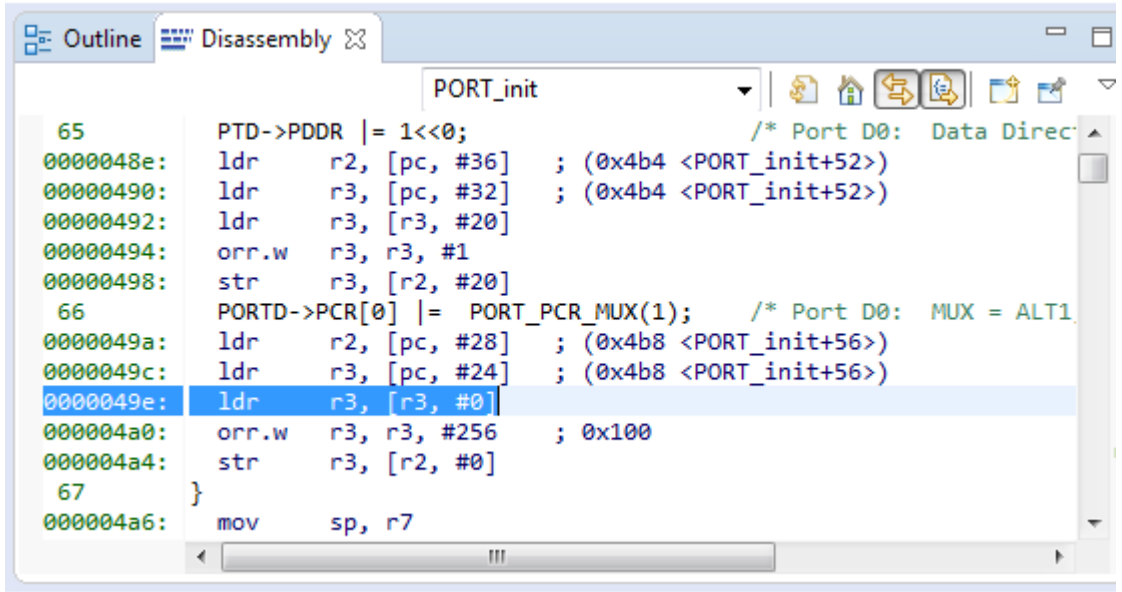
- Right-click “hardfault.c”
- Choose Resource Configurations → Exclude from Build
- Deselect All → OK



Find the Offending Line of Code…

- Run the program
- View Variables tab to find “stacked_psr”
- Scroll Disassembly to the indicated address
- Correlate to C source





How to Break an ARM (2)

- Turn on compiler optimization (-O2); then Clean; Build; Debug
  - Observe results – what happened to the LED flash?
- Change the code, to avoid optimization “gotchas”
  - Before

```
void LPIT0_Ch0_IRQHandler (void)
{
  LPIT0->MSR |= LPIT_MSR_TIF0_MASK; /* Clear LPIT0 timer flag 0 */
             /* Perform read-after-write to ensure flag clears before ISR exit */
  lpit0_ch0_flag_counter++;         /* Increment LPIT0 timeout counter */
  PTD->PTOR |= 1<<0;                /* Toggle output on port D0 (blue LED) */
}
```

  - After

Move this line of code

```
void LPIT0_Ch0_IRQHandler (void)
{
             /* Perform read-after-write to ensure flag clears before ISR exit */
  lpit0_ch0_flag_counter++;         /* Increment LPIT0 timeout counter */
  PTD->PTOR |= 1<<0;                /* Toggle output on port D0 (blue LED) */
 LPIT0->MSR |= LPIT_MSR_TIF0_MASK; /* Clear LPIT0 timer flag 0 */
}
```

Guidelines for ISR's

- Clear the peripheral interrupt flag first
- Perform memory R/M/W afterward
- Use __attribute__((optimize("O0"))) to override gcc compiler flags

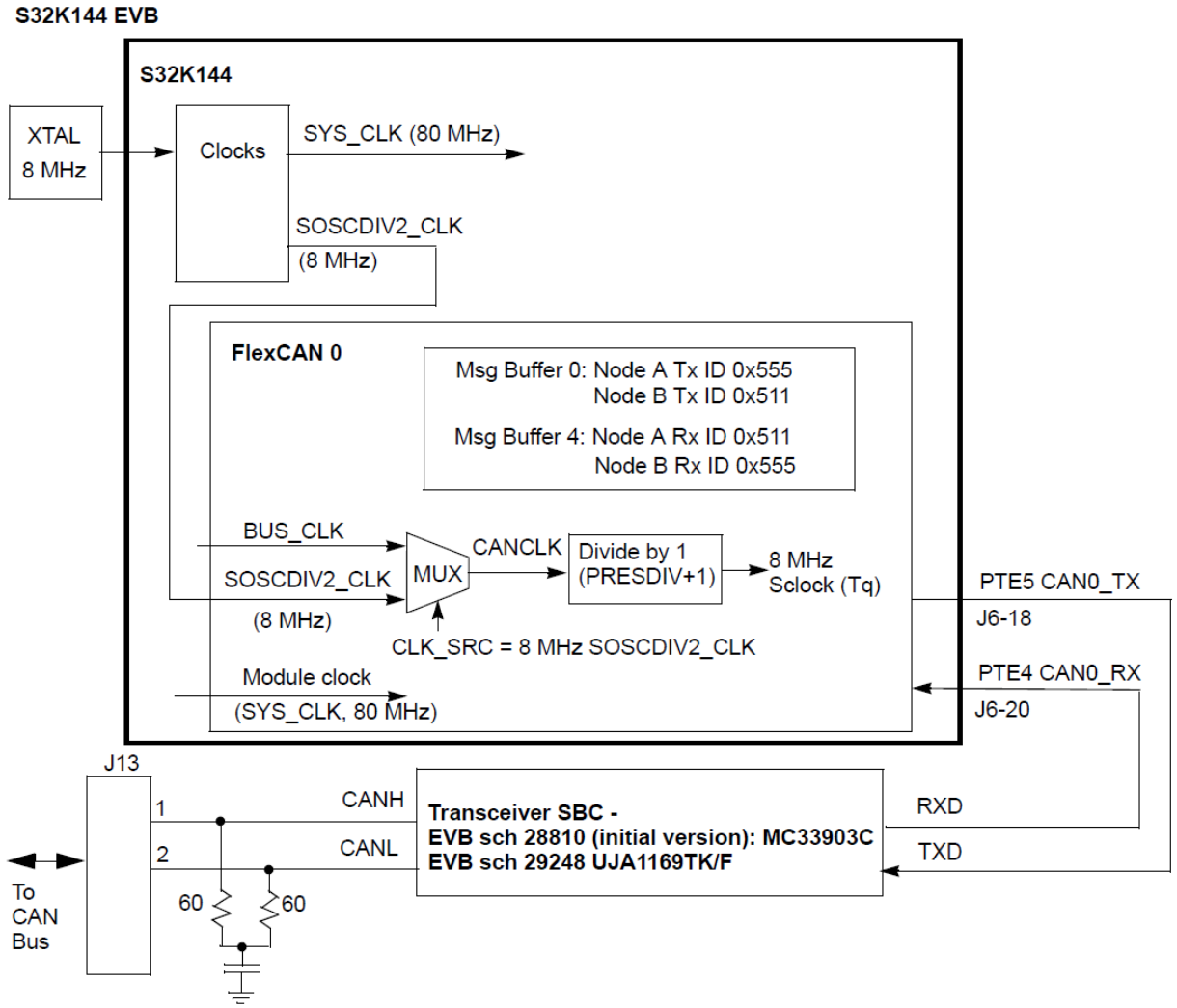
```
void __attribute__((optimize("O0"))) LPIT0_Ch0_IRQHandler (void)
{
  LPIT0->MSR |= LPIT_MSR_TIF0_MASK; /* Clear LPIT0 timer flag 0 */
  PTD->PTOR |= 1<<0;                /* Toggle output on port D0 (blue LED) */
  lpit0_ch0_flag_counter++;         /* Increment LPIT0 timeout counter */
}
```

9. Classic CAN

9. Classic CAN: Introduction

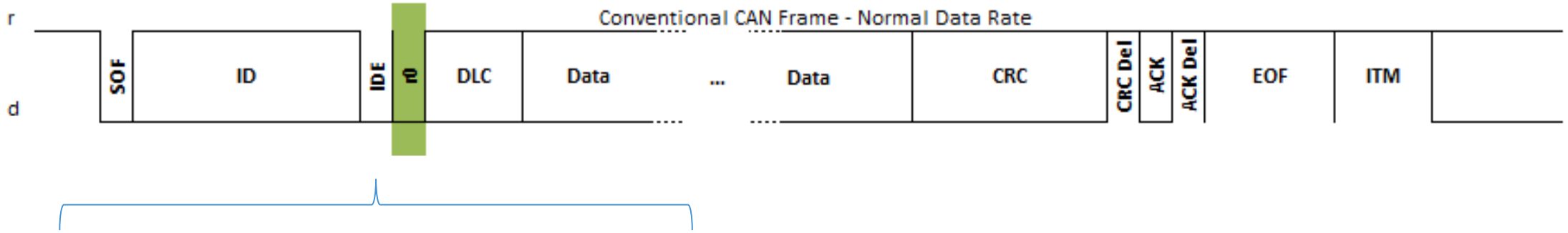
Summary:

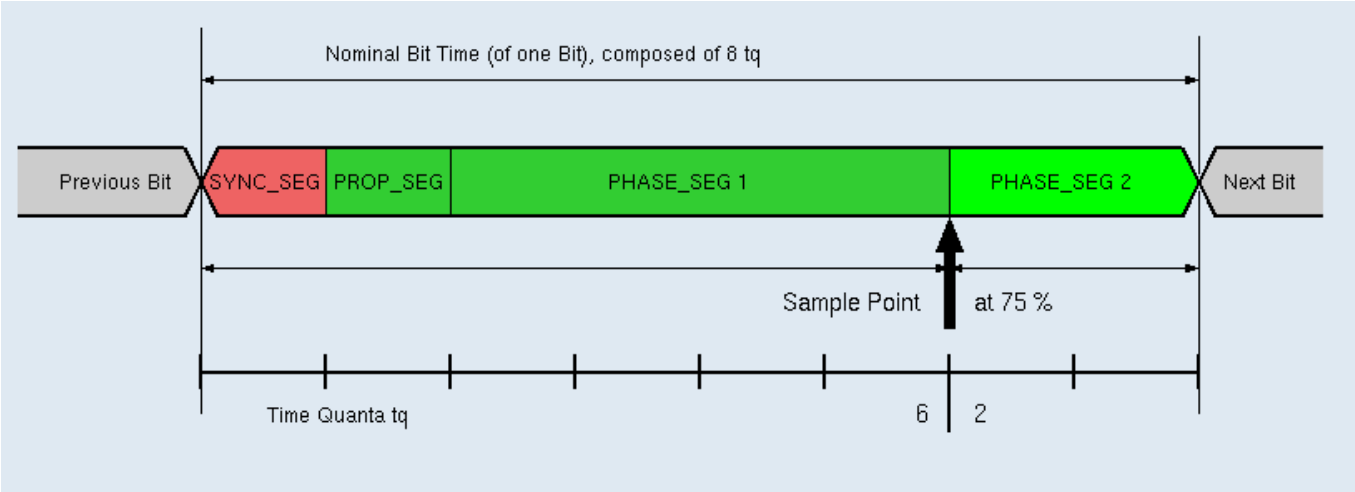
- FlexCAN module is initialized for 500K bps based on an 8 MHz crystal
- Setup a bus analyzer to prototype your basic message scheme
- Connect S32K EVB to VCAN4 analyzer



9. Classic CAN Frame Overview

- CAN is still the backbone of vehicle communications, today



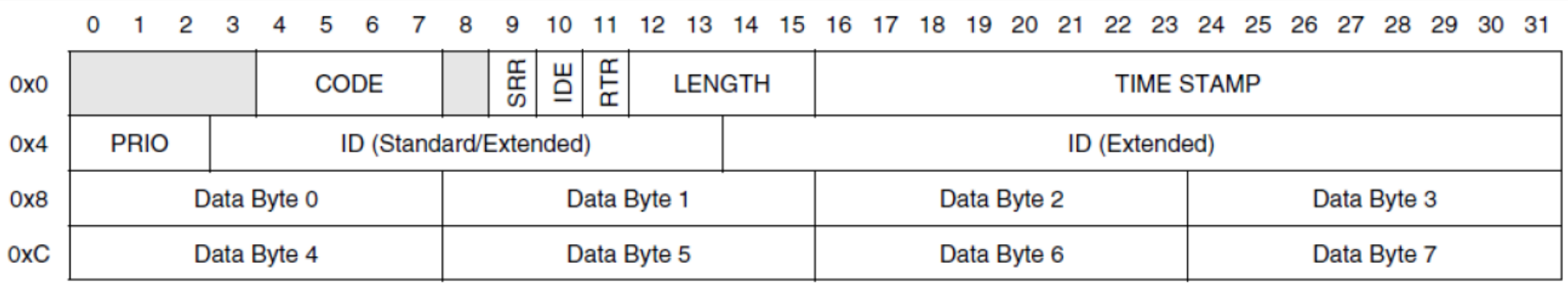


9. Classic CAN: Key Points

- Key design parts described:
  - UJA1169 System Basis Chip does not require initialization for CAN or LIN
  - CAN 2.0 timing calculations for 500 MHz bps

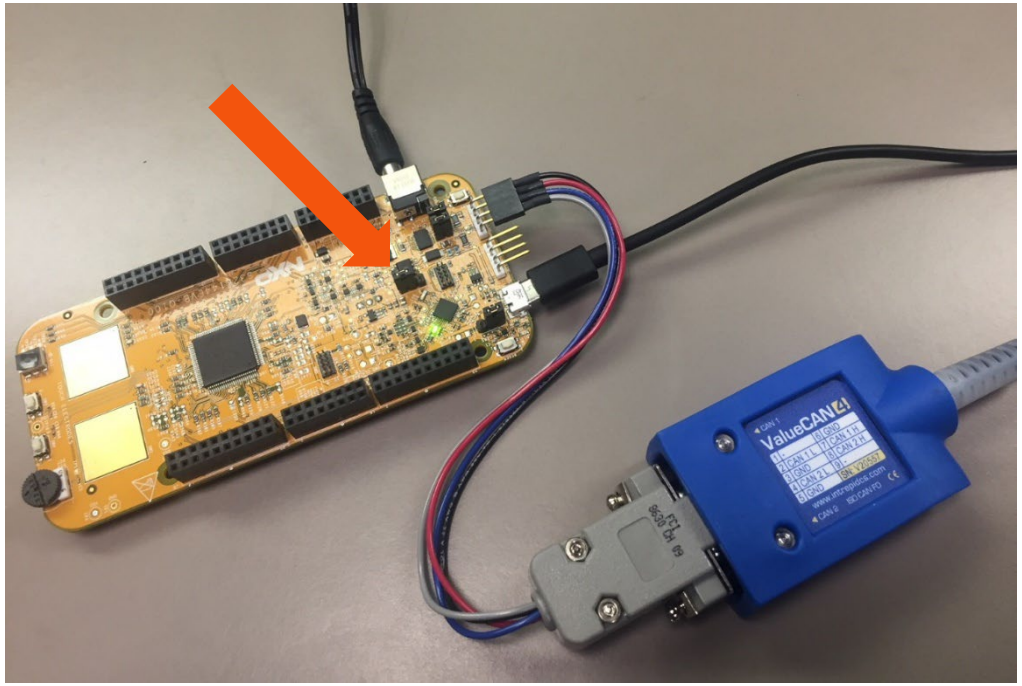
	SYNCSEG Time Quanta	PROP_SEG Time Quanta	PHASE_SEG1 Time Quanta	PHASE_SEG2 Time Quanta	Number of Time Quanta per bit time
CAN 2.0 Time Quanta	1	7	4	4	16
CAN_CTRL1 register bit fields	–	PROPSEG = 6	PSEG1 = 3	PSEG 2 = 3	-

  - Message buffer structure



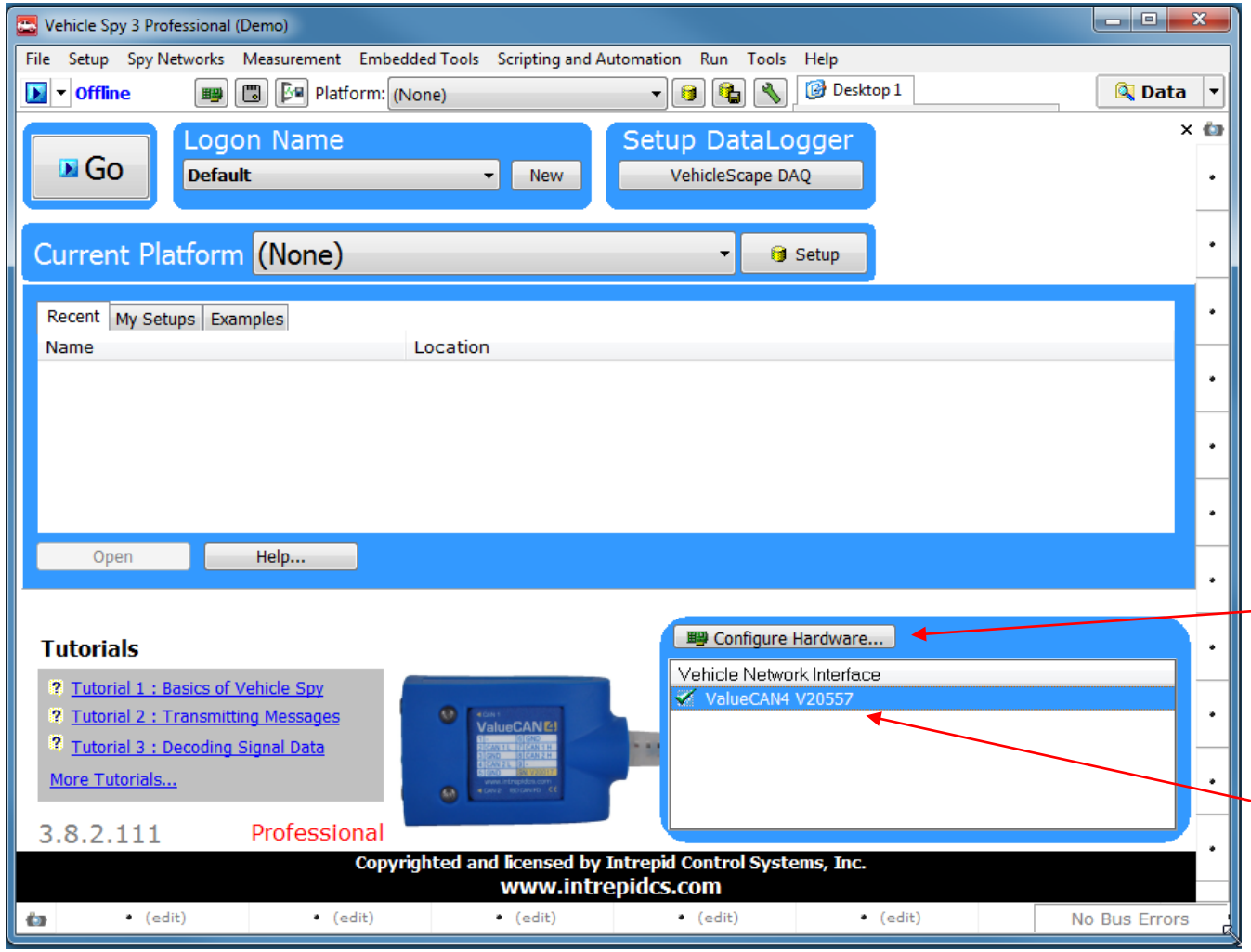
9. Classic CAN: Connect and Run…

- Make sure the power supply jumper is set for 12V external supply: J107: [2-3]
- Connect CAN, power cables as shown
- Connect USB cable for download & debug.
- Select the “S32K144_Project_FlexCan”
  - Code walk-through…
    - Change the conditional check that blinks the LED
- Build and Debug



ValueCAN4 & Vehicle Spy

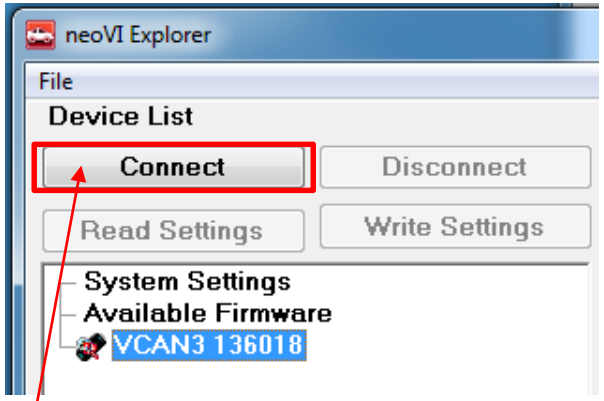
VSPY3



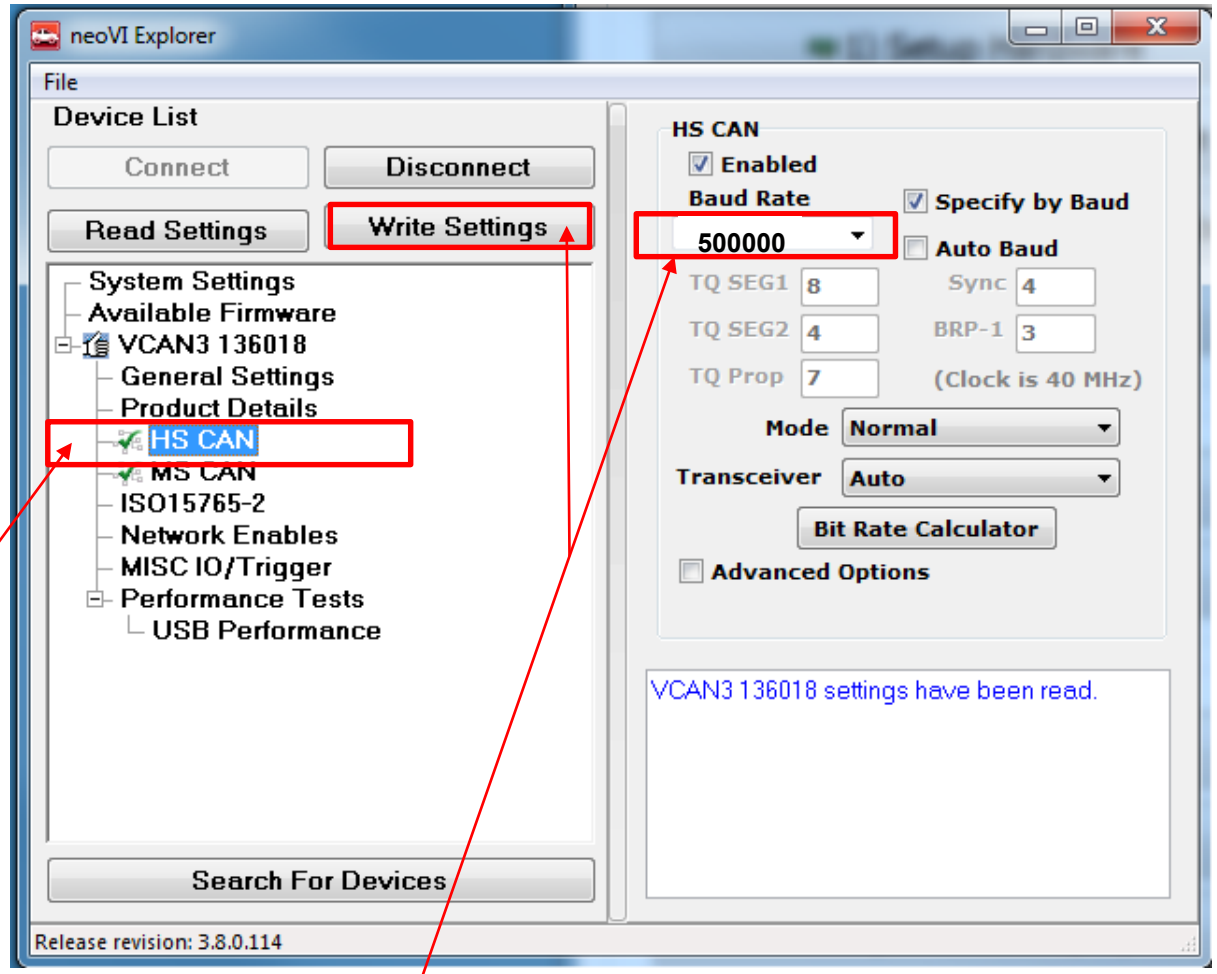
Configure Hardware (click here)

Detected Hardware

Hardware Configuration



1) Connect to HW

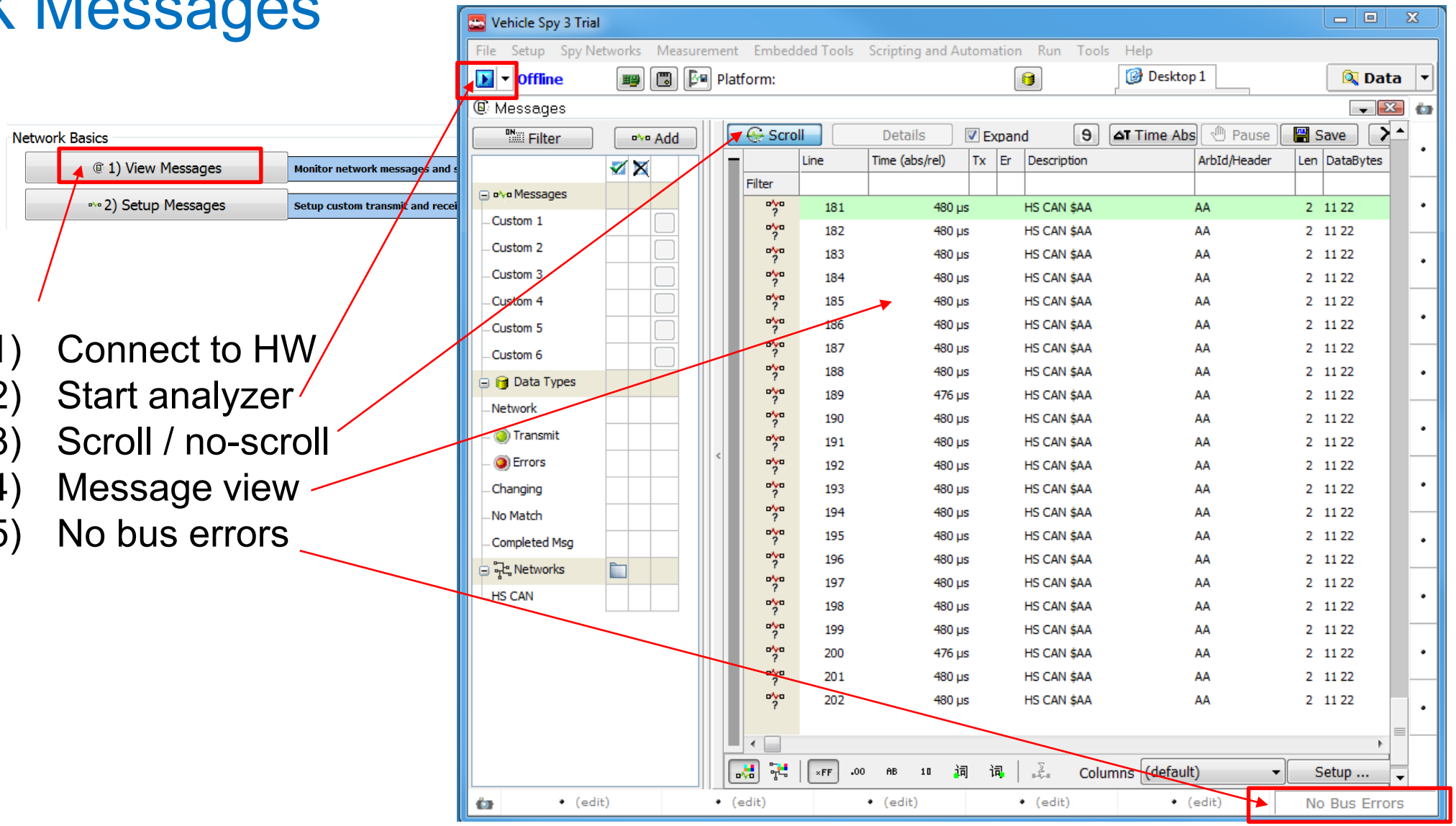


2) Select HS CAN

3) Choose baud rate 500,000 then write-in.

4) Close the window

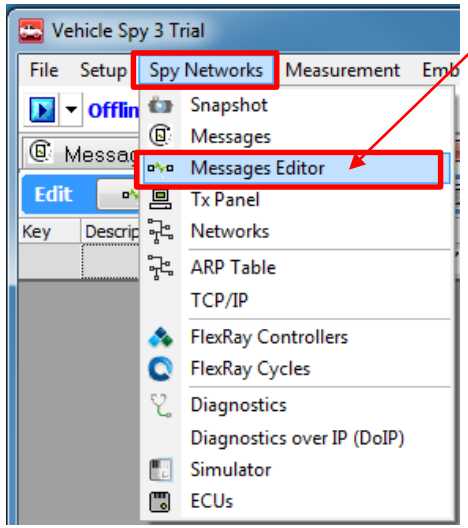
RX Messages



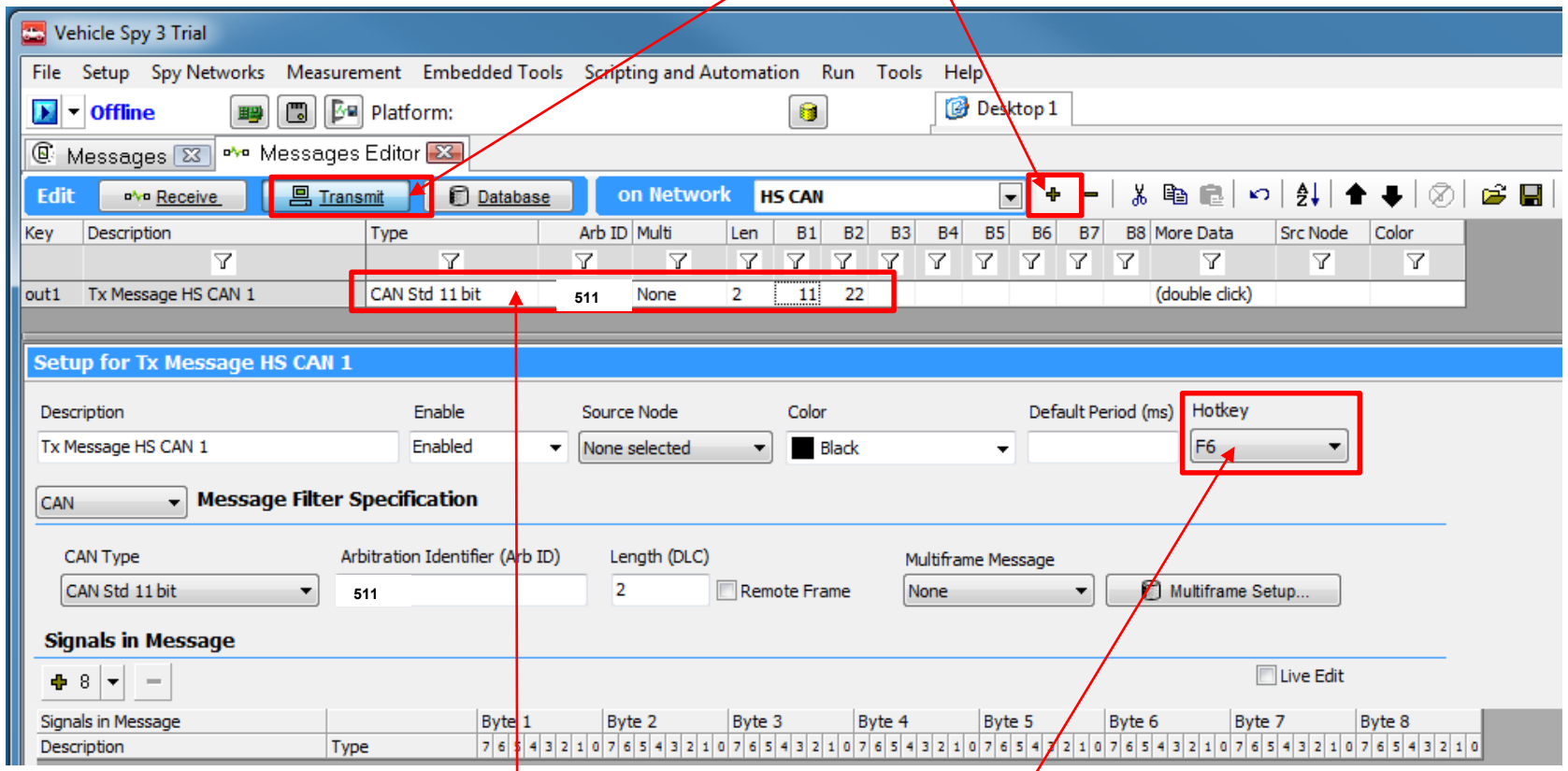
1) Connect to HW
2) Start analyzer
3) Scroll / no-scroll
4) Message view
5) No bus errors

Define TX Messages

2) Add a Transmit message with ID 0x511



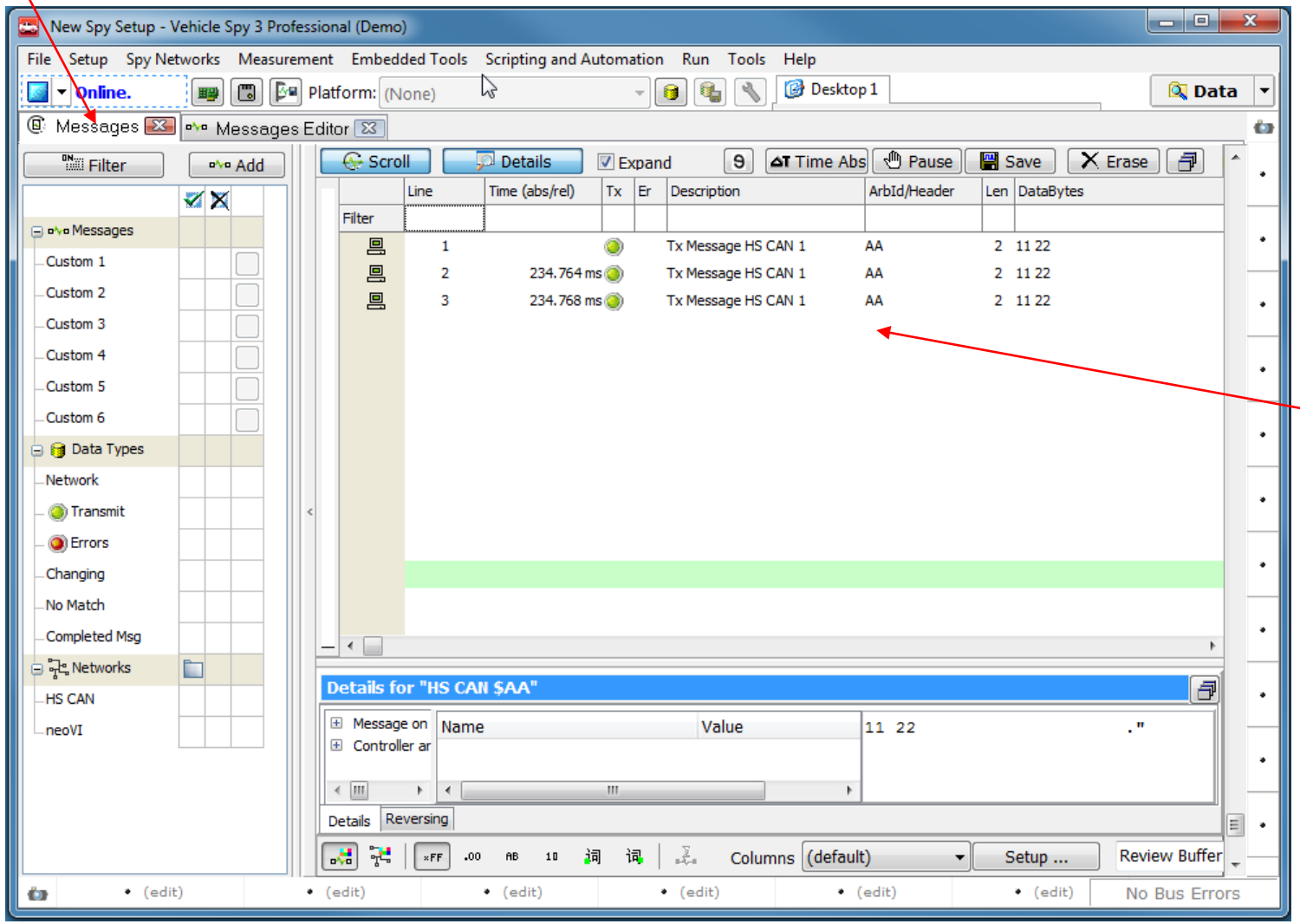
1) Launch Message Editor



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

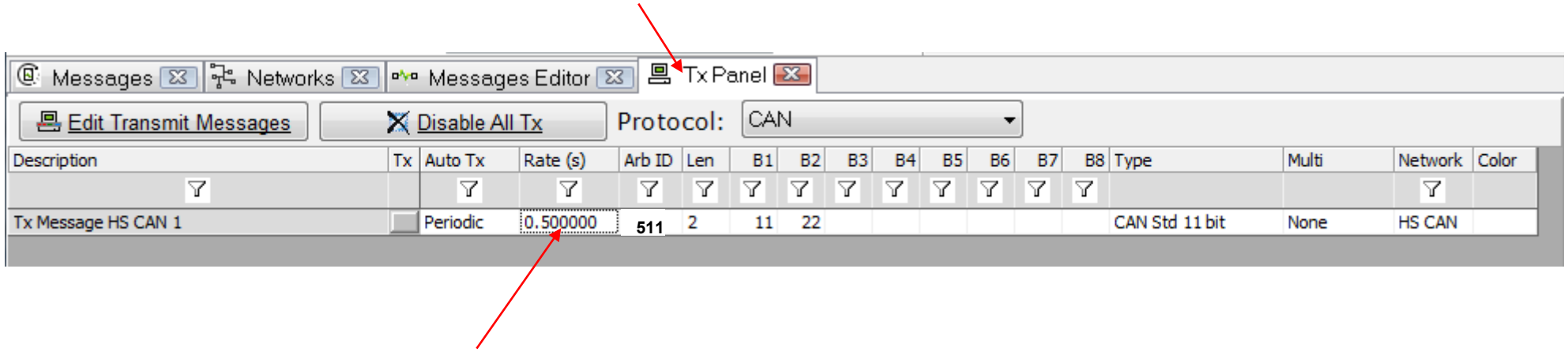
4) F6 hotkey

Return to Messages Tab



Press F6 on keyboard and see LED blink on the EVB

VSPY3 – Periodic Transmission



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…

Exercises

On Your Own…

- Transmit a CAN message at a periodic rate, using the LPIT0 interrupt from the previous section.
- Experiment with the ADC example. Perform a conversion and send its data in the periodic CAN message

SECURE CONNECTIONS
FOR A SMARTER WORLD