Getting Started with S32K Hands-on Workshop

Allen Willson

FAE

Automotive

October 2018 | AMF-AUT-T3375

SECURE CONNECTIONS FOR A SMARTER WORLD

First, Things First…

We need all of the hardware (thumb drives, EVB’s, etc.) to remain in the lab due to reuse across multiple hands-on sessions…

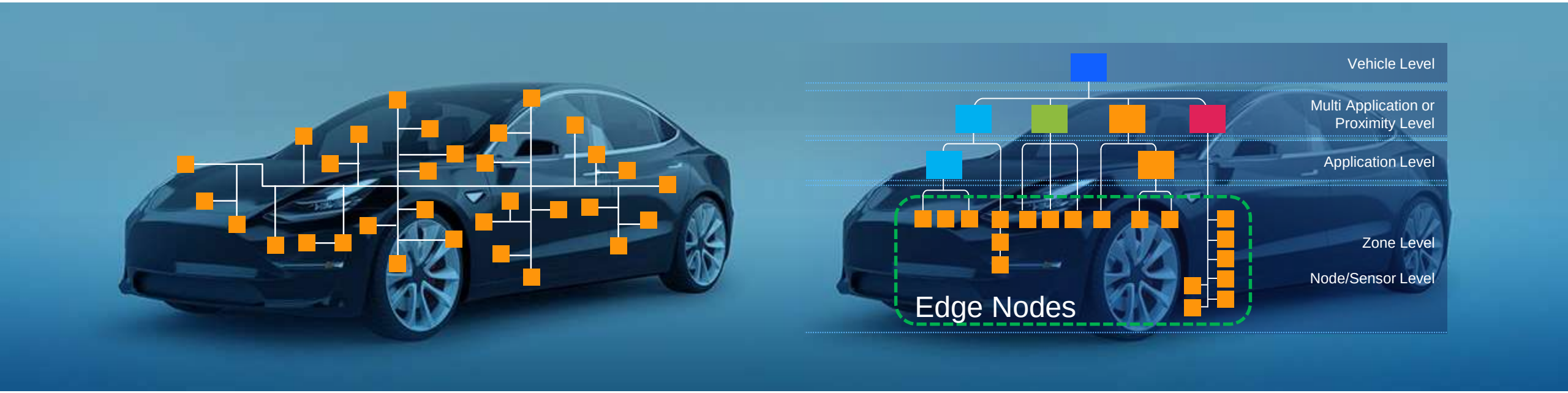
Agenda

- Brief S32K Product & Roadmap Overview
- Introduction
  - Hello World (GPIO) – Hands-On
  - Hello World + Clocks
  - Hello World + Interrupts – Hands-On
  - DMA
  - Timed I/O (FTM)
  - ADC
  - UART – Hands-On
  - SPI
  - CAN 2.0
  - CAN FD – Hands-On
- Appendix: S32 Design Studio Blank Project Steps

Product Overview & Roadmap

Introduction

Solutions for Edge Nodes – Today and Tomorrow



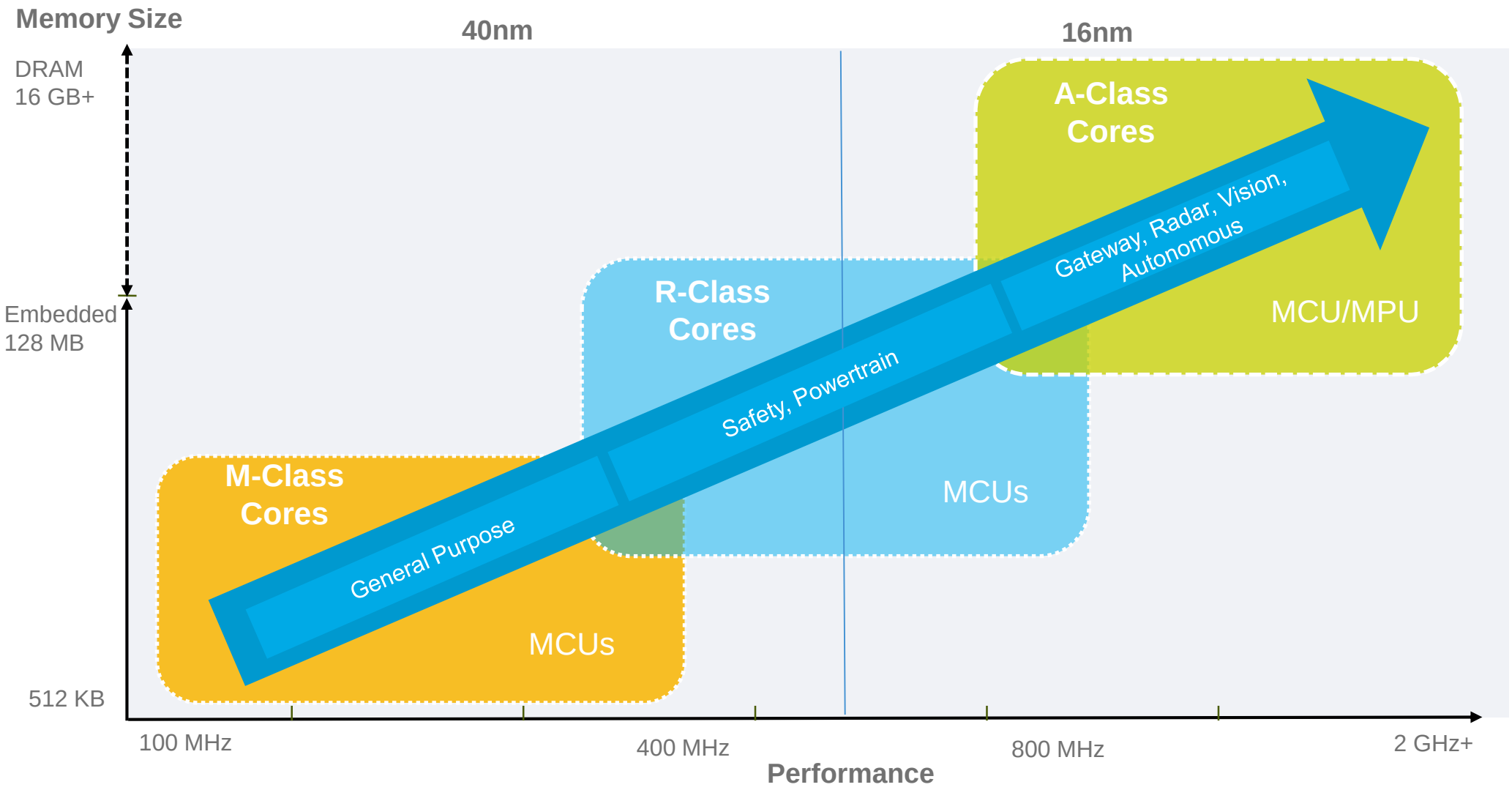
Today

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

Common Chassis Scalability



General Purpose and Integrated Solutions

Target Markets

General Purpose

Body Electronics
Lighting
HVAC

Other

Integrated Solutions

Motor Control
Window Lift
Pumps, Fans

Sensor Interfaces

Products

arm

KEA - S32K
The First Automotive MCU Designed for Software Engineers

S08 - S12 - MPC56xxB/C(Bolero)
- Shipping ~500Mu in 2016
- 8/16/32bit proven architectures

S12 MagniV
Shrink your application with MCU + HV analog integration

General Purpose

Integrated

Technology

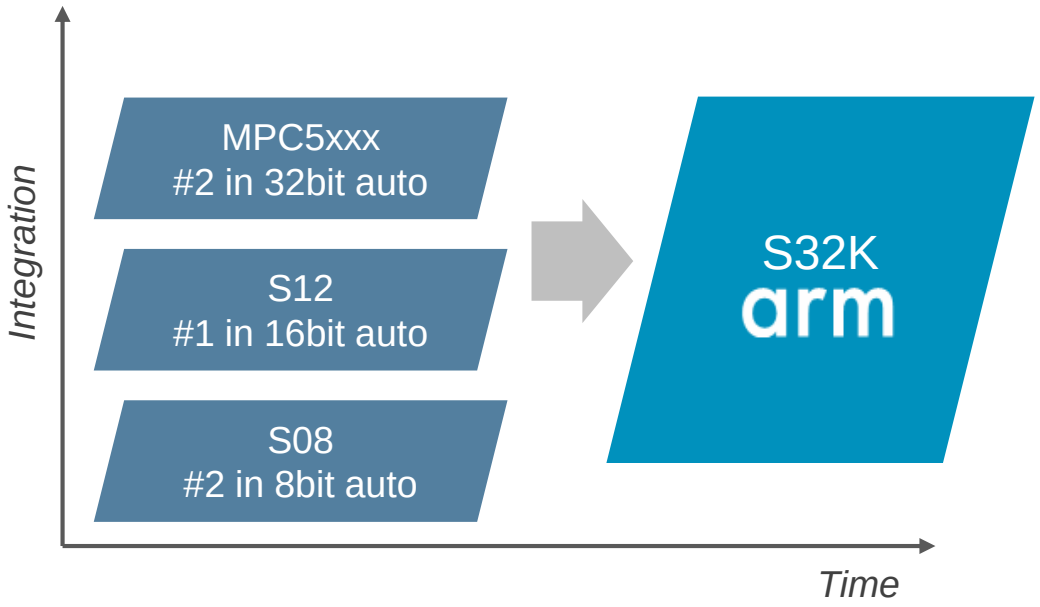
Software and Tools

Moving to ARM Cortex Architecture

HV Analog integration ASSP

S32K Value Proposition

ARM Cortex for Scalability



Future-proof Features

S32K1 / KEA Compatibility

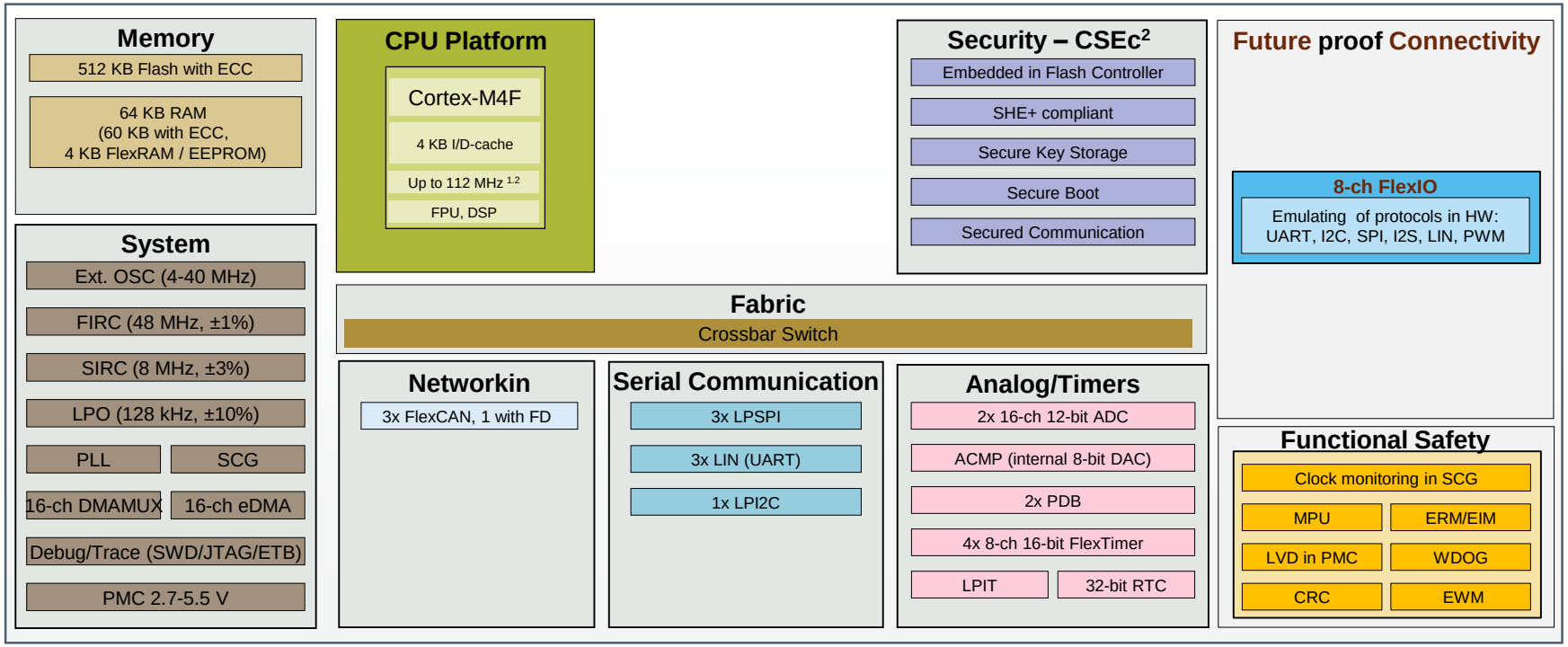
Pin Compatibility:
- Within S32K1xx product series
- Similar pinout as **KEA** products

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)

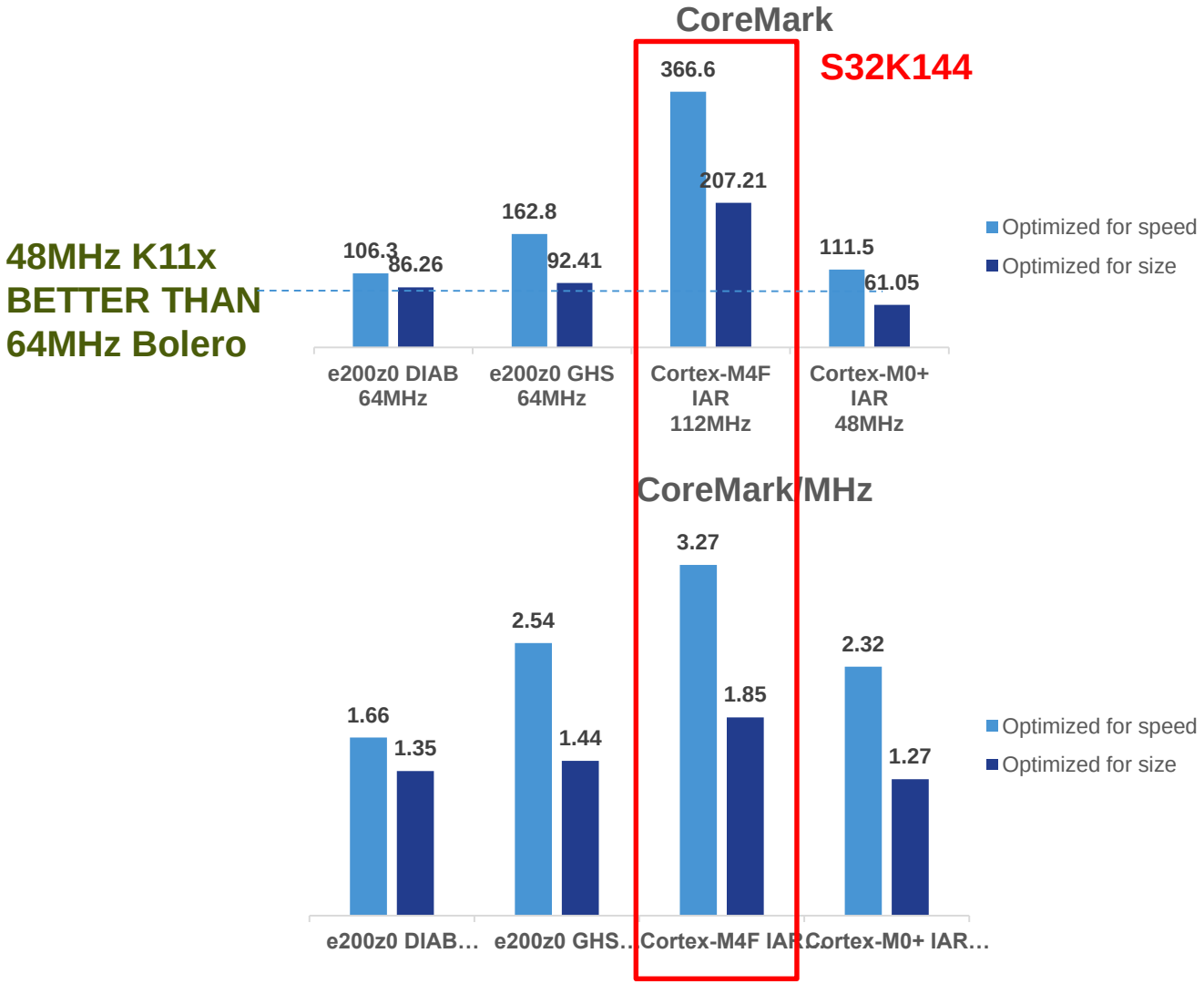
Key Features:

- **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

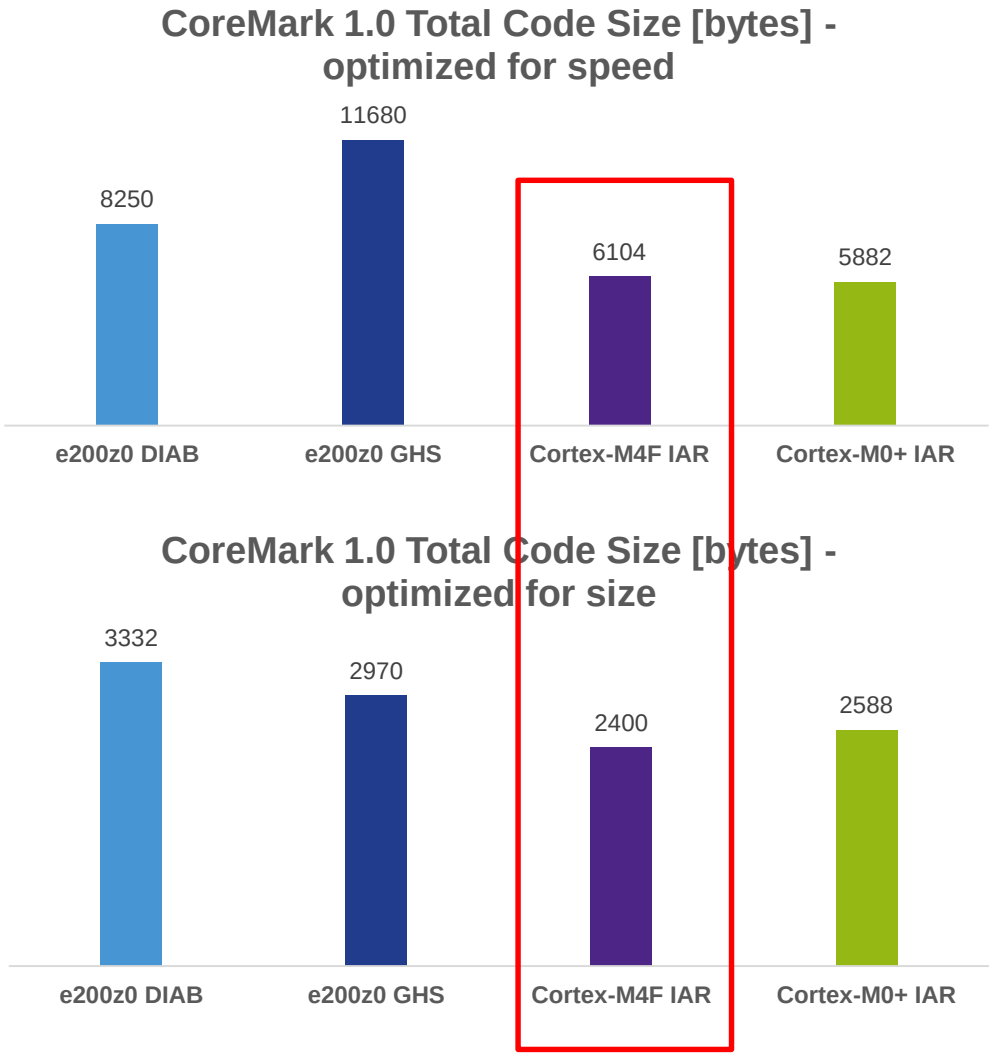
Footnote:

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

More Than a Standard EVB…

Get started fast

- Preinstalled firmware to explore features before installing tools
- Quick start guides

Add shields

- Arduino expandable
- Leverage NXP Portfolio MCU + SBC + Phys + Drivers

- Injector driver demo
- BLDC motor
- DC motor
- Audio / SAI expansion
- Ethernet Phys
- Low power demonstrator
- Matrix LED driver
- And more!

Reuse HW

- Reference HW solution
- Following HW design guidelines (Appnote AN5426)

Reuse SW

- Production grade Software development kit
- Cookbook style simple code examples

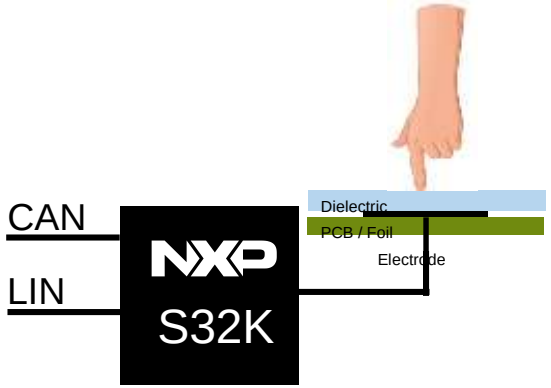
System Solutions – Released Already

ISELED Driver

- High-speed communication for creating dynamic lighting effects
- ISELED Driver for S32K
- Using FlexIO and SPI
- SDK and Autosar

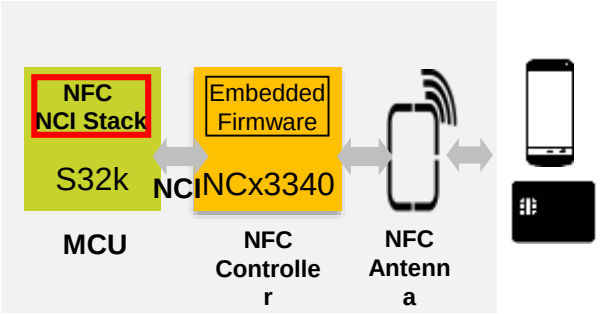
Touch Sense library

- 1D Touch Library
- SDK and Autosar
- Single chip solution for **automotive** TS.
- Suitable for up to 10 electrodes



NFC Stack

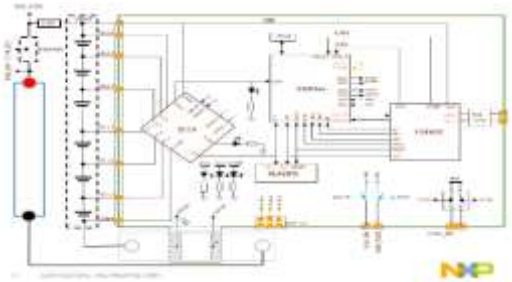
- **Interface** between MCU and NFC controller
- Specified by **NFC Forum**
- **Eases integration** of NFC controllers
- SDK and Autosar



BMS Reference Design

Creating safety.
With passion.

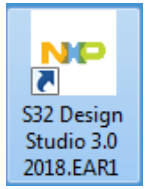
Turnkey solution for Safety Applications up to ASIL-C
4 NXP Devices:
- S32K144
- KEA
- SBC
- Battery Cell Management

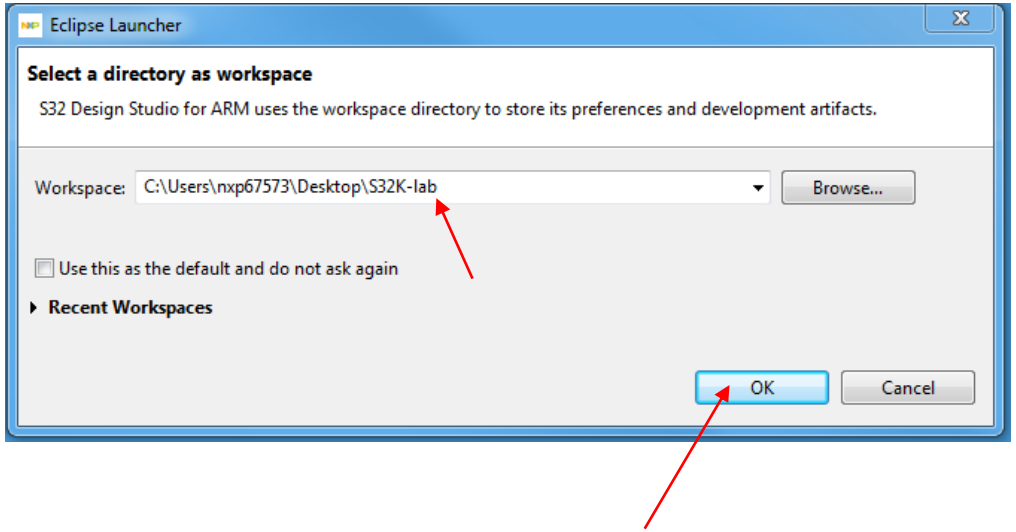


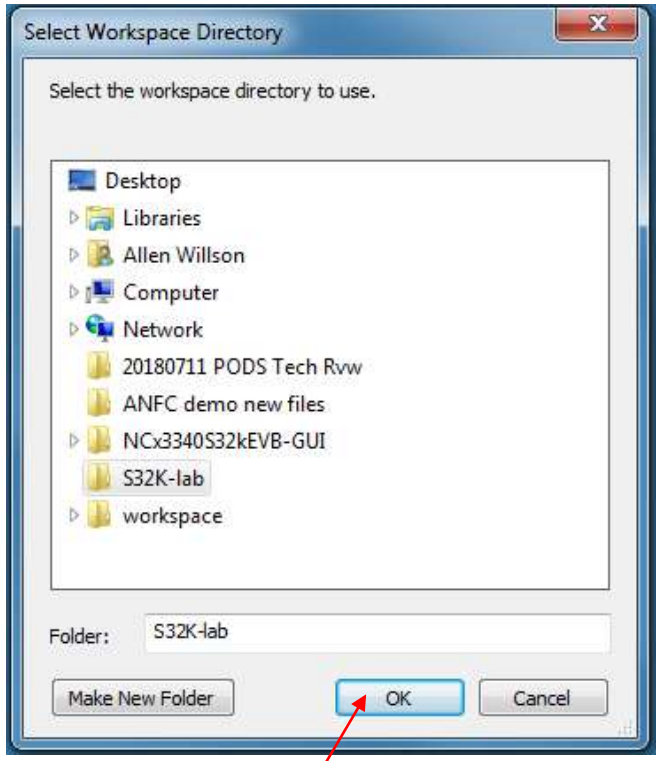
0. Start S32 Design Studio and Import the Labs

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

Toolbar and Menubar

Perspectives/Views

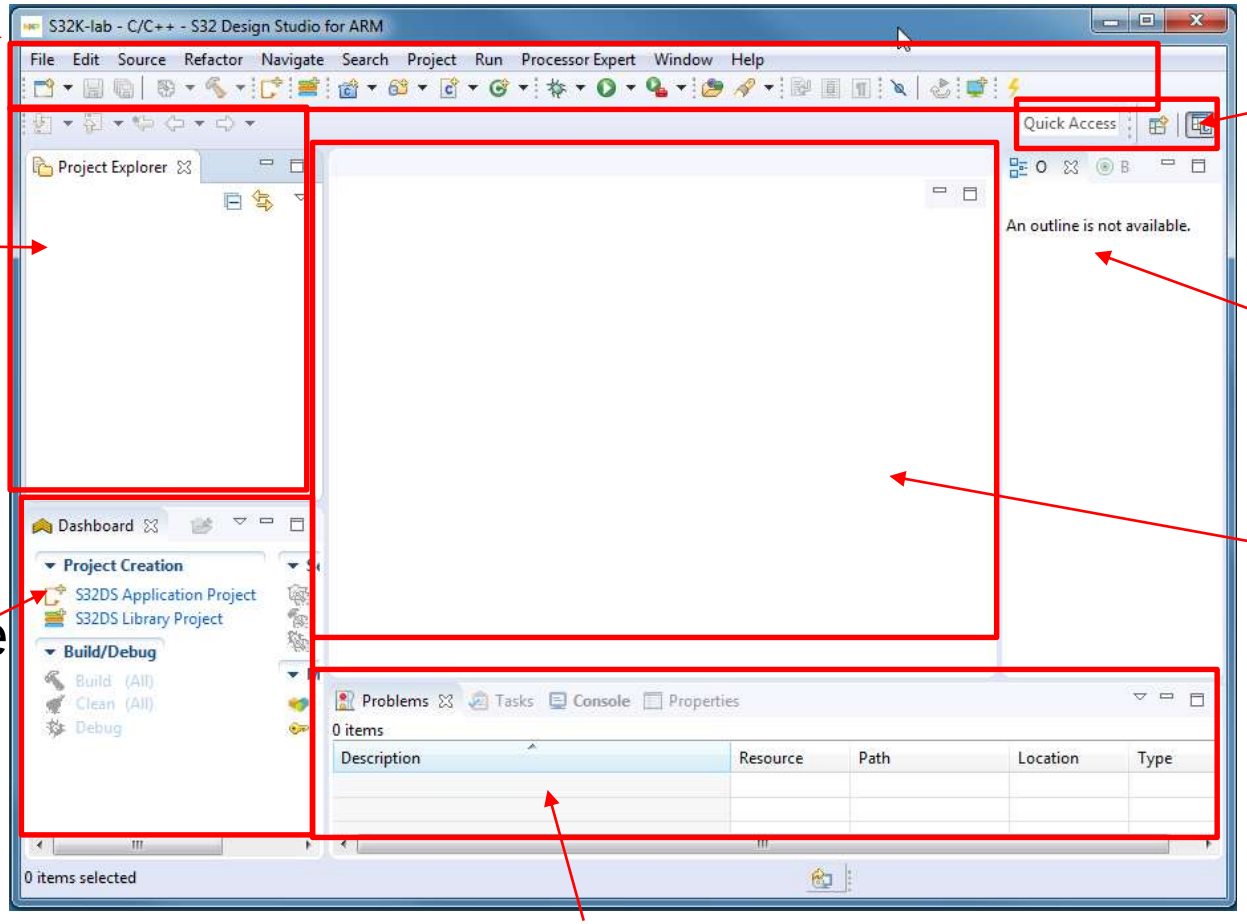
Project Pane

Outline

Editor

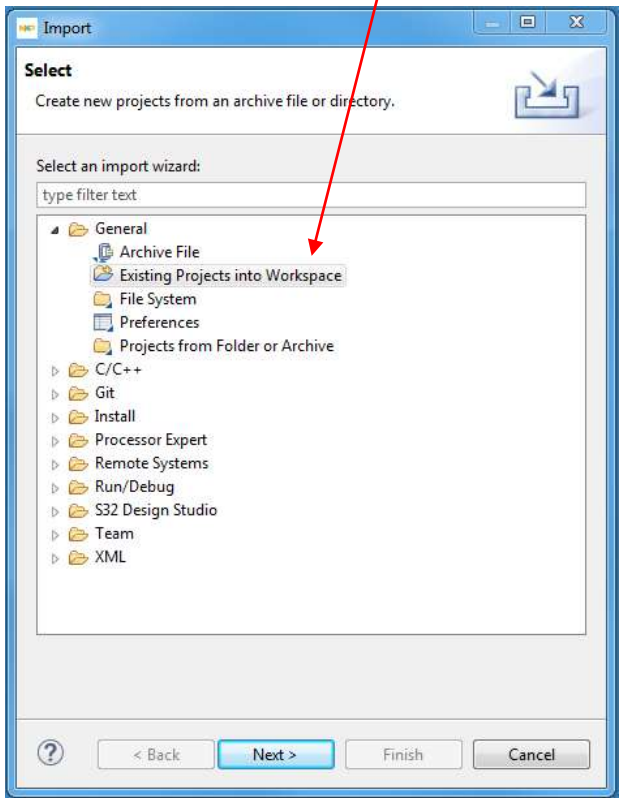
Command Pane

Build Console & Navigator

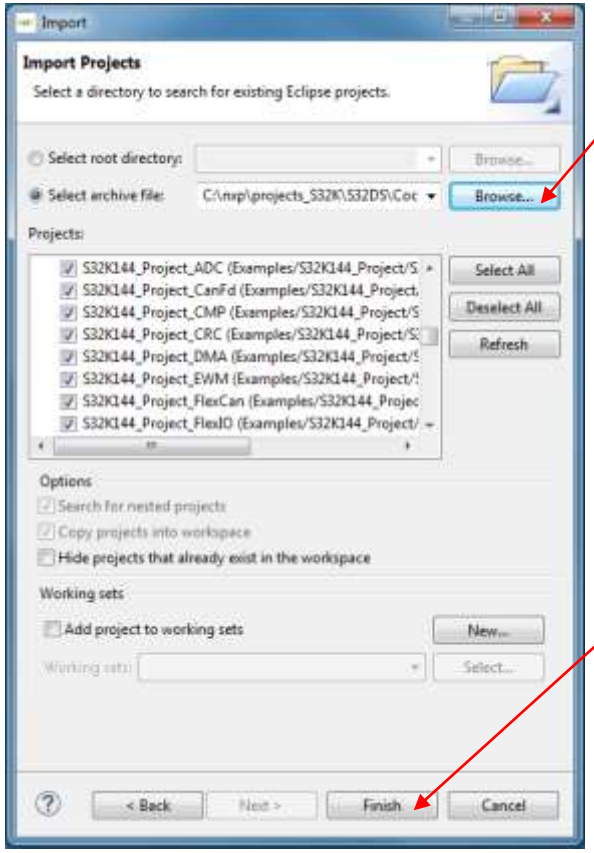


Import the Labs

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

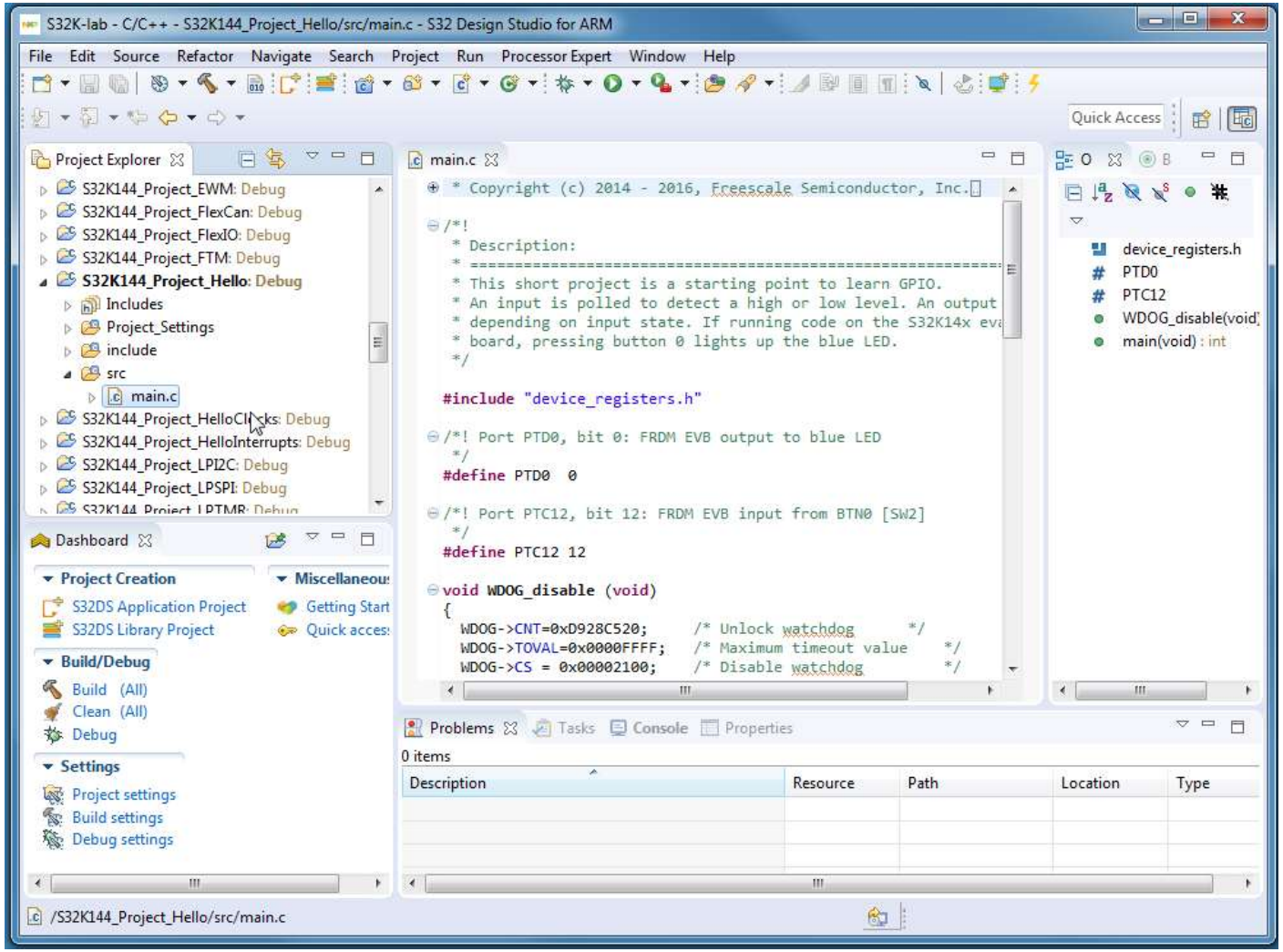


2. Select Archive File→ Browse and select file



3. Finish

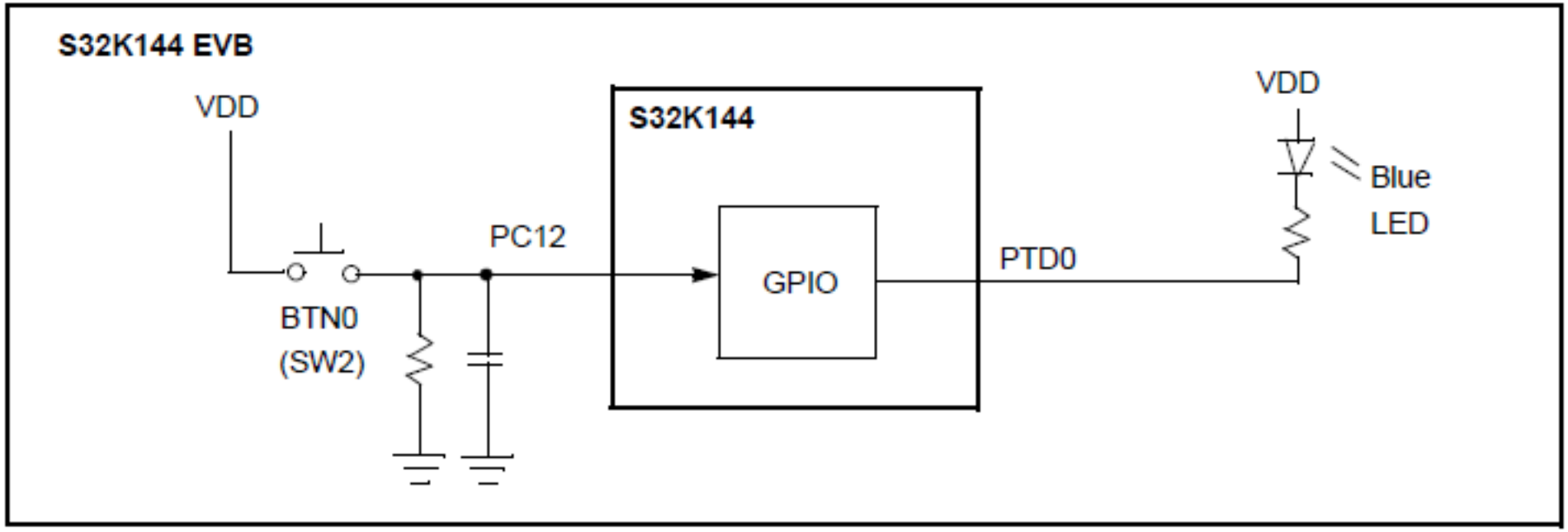
Take a Tour of S32 Design Studio



1. Hello World (GPIO)

1. Hello World: Introduction

<u>Summary</u>: 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

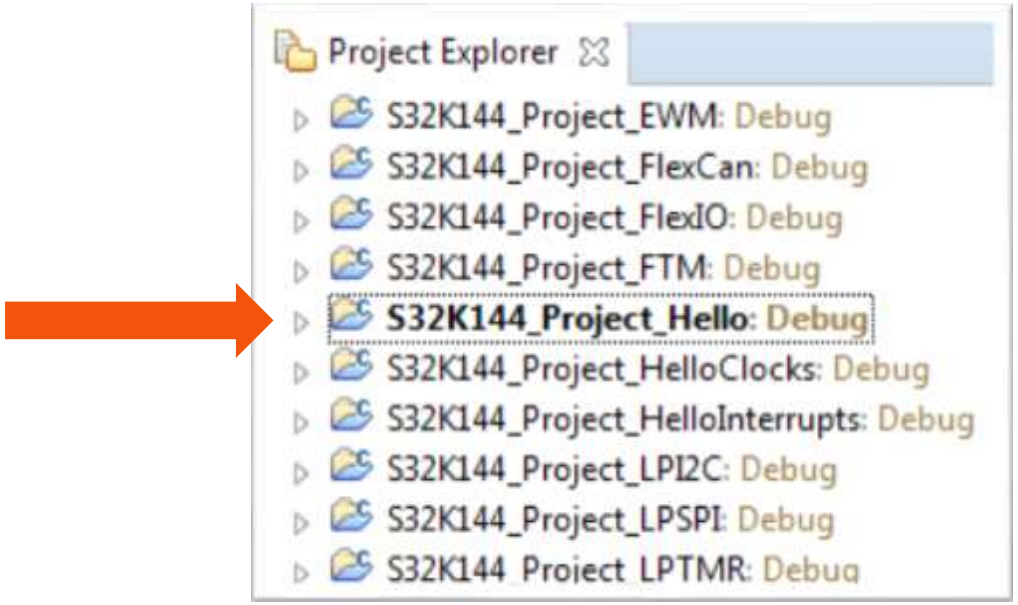
- Out of reset, clocks are not enabled to peripherals. (Done in PCC, Peripheral Clock Controller module)

- 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.

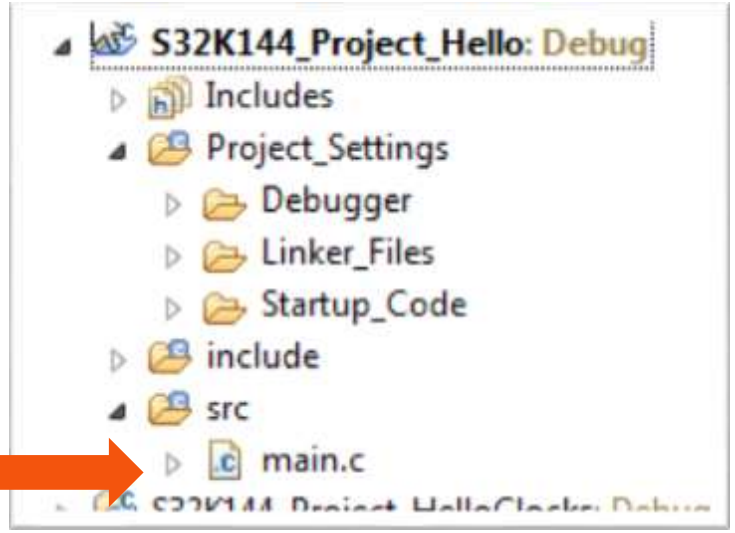
1. Hello World: Build 1 of 2

1. Select project “Hello”



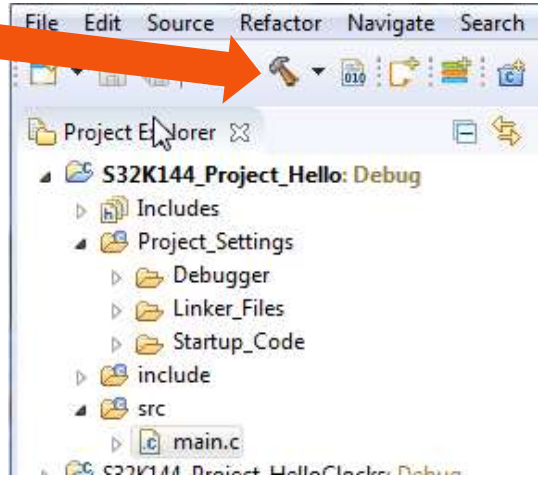
1. Hello World: Build 2 of 2

1. Expand Hello, then src folders

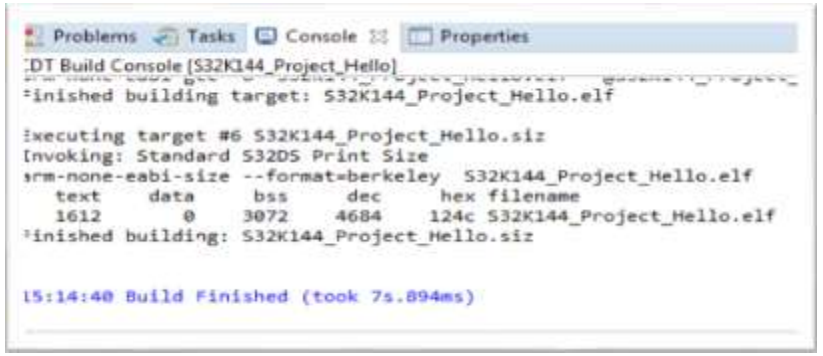


2. Double click on “main.c” to examine code

3. Click on hammer icon to build



4. Click on Console tab to ensure the build finished

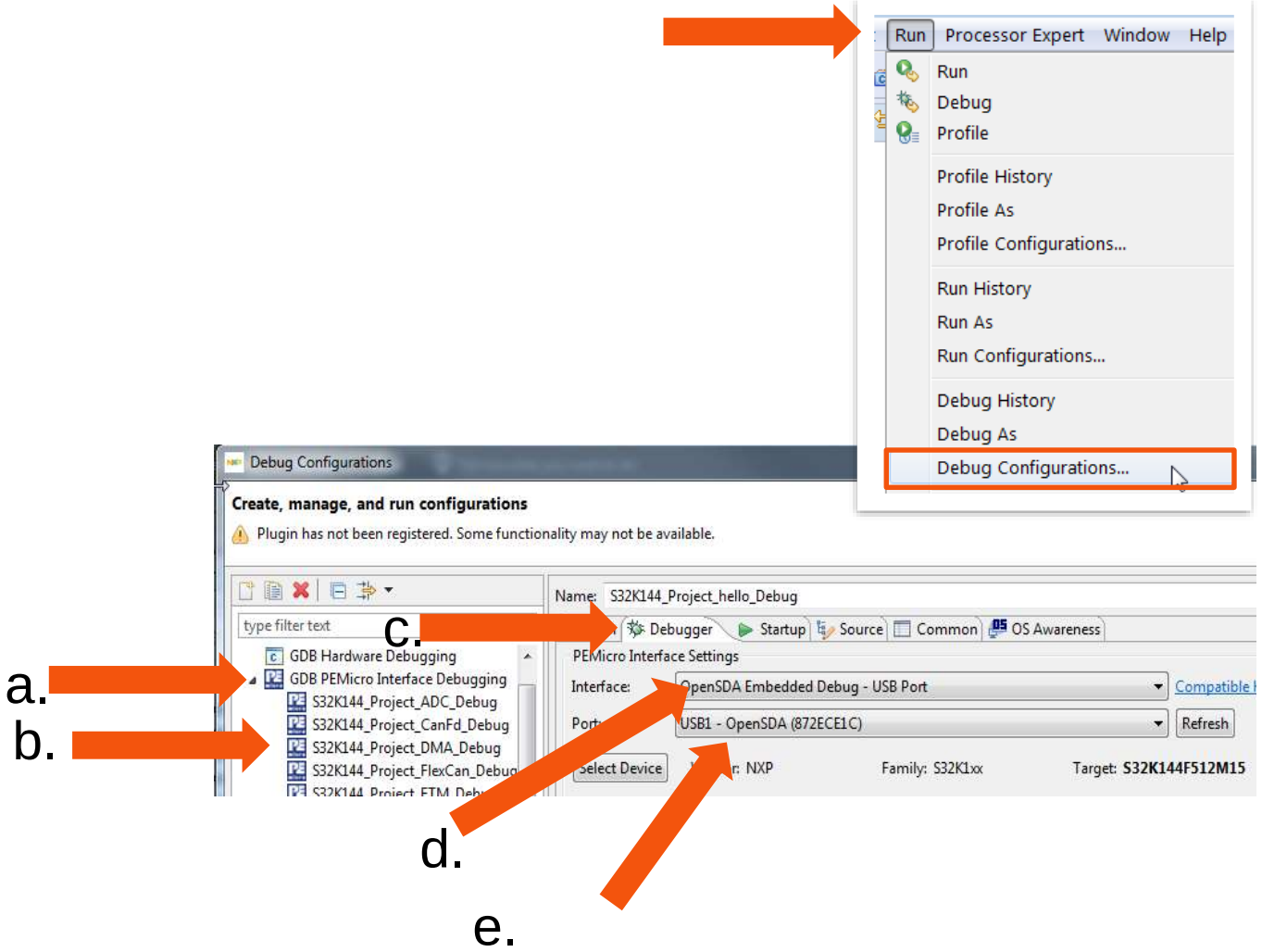


1. Hello World: Debug Configuration

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. Verify Port is filled.

3. Click Debug on bottom right corner

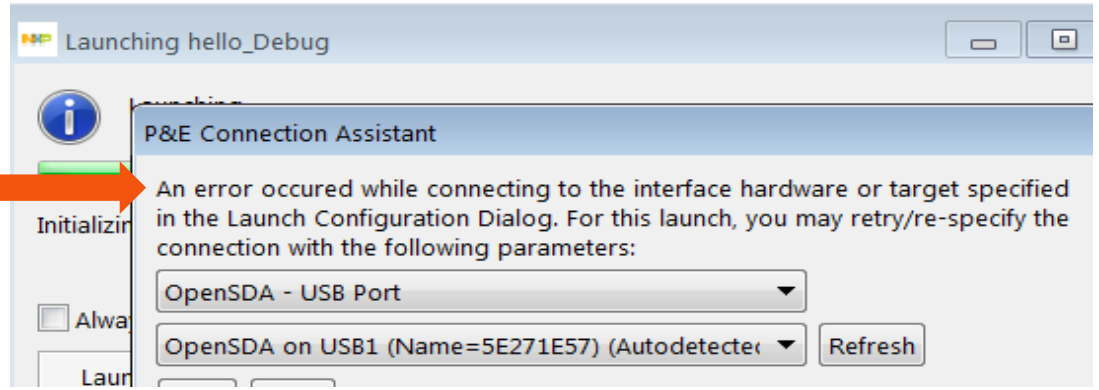


1. Hello World: Debug Tip

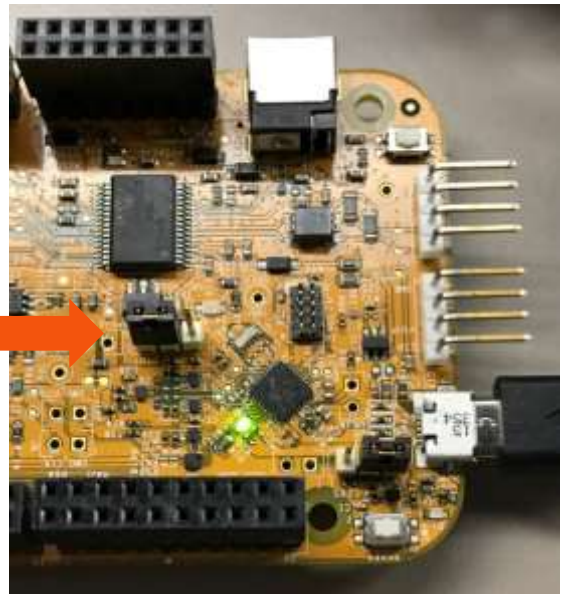
If you get this error message at Debug, you likely need to fix your power configuration.

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

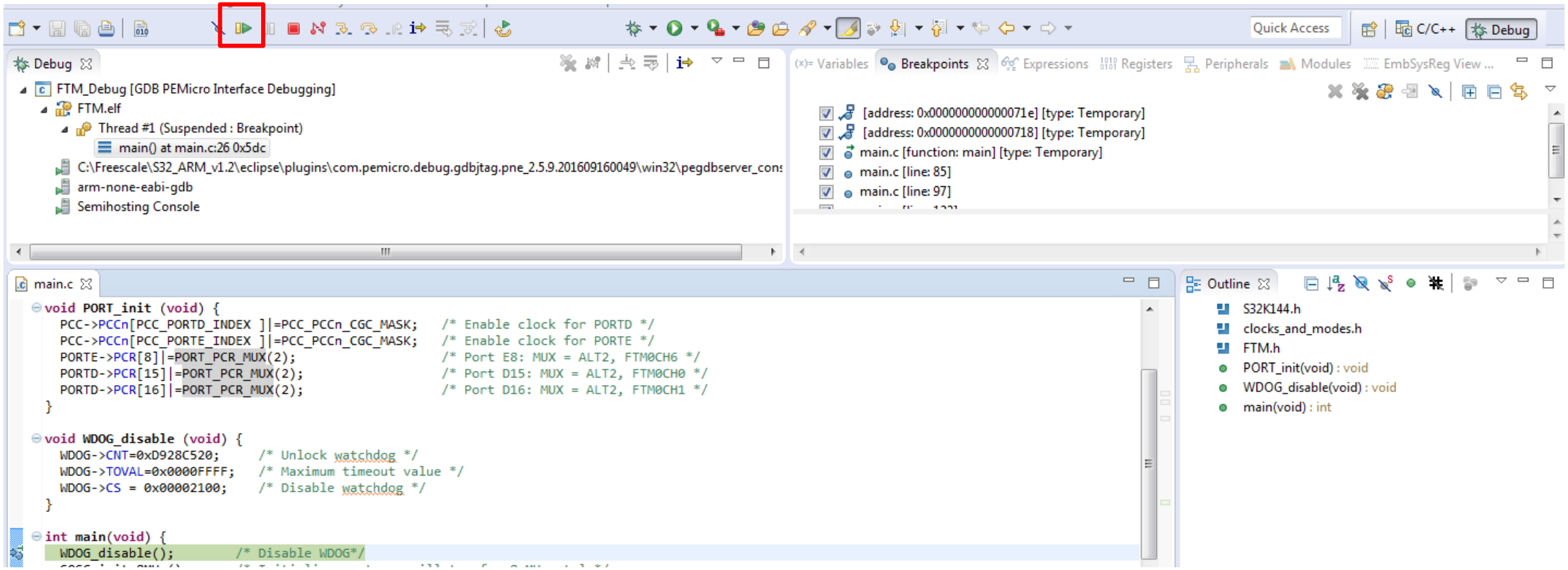


J107: move jumper to right if external 12V is not connected



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

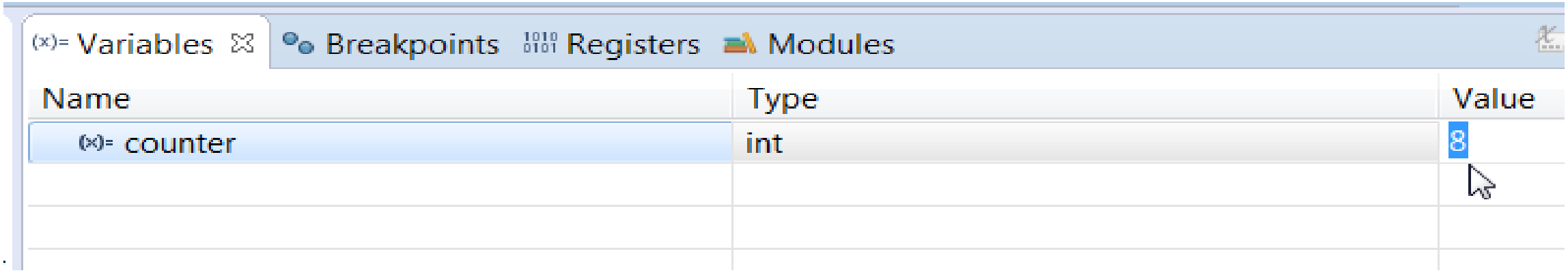
- Resume [“go”, “run”]

- Suspend [“stop”, “halt”]

- Terminate [exit debugger]

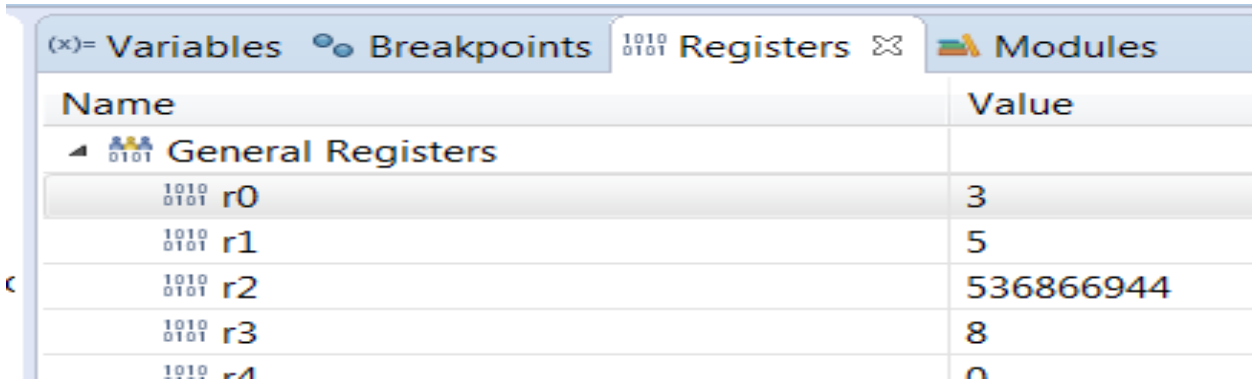
Debug Basics: View & Alter Variables

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

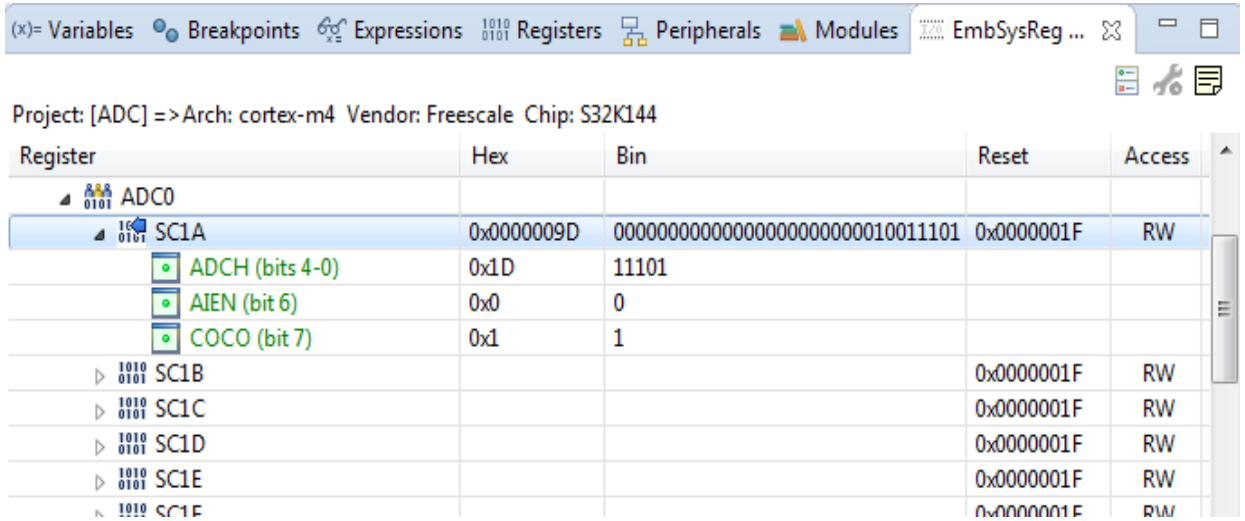


Debug Basics: View & Alter Registers

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

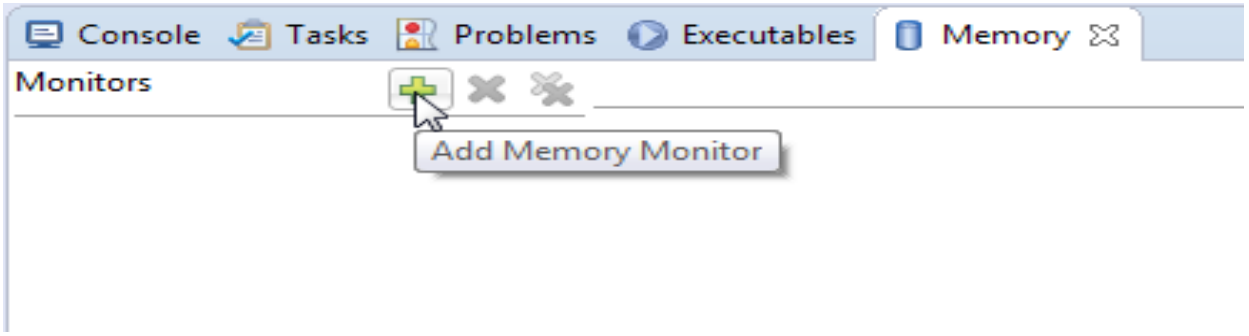


- View peripheral registers in the EmbSys Registers tab

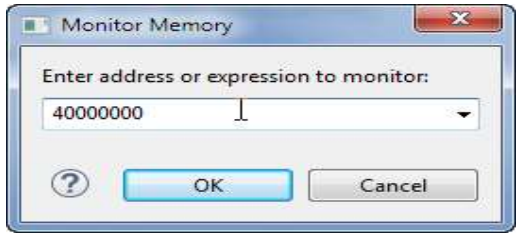


Debug Basics: View & Alter Memory

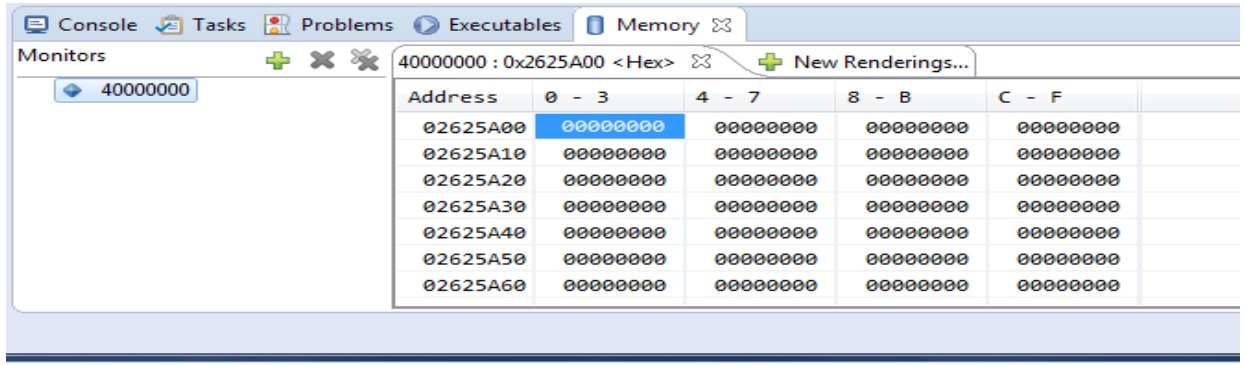
- Add Memory Monitor



- Select Base Address
- to Start at : 40000000



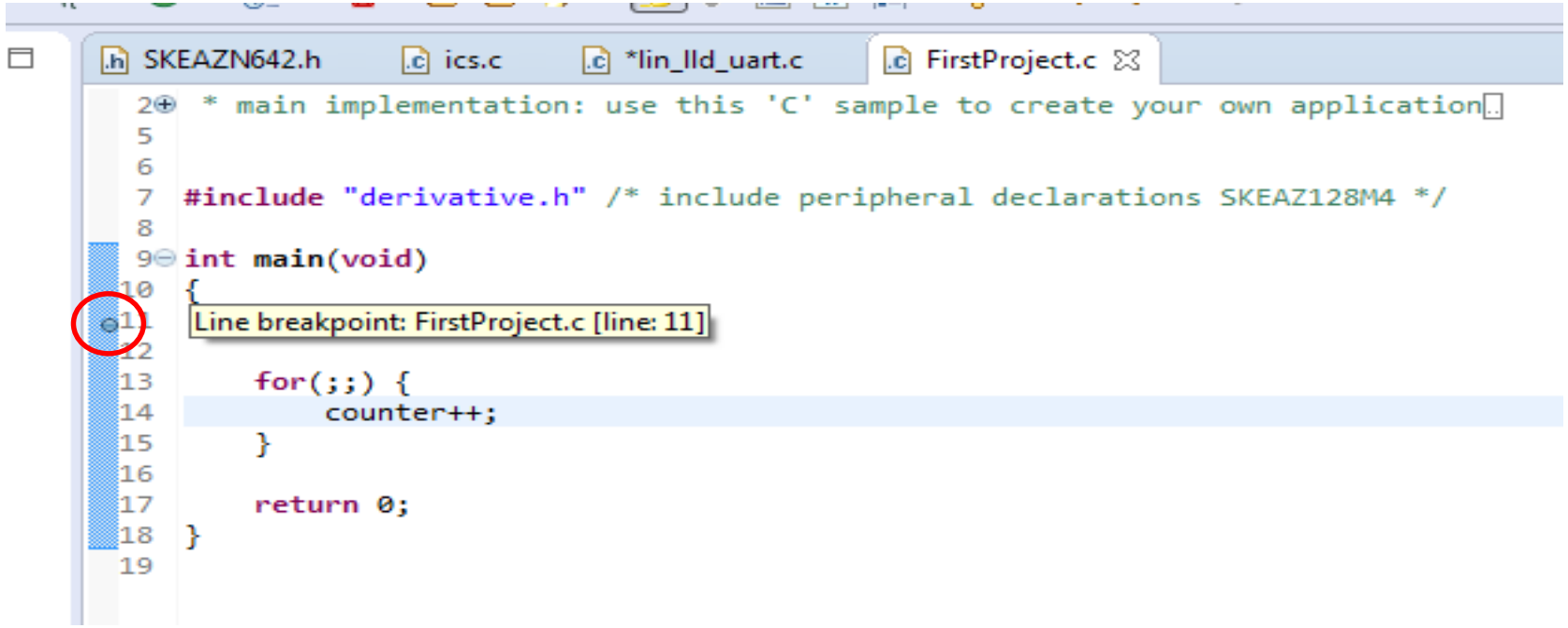
- View Memory



Debug Basics: Breakpoints

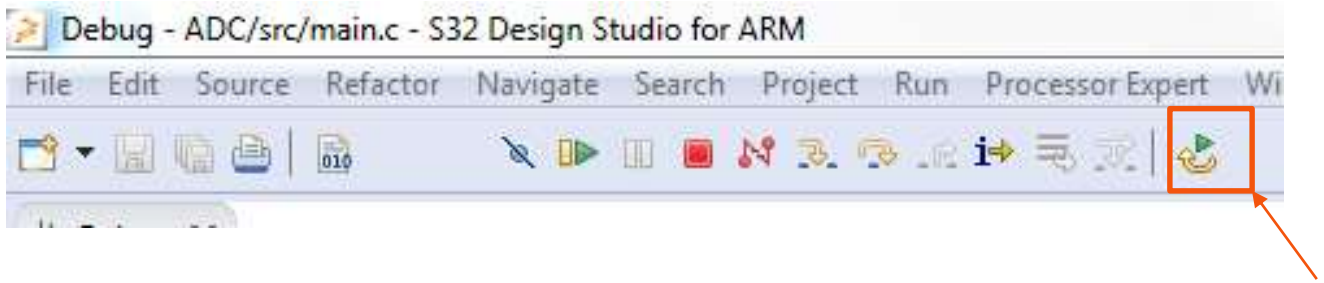
Add Breakpoint: Point and Click

- Light blue dot represents debugger breakpoint



Debug Basics: Reset & Terminate Debug Session

- Reset program counter



- Terminate Ctl+F2

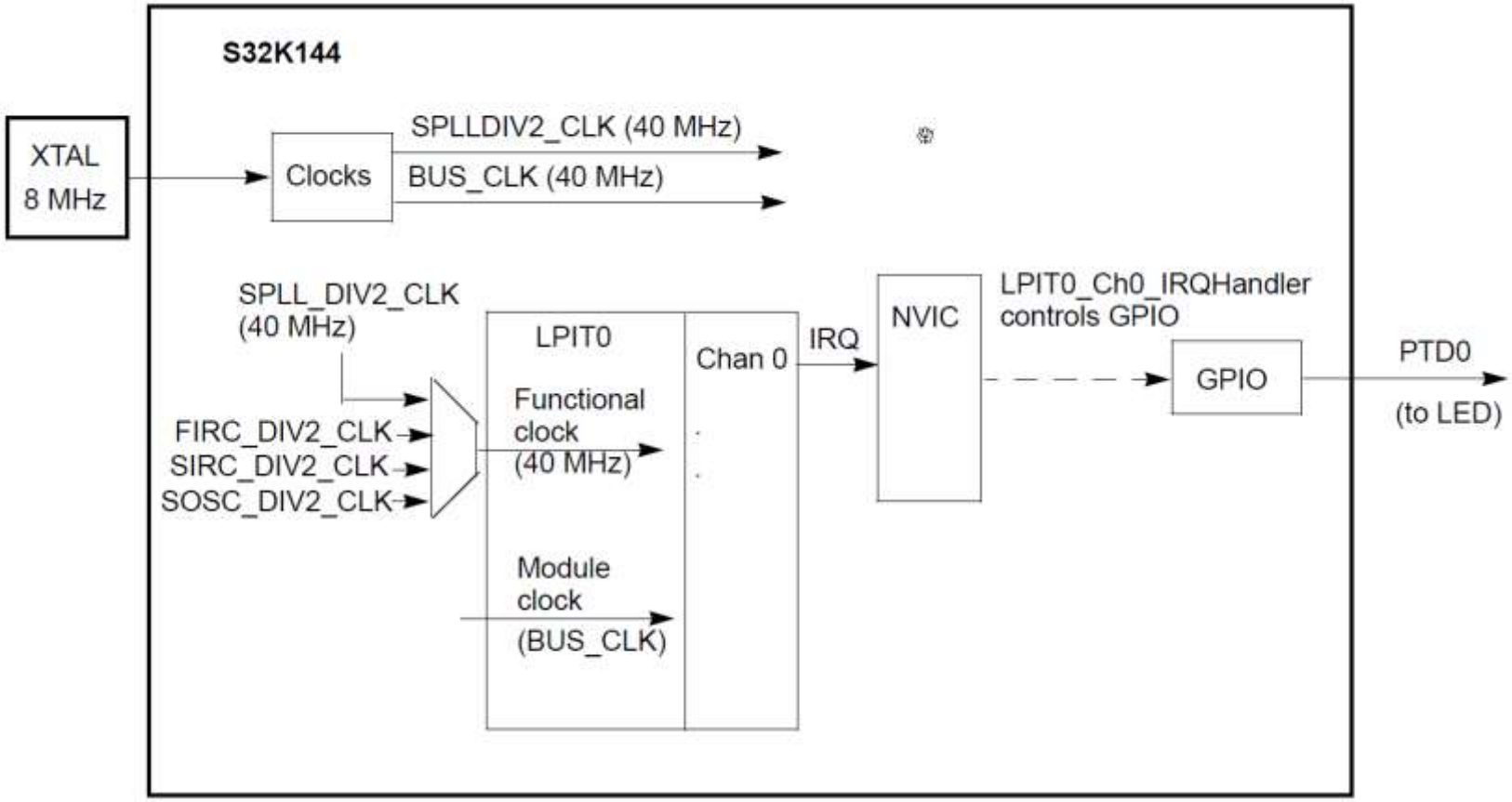
Always terminate a debug session when finished.

3. Hello World + Interrupts

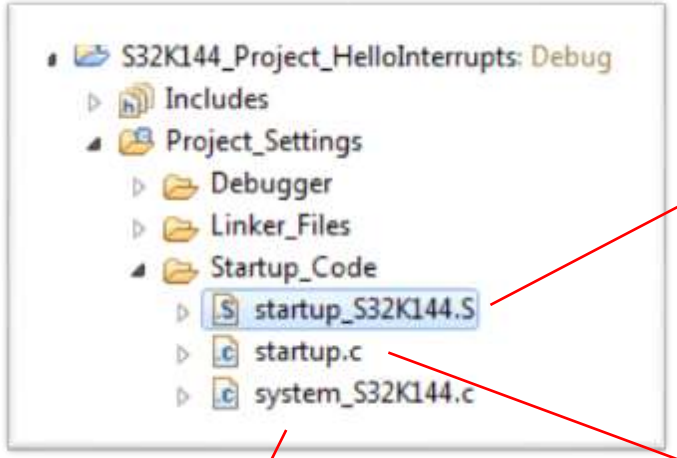
3. Hello World + Interrupts: Introduction

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*/
```

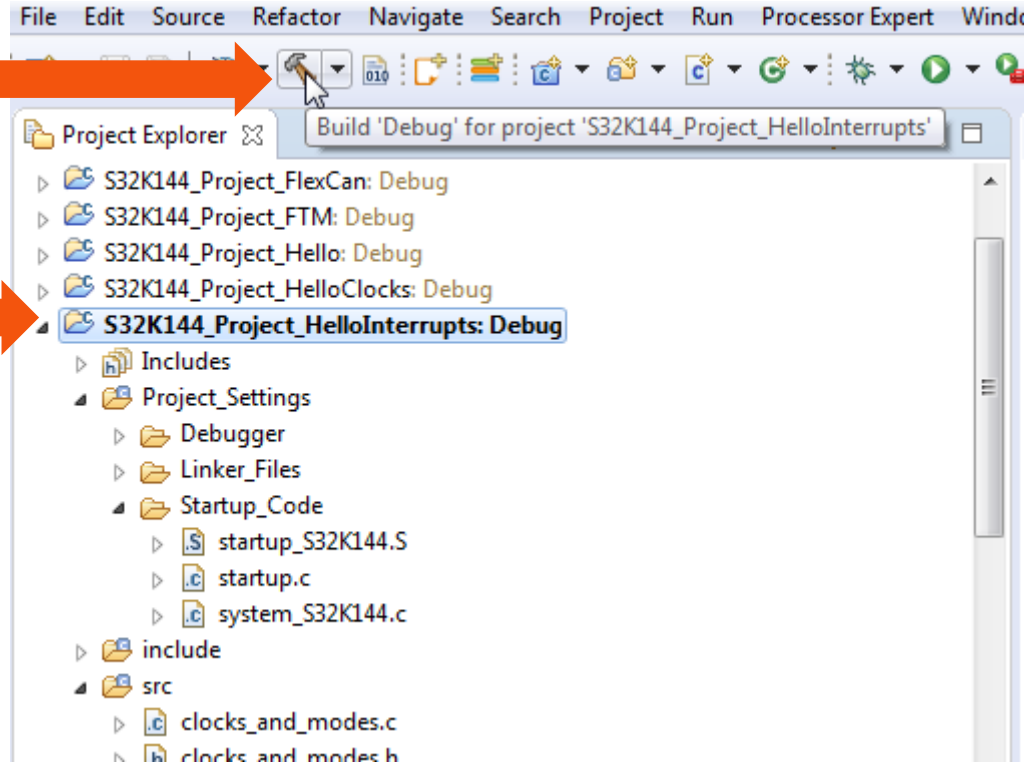
3. Hello World + Interrupts

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

2. Double click on HelloInterrupts project

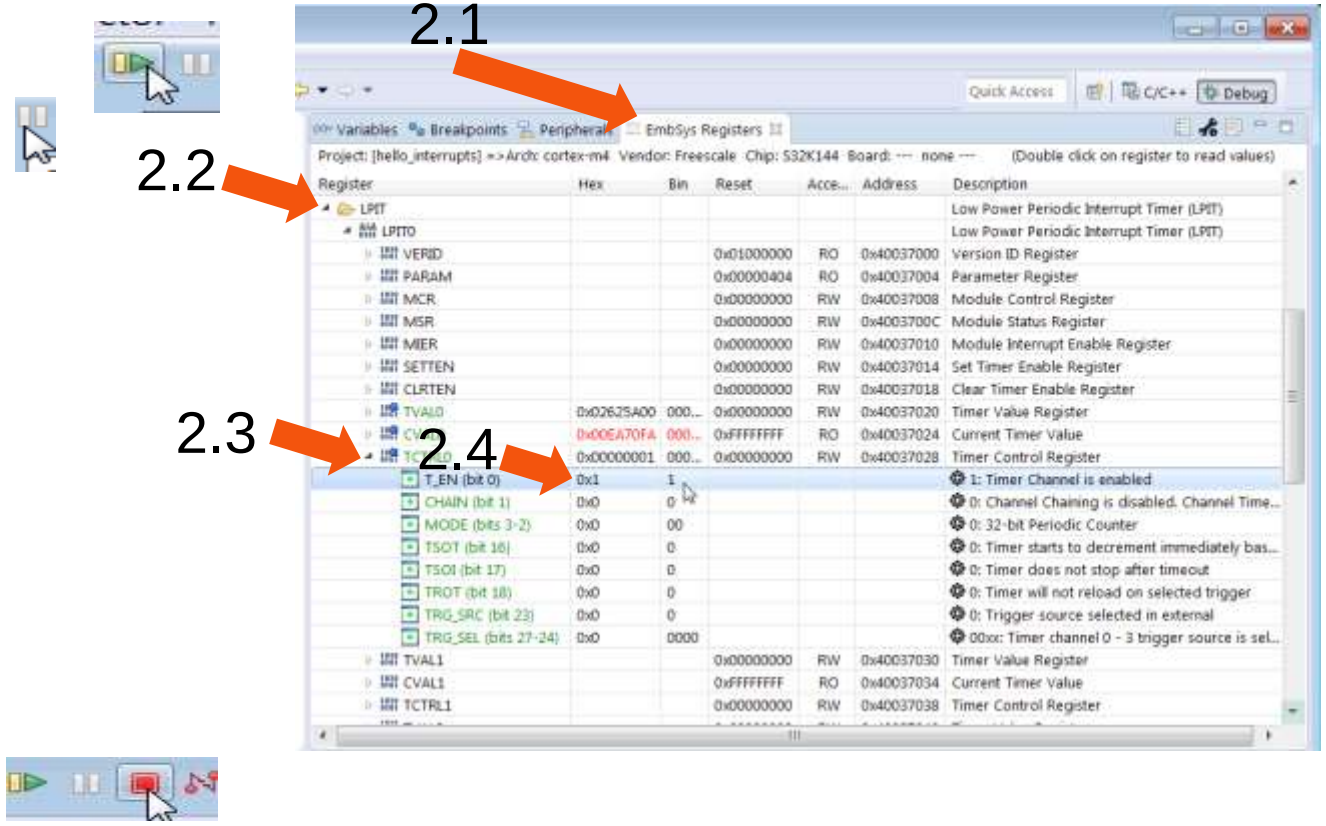
3. Click on Build icon

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:
   1. Click EmbSys Registers tab
   2. Scroll down list about half way and expand LPIT & LPIT0
   3. Expand register TCTRL0
   4. 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

- 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
  - Use debugger to find the offending line of code
    - Register View

How to Break an ARM

Set compiler optimization (-O2); Clean; Build; Debug

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) */
}
```

Move this line of code

After

```
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

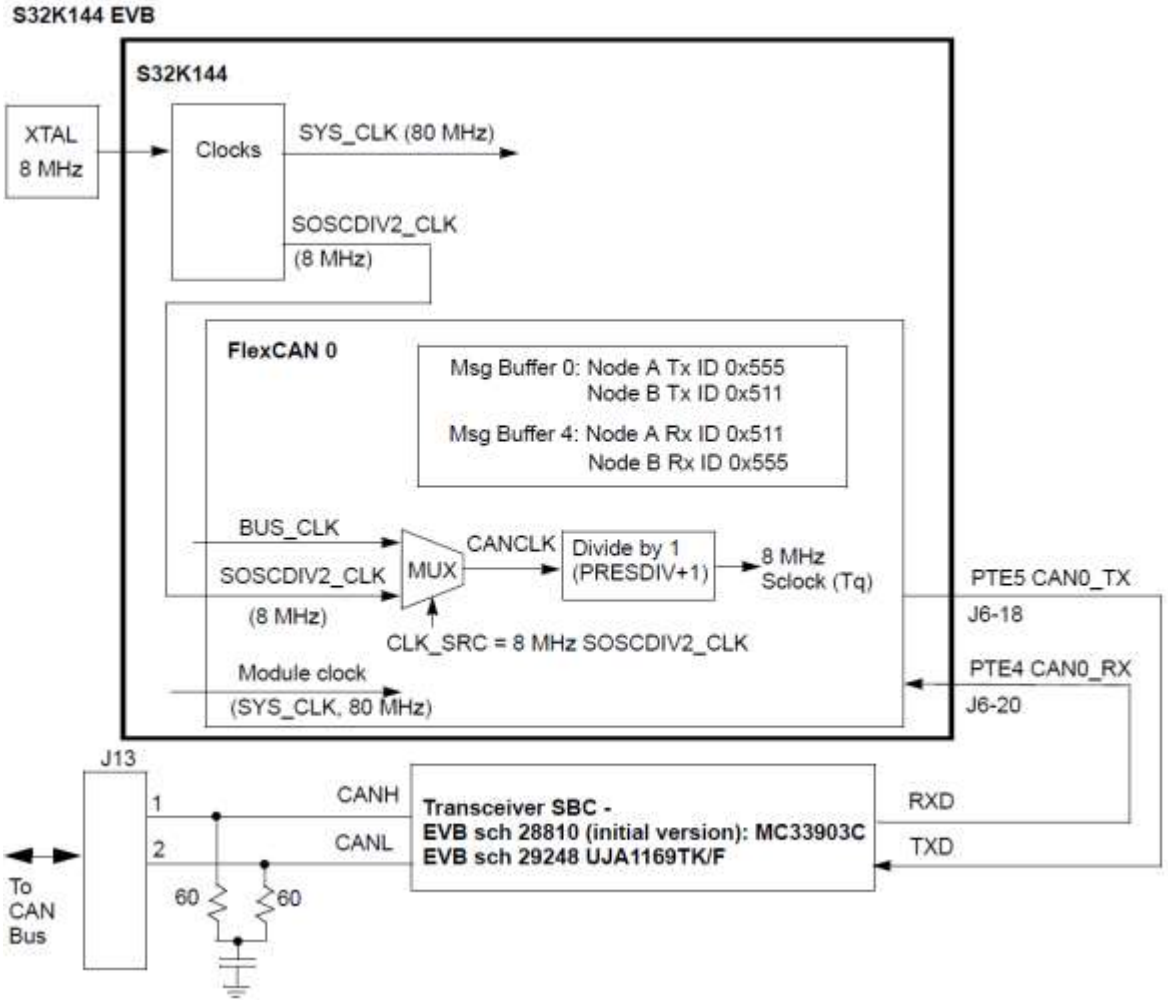
```
void __attribute__((optimize("O0"))) LPIT0_Ch0_IRQHandler (void)
{
  LPIT0->MSR |= LPIT_MSR_TIF0_MASK; /* Clear LPIT0 timer flag 0 */
  PTD->PTOR |= 1<<0;                /* Toggle output on port D0 (blue LED) */
  lpit0_ch0_flag_counter++;         /* Increment LPIT0 timeout counter */
}
```

9. Classic CAN & CAN-FD

9. CAN 2.0: Introduction

Summary

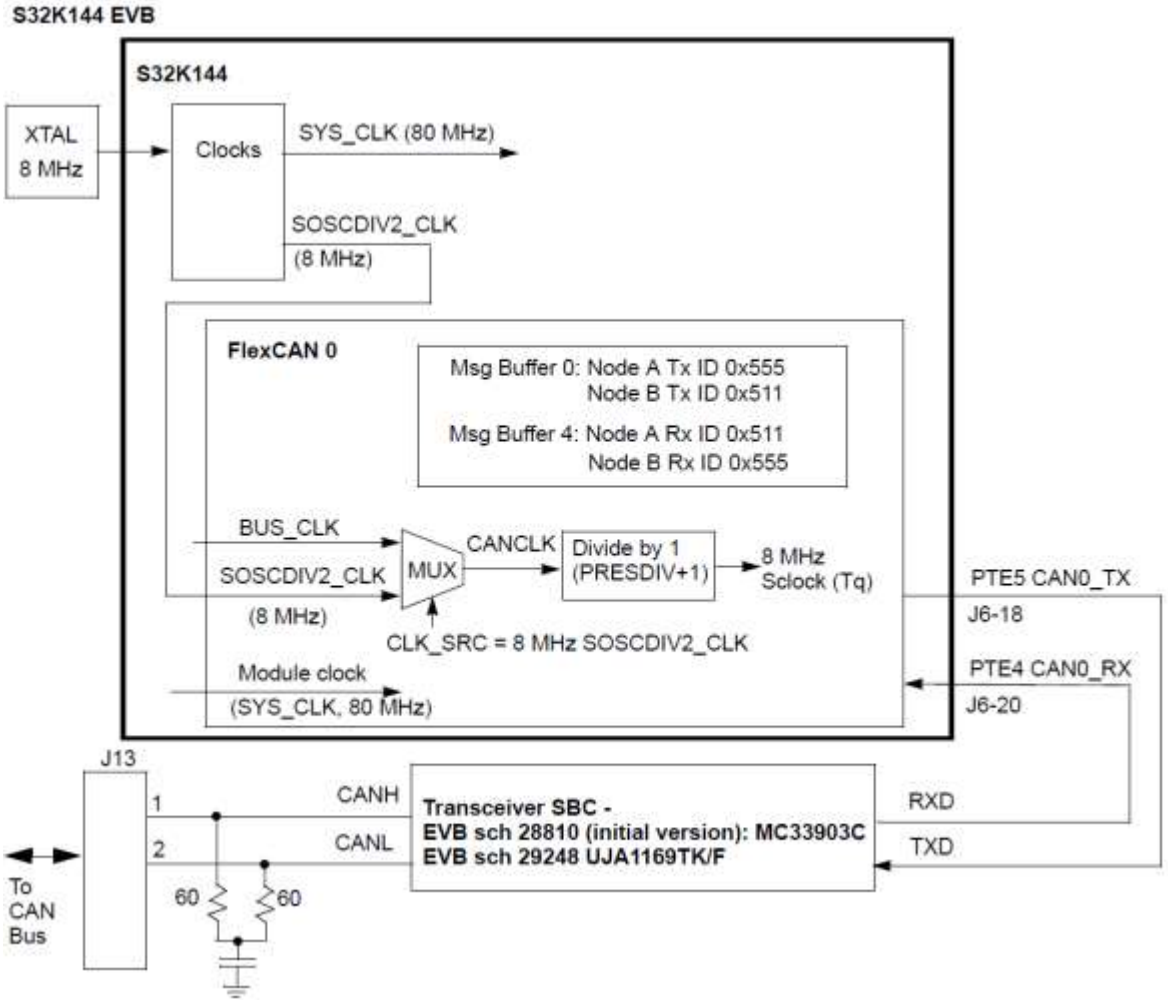
- FlexCAN module is initialized for 500K bps based on an 8 MHz crystal
- Two boards can be used to communicate using an CAN cable (CAN HI, CAN LO, ground)
- Software builds are to be configured for:
  - Node A or B
  - SBC MC33903 or UJS1169



10. CAN FD: Introduction

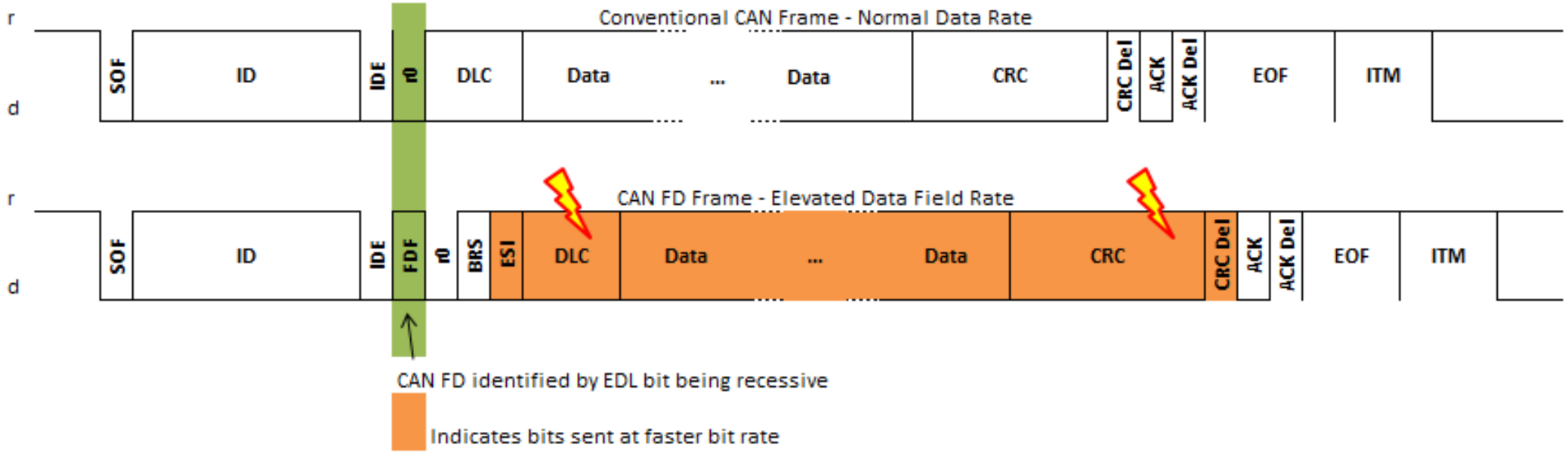
Summary

- FlexCAN module is initialized for 500K bps/ 2 MHz based on an 8 MHz crystal
- Two boards can be used to communicate using an CAN cable (CAN HI, CAN LO, ground)
- Software builds are to be configured for:
  - Node A or B
  - SBC MC33903 or UJS1169



10. CAN FD Frame Overview

- CAN FD stands for CAN with Flexible Data-Rate
- CAN FD was a proposal by Bosch to:
  - Increase the baud rate of the data portion of a CAN message
  - Increase the number of data bytes that can be sent in a single CAN message to up to 64 bytes
  - No changes to arbitration field allow for existing physical layers to be used



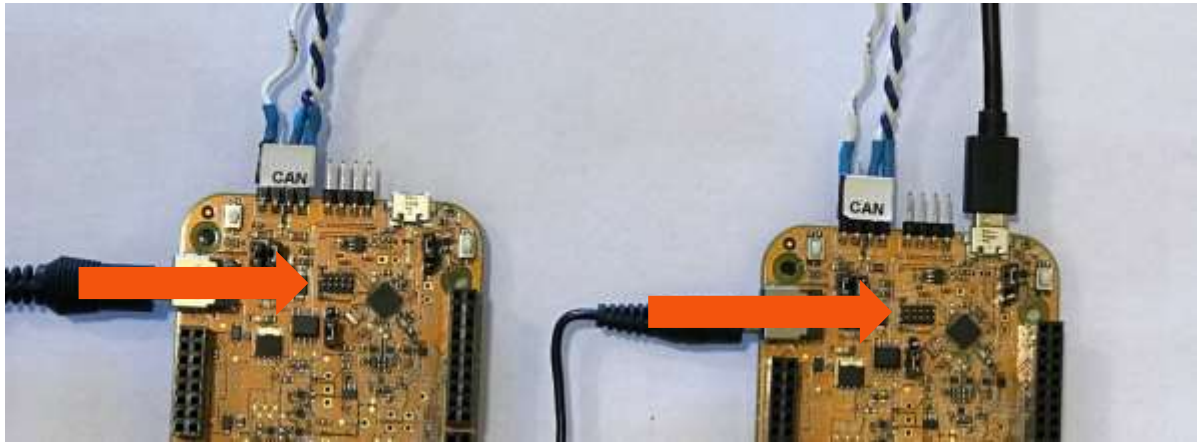
10. CAN FD: Key Points

- Key design parts described:
  - CAN 2.0 vs CAN FD (ISO) initialization comparison.
  - CAN FD timing calculations for arbitration and data phases
  - New message buffer structure for larger data frames
  - Transceiver Delay Compensation (TDC) for faster baud rates
- Modify FlexCAN.h file for Node A/B ~~and two SBCs~~:

SBC (data phase)	Node A	Node B
~~MC33903 (max. 1 MHz)~~	#define NODE_A ~~#define SBC_MC33903~~	// #define NODE_A ~~#define SBC_MC33903~~
~~UJA1169 (max. 2 MHz)~~	#define NODE_A ~~// #define SBC_MC33903~~	// #define NODE_A ~~// #define SBC_MC33903~~

10. CAN FD: Board Connections, Defines

- Move power supply selection jumper to use external 12V (away from CAN connector; see red arrows for jumper)



- Connect CAN, power cables as shown
- USB cable can still be connected for downloads, debug.
- Program Node A and Node B.
- High level summary:
  - Node B (Power up this EVB first)
    - Loop: For every received message from Node A, one is transmitted back.
  - Node A
    - Transmits initial message to Node B
    - Loop: For every received message from Node B, one is transmitted back.

SECURE CONNECTIONS
FOR A SMARTER WORLD

www.nxp.com