

HANDS-ON WORKSHOP: MPC574XG SDK FOR GATEWAY

1ST SDK FOR POWER ARCHITECTURE

JULY 2017



SECURE CONNECTIONS
FOR A SMARTER WORLD

NXP and the NXP logo are trademarks of NXP B.V. All other product or service names are the property of their respective owners. © 2017 NXP B.V.

PUBLIC





AGENDA

- Introduction
 - S32 Software Development Kit (SDK)
 - S32 Design Studio
- Hands-on
 - Blinking LED
 - ENET – CAN Communication





01.

Introduction

S32 Software Development Kit (SDK) – Goals

Easy prototype

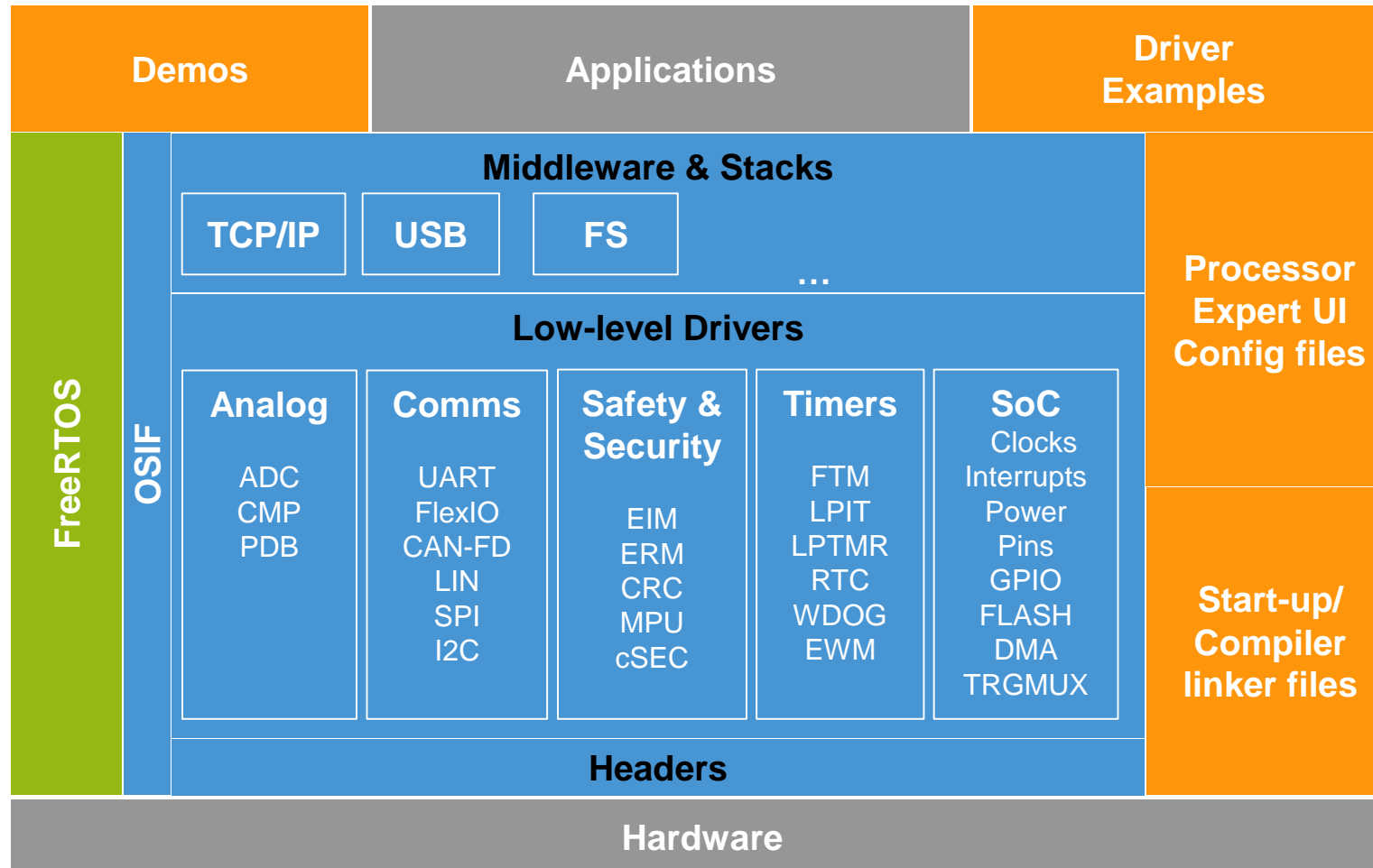
- Graphical configuration
- Documented Source code and examples
- Eclipse or other IDEs
- Middleware + FreeRTOS

Easy production

- Quality level: SPICE/CMMI compliant (Class B), MISRA 2012 compliant
- Multiple toolchains supported
- Consolidates other S32 SW projects
 - Stacks, Flash drivers, FreeRTOS



S32 SDK – Architecture

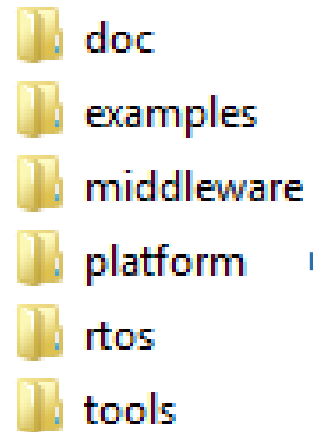


Features

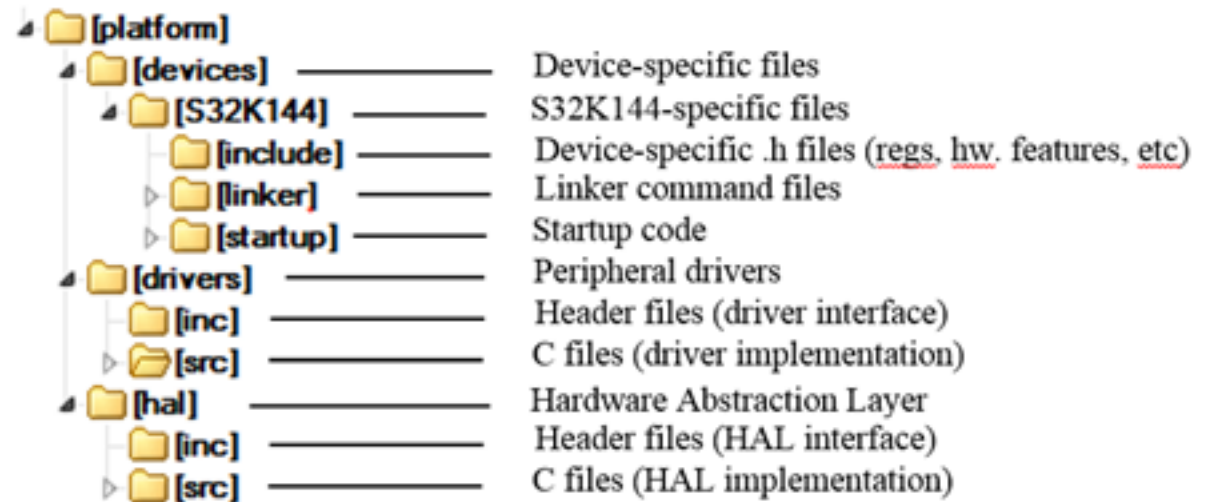
- Integrated **Non-Autosar SW Production-grade** software
- Graphical-based Configuration
- Layered Software Architecture
- Documented Source Code and Examples
- Integrated with S32 Design Studio and other IDEs
- FreeRTOS integration
- Multiple toolchains supported
- Several examples and demos

S32 SDK – File Structure

S32 SDK

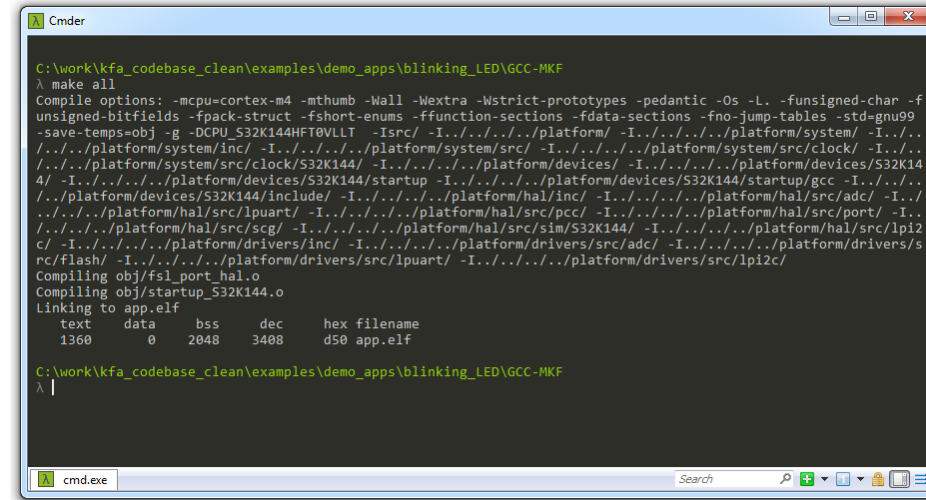


Platform folder



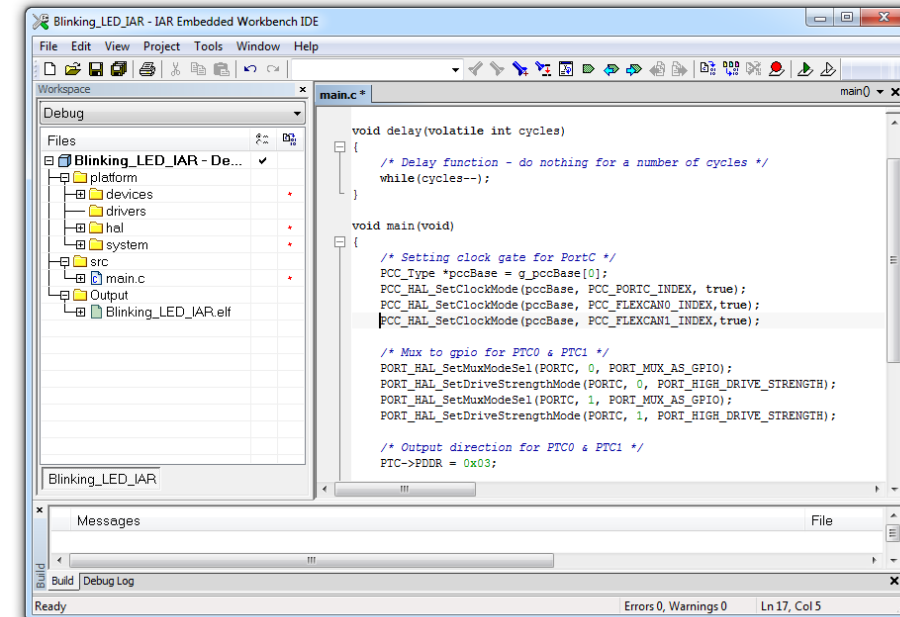
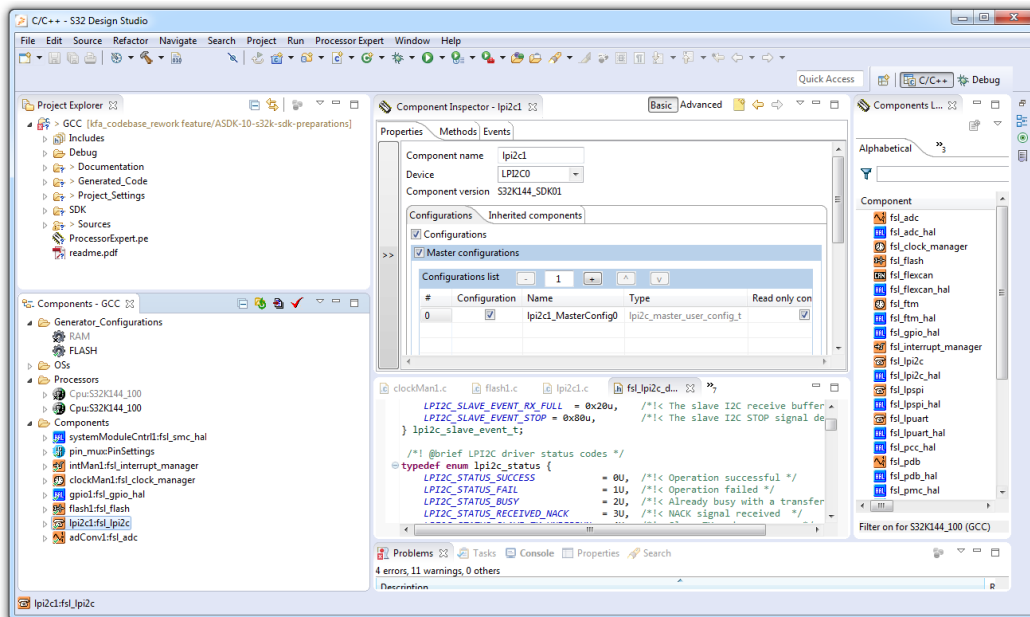
S32 SDK – Usage

- Stand alone
 - Makefile examples
 - IAR example
- Design Studio
 - PEx configurator
 - Integrated compiler and debugger



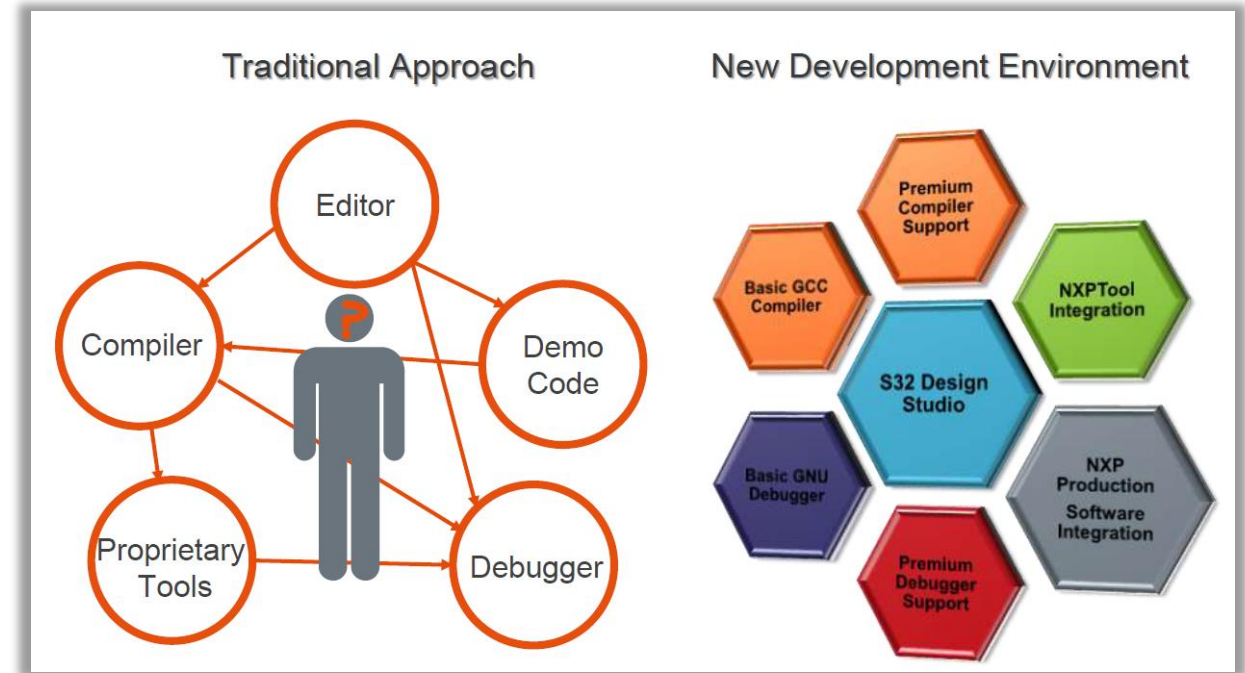
```
C:\work\kfa_codebase_clean\examples\demo_apps\blinking_LED\GCC-MKF
^ make all
Compile options: -mcpu=cortex-m4 -mthumb -Wall -Wextra -Wstrict-prototypes -pedantic -Os -L. -funsigned-char -f
unsigned-bitfields -fpack-struct -fshort-enums -ffunction-sections -fdata-sections -fno-jump-tables -std-gnu99
-save-temps=obj -g -DCPU_S32K144HFT0VLLT -Isrc/ -I../platform/ -I../platform/system/ -I../platform/system/inc/ -I../platform/system/src/ -I../platform/system/src/clock/ -I../platform/system/src/clock/S32K144/ -I../platform/system/src/clock/S32K144/include/ -I../platform/hal/inc/ -I../platform/hal/src/adc/ -I../platform/hal/src/lpuart/ -I../platform/hal/src/pcc/ -I../platform/hal/src/port/ -I../platform/hal/src/scg/ -I../platform/hal/src/sim/S32K144/ -I../platform/hal/src/lpi2c/ -I../platform/drivers/inc/ -I../platform/drivers/src/adc/ -I../platform/drivers/src/lpuart/ -I../platform/drivers/src/lpi2c/
Compiling obj/startup_S32K144.o
Linking to app.elf
text    data    bss     dec      hex filename
1360     0      2048    3408     d50 app.elf

C:\work\kfa_codebase_clean\examples\demo_apps\blinking_LED\GCC-MKF
^
```

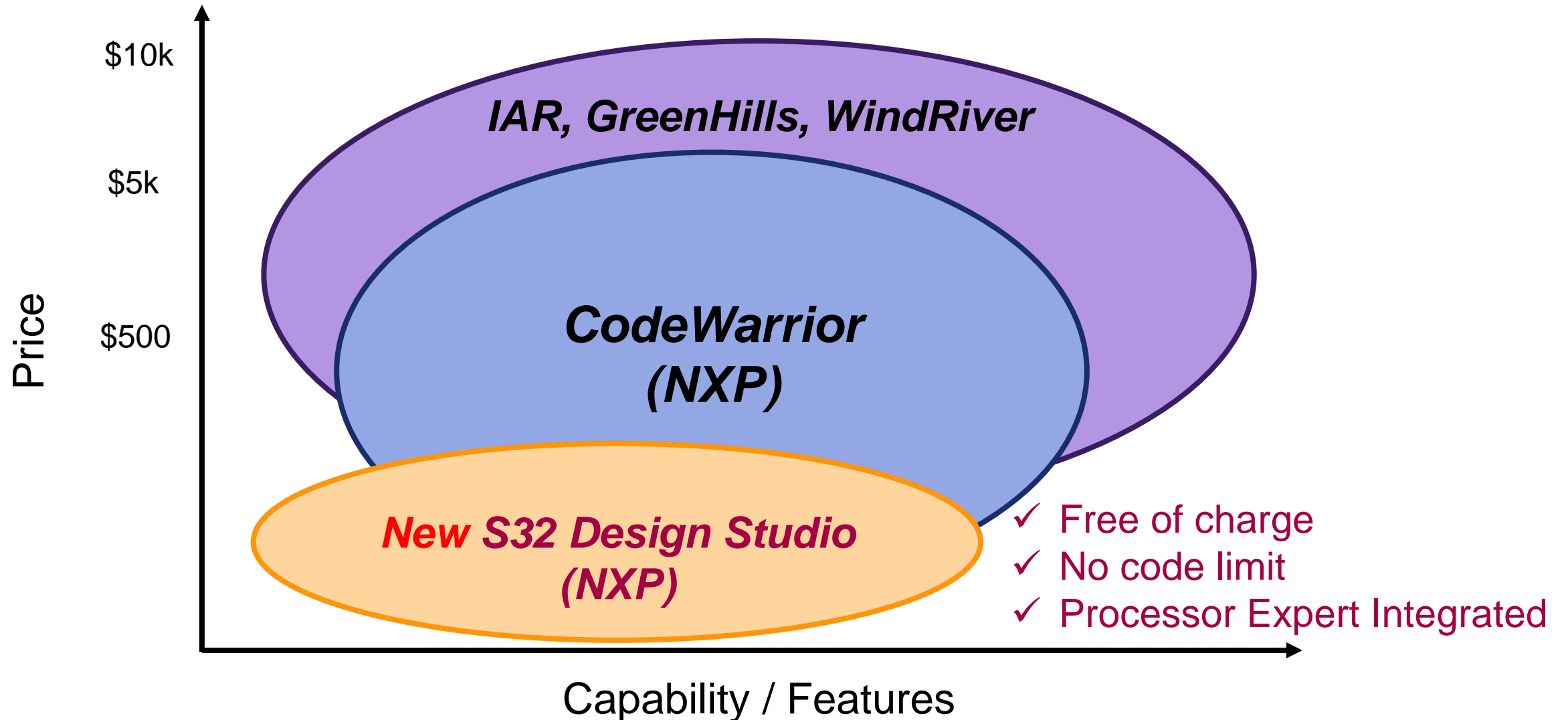


NXP S32 Design Studio IDE www.nxp.com/S32DS

- Free of charge
- Unlimited code size
- Eclipse based environment
- GNU compiler & debugger integrated
- S32 SDK integrated (graphical configuration)
- Processor Expert integrated (automatic code generator)
- Can use with 3rd party compilers & debuggers (IAR) via Connection Utility
- Supports S32K and Power Architecture (MPC) products
- Not a replacement for NXP's CodeWarrior IDE
- Not intended to compete with premium 3rd party IDEs



NXP & 3rd Party IDEs – Performance/Price Map



S32 Design Studio – graphical configuration environment

The screenshot displays the S32 Design Studio interface for configuring the pin_mux component. The main window is titled "Component Inspector - pin_mux" and shows the "Routing" tab. The "View Mode" is set to "Collapsed", and the "Options" section includes "Show Only Configurable Signals". The "Signals" table lists the pin routing for the CAN0, CAN1, and CAN2 peripherals, showing that no pins are routed for these signals.

Signals	Pin Selection	Direction	Selected Pin/Signal Name
CAN0			
Receiver Input	No pin routed	Input	
Transmitter Output	No pin routed	Output	
CAN1			
Receiver Input	No pin routed	Input	
Transmitter Output	No pin routed	Output	
CAN2			
Receiver Input	No pin routed	Input	
Transmitter Output	No pin routed	Output	

The bottom console shows the build output for the S32K144 SDK Lab Clocks project, indicating that the build finished successfully at 16:01:09, taking 8s.789ms.

```
CDT Build Console [S32K144_SDK_Lab_Clocks]
Finished building: ../Generated_Code/clockMan1.c
Finished building: ../Generated_Code/pin_mux.c
Building target: S32K144_SDK_Lab_Clocks.elf
Executing target #16 S32K144_SDK_Lab_Clocks.elf
Invoking: Standard S32DS C Linker
arm-none-eabi-gcc -o "S32K144_SDK_Lab_Clocks.elf" "@S32K144_SDK_Lab_Clocks.args"
Finished building target: S32K144_SDK_Lab_Clocks.elf

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

S32 Design Studio – graphical configuration environment

Pins configuration

The screenshot displays the S32 Design Studio interface for configuring the `pin_mux` component. The **Component Inspector** is open, showing the **Routing** tab. The **View Mode** is set to **Collapsed**, and the **Options** include **Show Only Configurable Signals**. The **Generate Report** button is visible, with a sub-button for **HTML Report**.

The **Signals** table lists the configuration for the `pin_mux` component, showing the selected pin and signal name for each peripheral type.

Signals	Selected Pin/Signal Name
CAN0	
Receiver Input	No pin routed
Transmitter Output	No pin routed
CAN1	
Receiver Input	No pin routed
Transmitter Output	No pin routed
CAN2	
Receiver Input	No pin routed
Transmitter Output	No pin routed

The **Console** shows the build output for the `pin_mux` component, indicating that the build was successful and the target was built.

```
CDT Build Console [S32K144_SDK_Lab_Clocks]
Finished building: ../Generated_Code/clockMan1.c

Finished building: ../Generated_Code/pin_mux.c

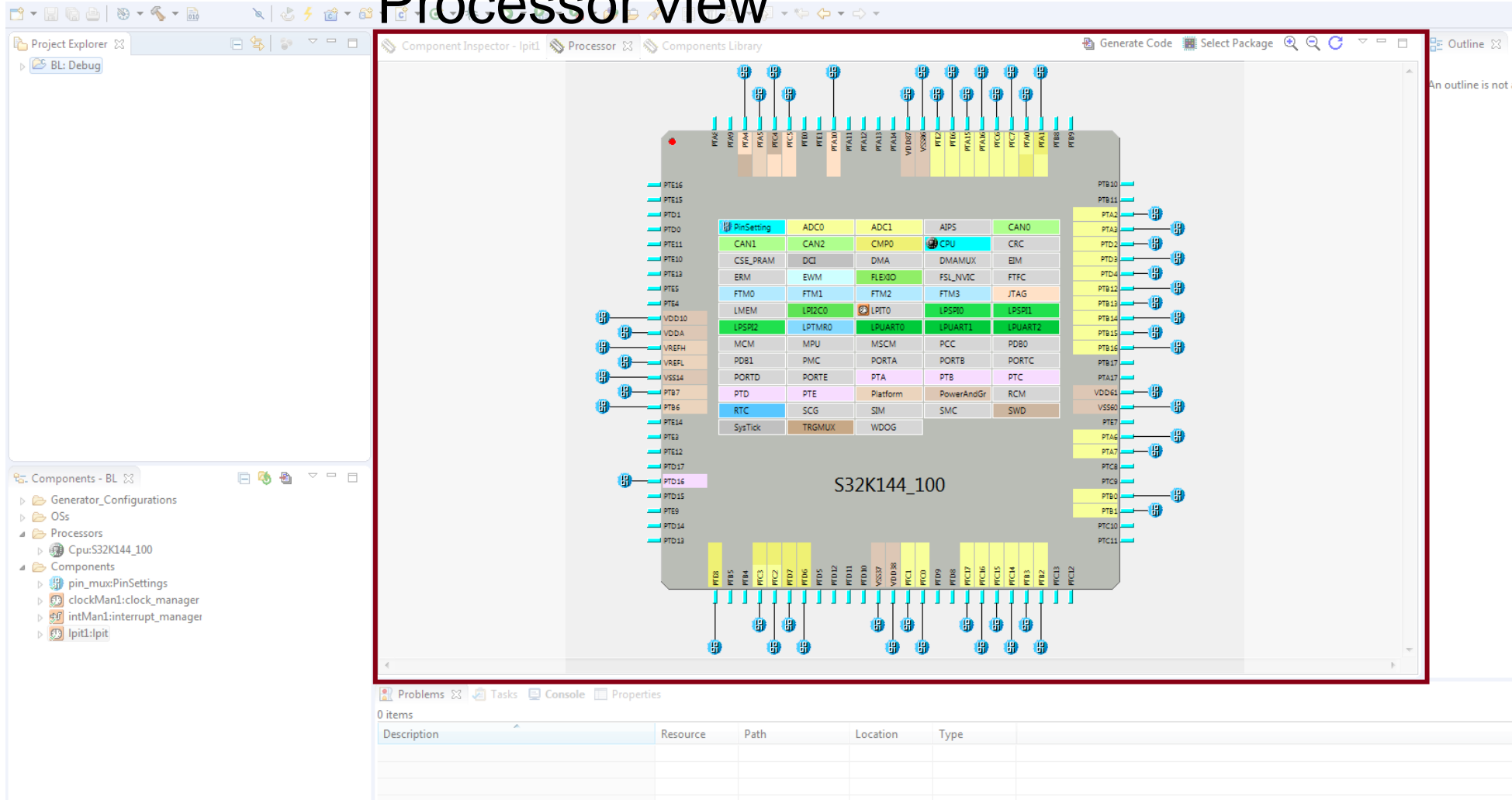
Building target: S32K144_SDK_Lab_Clocks.elf
Executing target #16 S32K144_SDK_Lab_Clocks.elf
Invoking: Standard S32DS C Linker
arm-none-eabi-gcc -o "S32K144_SDK_Lab_Clocks.elf" "@S32K144_SDK_Lab_Clocks.args"
Finished building target: S32K144_SDK_Lab_Clocks.elf

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

An **Updates Available** notification is displayed in the bottom right corner, indicating that updates are available for the software.

S32 Design Studio – graphical configuration environment

Processor view



S32 Design Studio – graphical configuration environment

Components library

The screenshot displays the S32 Design Studio interface. The main window is titled "C/C++ - S32 Design Studio for ARM". The "Components Library" window is open, showing a list of components for S32K144. The components are organized into a table with columns: Component, Component Repository, and Description. The components are filtered for S32K144_100 (S32K144_SDK_Lab_Clocks). The console shows the build output for S32K144_SDK_Lab_Clocks and pin_mux.c.

Component	Component Repository	Description
fsl_adc	S32K144_SDK01	S32 SDK Peripheral Driver for Analog-to-Digital Converter (ADC)
fsl_adc_hal	S32K144_SDK01	S32 SDK HAL for Analog-to-Digital Converter (ADC HAL)
fsl_clock_manager	S32K144_SDK01	S32 SDK Peripheral Driver for Clock Manager (clock_manager)
fsl_cmp	S32K144_SDK01	S32 SDK Peripheral Driver for Comparator (cmp)
fsl_cmp_hal	S32K144_SDK01	S32 SDK HAL for Comparator (cmp)
fsl_crc	S32K144_SDK01	S32 SDK Peripheral Driver for Cyclic Redundancy Check (CRC)
fsl_crc_hal	S32K144_SDK01	S32 SDK HAL for Cyclic Redundancy Check (CRC HAL)
fsl_dmamux_hal	S32K144_SDK01	S32 SDK HAL for Direct Memory Access Multiplexer (dmamux)
fsl_edma	S32K144_SDK01	S32 SDK Peripheral Driver for Enhanced Direct Memory Access controller ...
fsl_edma_hal	S32K144_SDK01	S32 SDK HAL for Enhanced Direct Memory Access controller(edma)
fsl_flash	S32K144_SDK01	S32 SDK Peripheral Driver for Flash Memory (FLASH)
fsl_flexcan	S32K144_SDK01	S32 SDK Peripheral Driver for Flexible Controller Area Network (FlexCAN)
fsl_flexcan_hal	S32K144_SDK01	S32 SDK HAL for Flexible Controller Area Network (FlexCAN HAL)
fsl_flexio_hal	S32K144_SDK01	S32 SDK HAL for Flexible I/O (flexio)
fsl_flexio_i2c	S32K144_SDK01	S32 SDK Peripheral Driver for Inter-Integrated Circuit over Flexible I/O (FL...
fsl_flexio_spi	S32K144_SDK01	S32 SDK Peripheral Driver for Serial Peripheral Interface over Flexible I/O (...)
fsl_ftm	S32K144_SDK01	S32 SDK Peripheral Driver for FlexTimer Module (FTM)
fsl_ftm_hal	S32K144_SDK01	S32 SDK HAL for FlexTimer Module (FTM HAL)
fsl_gpio_hal	S32K144_SDK01	S32 SDK HAL for General-Purpose Input/Output (GPIO HAL)
fsl_interrupt_manager	S32K144_SDK01	S32 SDK Peripheral Driver for Interrupt Manager (Interrupt_manager)
fsl_lin	S32K144_SDK01	S32 SDK Peripheral Driver for Local Interconnect Network (LIN)
fsl_lpi2c	S32K144_SDK01	S32 SDK Peripheral Driver for Low Power Inter-Integrated Circuit (LPI2C)
fsl_lpi2c_hal	S32K144_SDK01	S32 SDK HAL for Low Power Inter-Integrated Circuit (LPI2C HAL)
fsl_lpit	S32K144_SDK01	S32 SDK Peripheral Driver for Low Power Interrupt Timer (LPIT)
fsl_lpit_hal	S32K144_SDK01	S32 SDK HAL for Low Power Interrupt Timer Module (LPIT HAL)
fsl_lpspi	S32K144_SDK01	S32 SDK Peripheral Driver for Low Power Serial Peripheral Interface (LPSPI)
fsl_lpspi_hal	S32K144_SDK01	S32 SDK HAL for Low Power Serial Peripheral Interface (LPSPI HAL)

Filter on for S32K144_100 (S32K144_SDK_Lab_Clocks)

CDT Build Console [S32K144_SDK_Lab_Clocks]
Finished building: ../Generated_Code/clockMan1.c
Finished building: ../Generated_Code/pin_mux.c
Building target: S32K144_SDK_Lab_Clocks.elf
Executing target #16 S32K144_SDK_Lab_Clocks.elf
Invoking: Standard S32DS C Linker
arm-none-eabi-gcc -o "S32K144_SDK_Lab_Clocks.elf" "@S32K144_SDK_Lab_Clocks.args"
Finished building target: S32K144_SDK_Lab_Clocks.elf
16:01:09 Build Finished (took 8s.789ms)

Updates Available
Updates are available for your software.
Click to review and install updates.
Set up [Reminder options](#)

S32 Design Studio – graphical configuration environment

Project components

The screenshot displays the S32 Design Studio interface. On the left, the Project Explorer shows the project structure for 'S32K144_SDK_Lab_Clocks'. A red box highlights the 'Components' folder, which contains 'pin_muxPinSettings' and 'clockMan1fsl_clock_manager'. The main workspace shows the 'Component Inspector - pin_mux' with the 'Routing' tab selected. It displays a table of signals and their pin routing for peripheral type LPI2C. The table has columns for 'Signals', 'The group contains selection of the pin routing for peripheral type LPI2C', and 'Selected Pin/Signal Name'. The signals listed are CAN0, CAN1, and CAN2, each with 'Receiver Input' and 'Transmitter Output' options. The status for all is 'No pin routed'. The bottom console shows build output for 'S32K144_SDK_Lab_Clocks', indicating a successful build of 'S32K144_SDK_Lab_Clocks.elf'.

Signals	The group contains selection of the pin routing for peripheral type LPI2C	Selected Pin/Signal Name
CAN0		
Receiver Input	No pin routed	Input
Transmitter Output	No pin routed	Output
CAN1		
Receiver Input	No pin routed	Input
Transmitter Output	No pin routed	Output
CAN2		
Receiver Input	No pin routed	Input
Transmitter Output	No pin routed	Output

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

S32 Design Studio – graphical configuration environment

Component inspector

The screenshot shows the S32 Design Studio interface with the Component Inspector window open for the `lpuart1` component. The window is divided into several sections:

- Properties:** Displays the component name (`lpuart1`), device (`LPUART0`), and component version (`S32K144_SDK01`).
- Configurations:** Shows the state structure name (`lpuart1_State`) and a table of configurations.
- Configurations list:** A table with columns for configuration number, name, type, read-only status, baud rate, parity mode, stop bits, and bits per character.
- Project Explorer:** Shows the project structure on the left, including `S32K144_SDK_Lab_Clocks`, `S32K144_SDK_Lab_GPIOs`, and `S32K144_SDK_Lab_Interrupts`.
- Console:** Displays the Processor Expert service log at the bottom.

The configuration table shows one configuration, `lpuart1_InitConfig0`, which is selected and has a checkmark in the "Read only configuration" column.

#	Configuration	Name	Type	Read only configuration	Baud rate	Parity mode	Stop bits	Bits per char
0	<input checked="" type="checkbox"/>	lpuart1_InitConfig0	lpuart_user_config_t	<input checked="" type="checkbox"/>	600	Disabled	1	8

Processor Expert log:

```
Sep 28, 2016 7:31:15 PM Starting Processor Expert service
System directory = C:\Freescale\S32_ARM_v1.2_clean\eclipse\ProcessorExpert
Internal cache directory = C:\ProgramData\Processor Expert\PECache\ef2a80e7
Processor Expert license file = not used (no license file)
Sep 28, 2016 7:31:16 PM Successfully started Processor Expert service
GroupItem.loadItem: Null item - ignored symbol:null
```

An "Updates Available" notification is visible in the bottom right corner, indicating that updates are available for the software.

S32 Design Studio – graphical configuration environment

Code generation

The screenshot displays the S32 Design Studio interface for the project 'S32K144_SDK_Lab_Clocks'. The 'Project Explorer' on the left shows the project structure, with 'lpuart1.c' and 'lpuart1.h' highlighted in the 'Generated_Code' folder. The 'Components' pane at the bottom left shows the 'lpuart1.fsl_lpuart' component selected. The main editor window shows the generated C code for 'lpuart1.c', which includes a configuration structure 'lpuart_user_config_t' and a driver state structure 'lpuart_state_t'. The 'Console' at the bottom shows the build output, indicating that the target 'S32K144_SDK_Lab_Clocks.elf' was successfully built and executed. A small 'Updates Available' dialog box is visible in the bottom right corner.

```
/* This component module is generated by Processor Expert. Do not modify it. */
/*
 * @file lpuart1.c
 * @version 01.00
 * @brief
 */
/*
 * @addtogroup lpuart1_module lpuart1 module documentation
 * @{
 */
/* MODULE lpuart1. */

#include "lpuart1.h"

/* lpuart1 configuration structure */
const lpuart_user_config_t lpuart1_InitConfig0 = {
    .baudRate = 6000U,
    .parityMode = LPUART_PARITY_DISABLED,
    .stopBitCount = LPUART_ONE_STOP_BIT,
    .bitCountPerChar = LPUART_8_BITS_PER_CHAR,
};

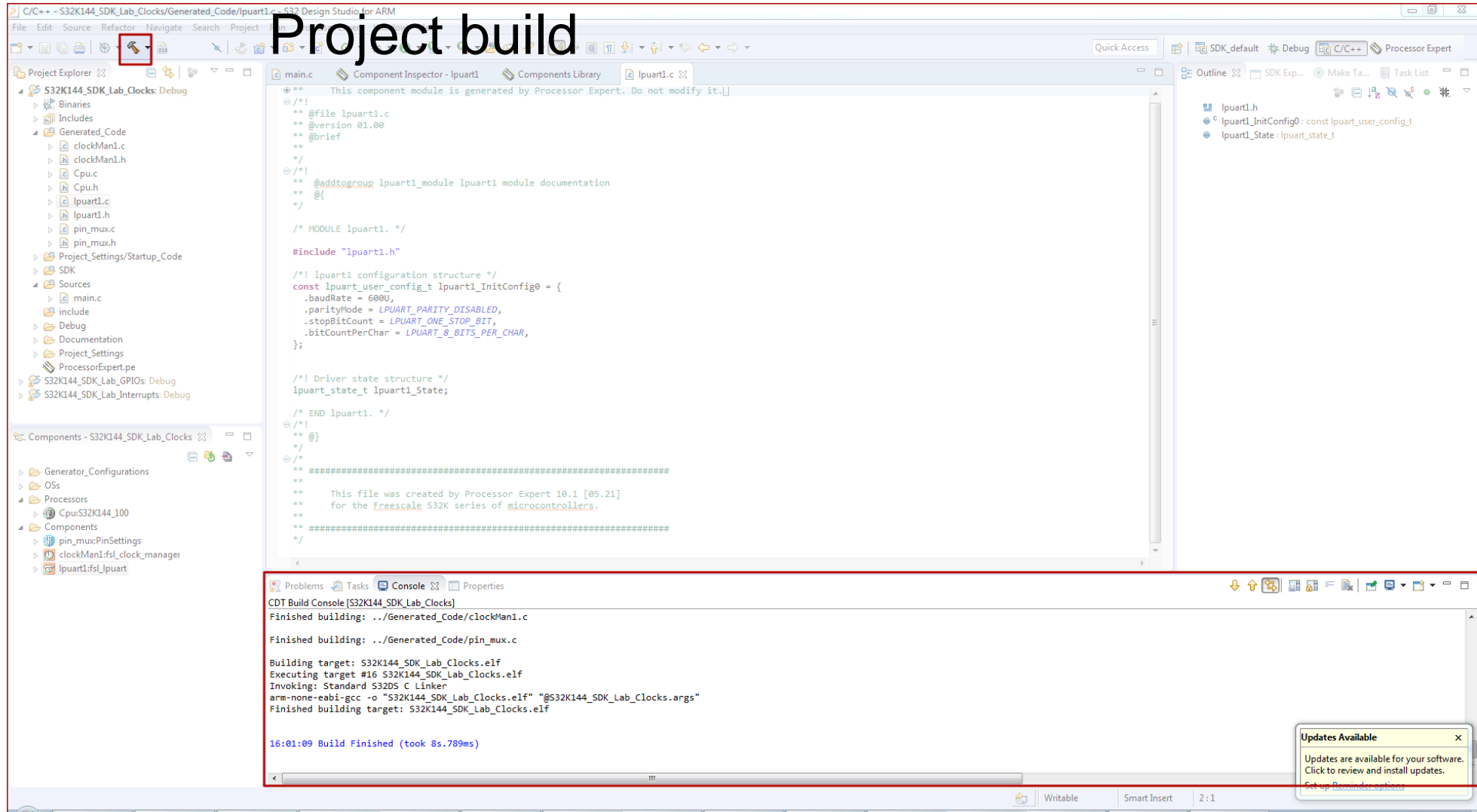
/* Driver state structure */
lpuart_state_t lpuart1_State;

/* END lpuart1. */
/*
 * @}
 */
/*
 * #####
 * This file was created by Processor Expert 10.1 [05.21]
 * for the Freescale S32K series of microcontrollers.
 * #####
 */
```

CDT Build Console [S32K144_SDK_Lab_Clocks]
Finished building: ../Generated_Code/clockMan1.c
Finished building: ../Generated_Code/pin_mux.c
Building target: S32K144_SDK_Lab_Clocks.elf
Executing target #16 S32K144_SDK_Lab_Clocks.elf
Invoking: Standard S32DS C Linker
arm-none-eabi-gcc -o "S32K144_SDK_Lab_Clocks.elf" "@S32K144_SDK_Lab_Clocks.args"
Finished building target: S32K144_SDK_Lab_Clocks.elf
16:01:09 Build Finished (took 8s.789ms)

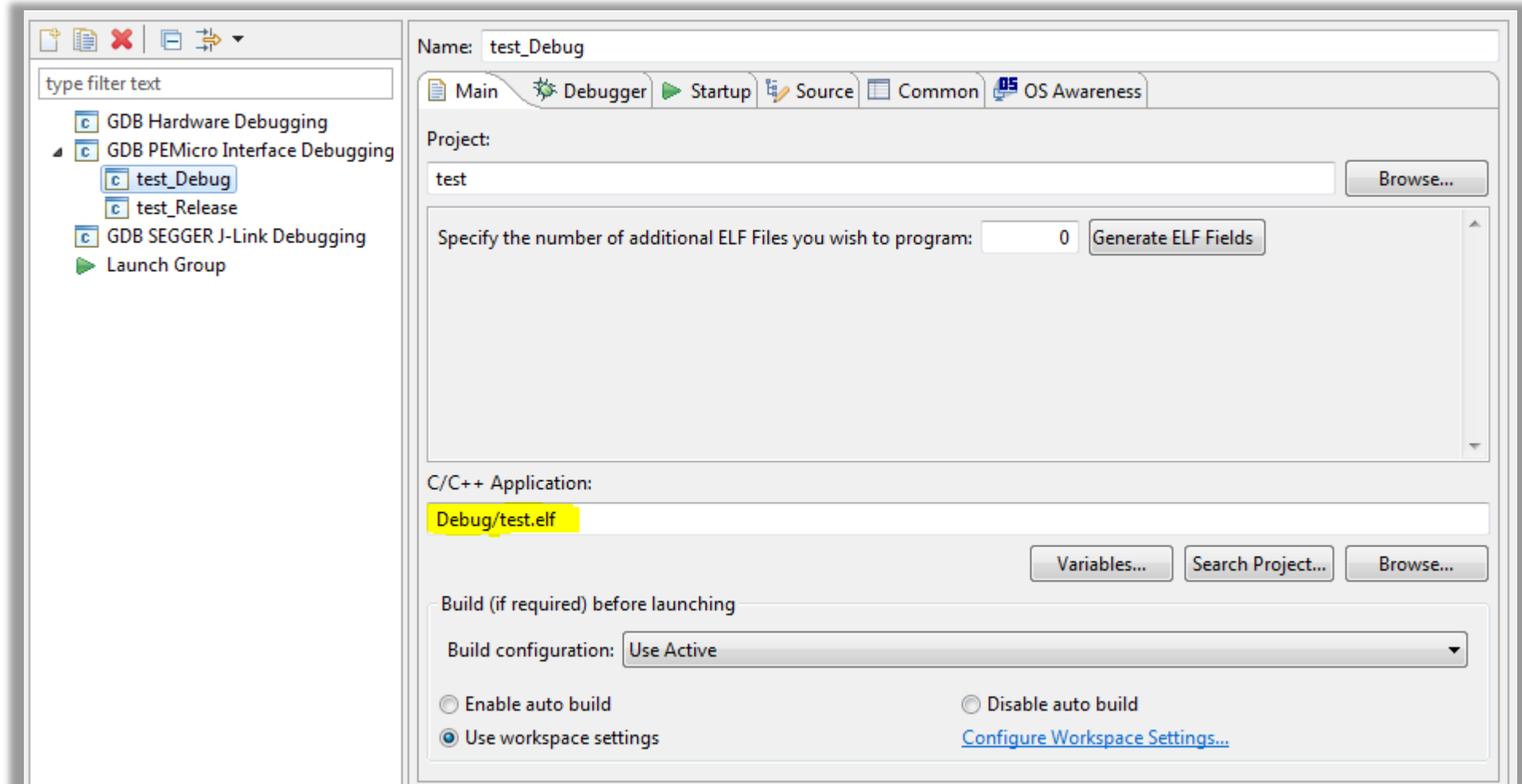
Updates Available
Updates are available for your software.
Click to review and install updates.
Set up [Reminder options](#)

S32 Design Studio – graphical configuration environment



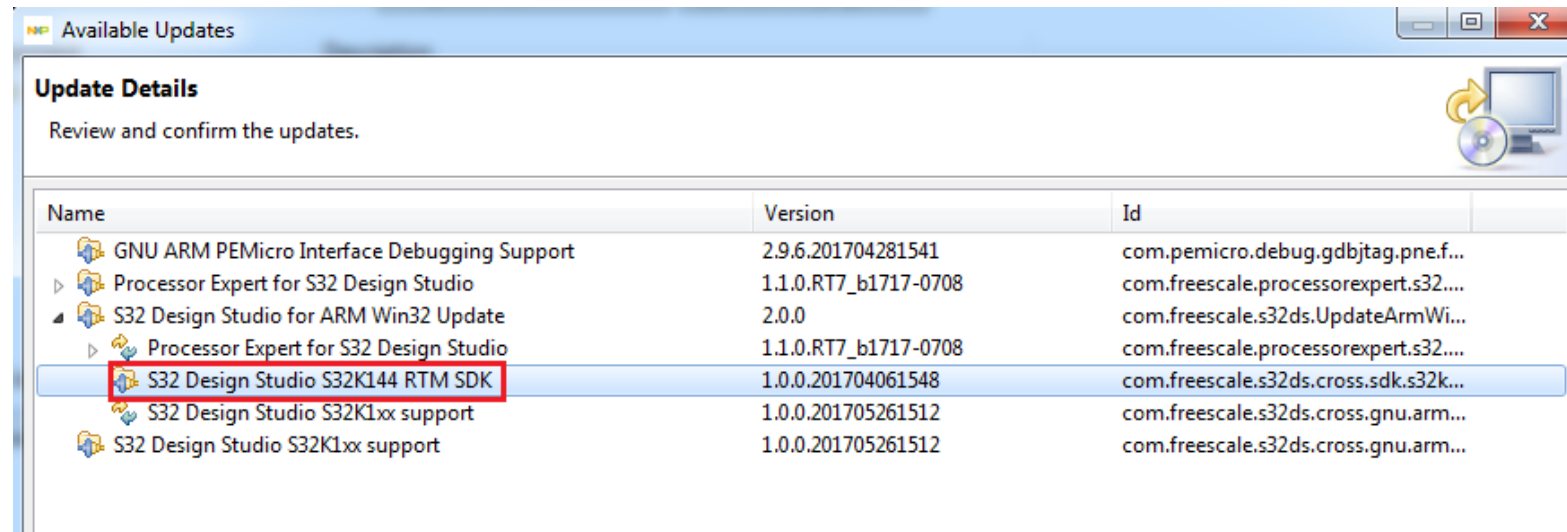
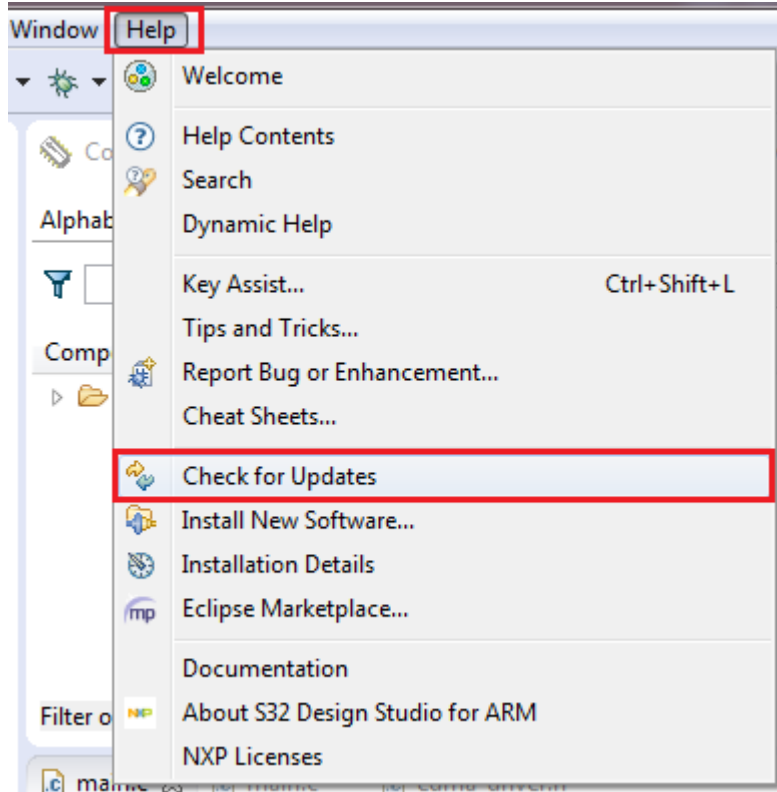
S32 Design Studio – deploying the application

Target debug



S32 Design Studio – SDK Update

Newer SDK versions are provided as update sites for DS





02.

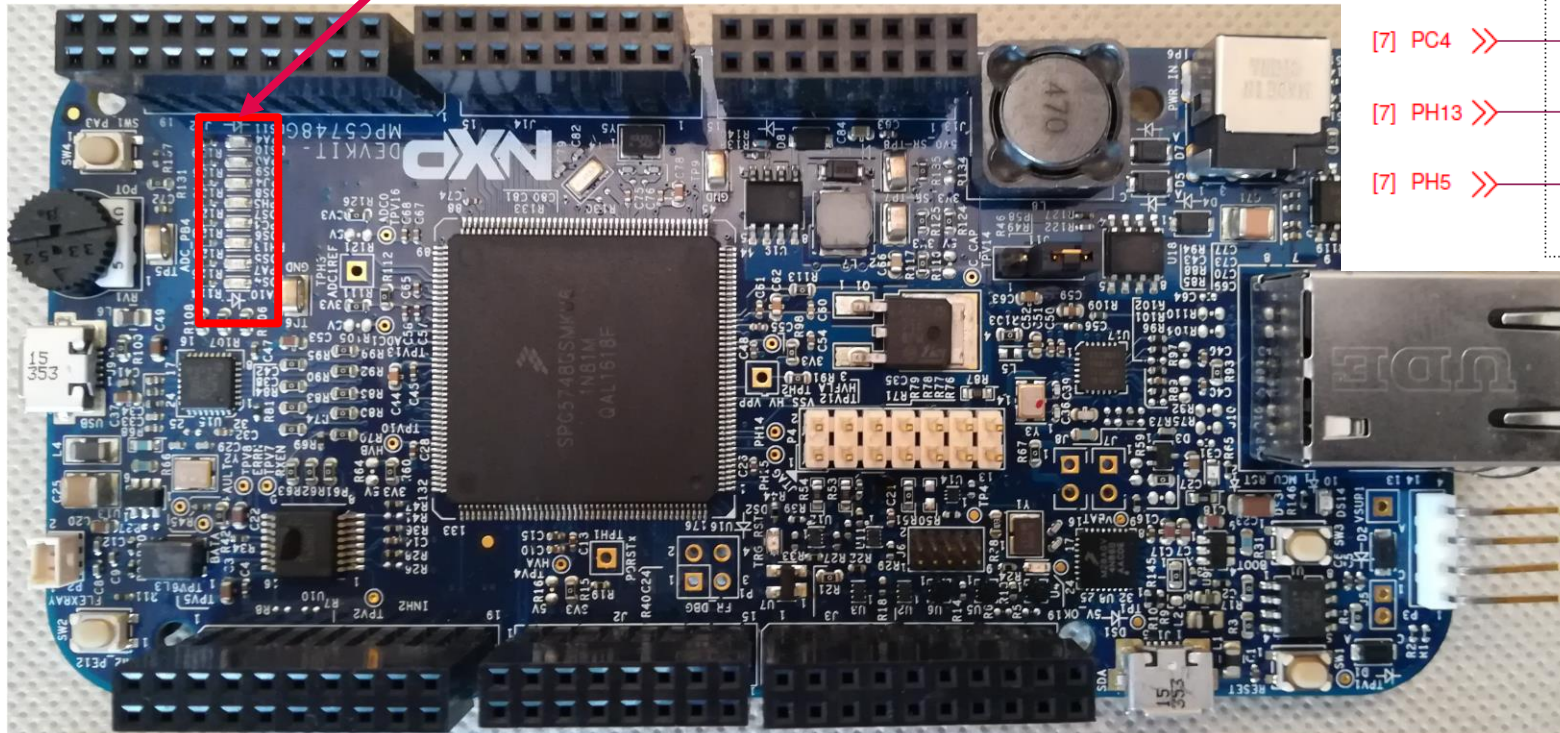
Hands-on – Blinking LED

Hands-on – Blinking LED: Objective

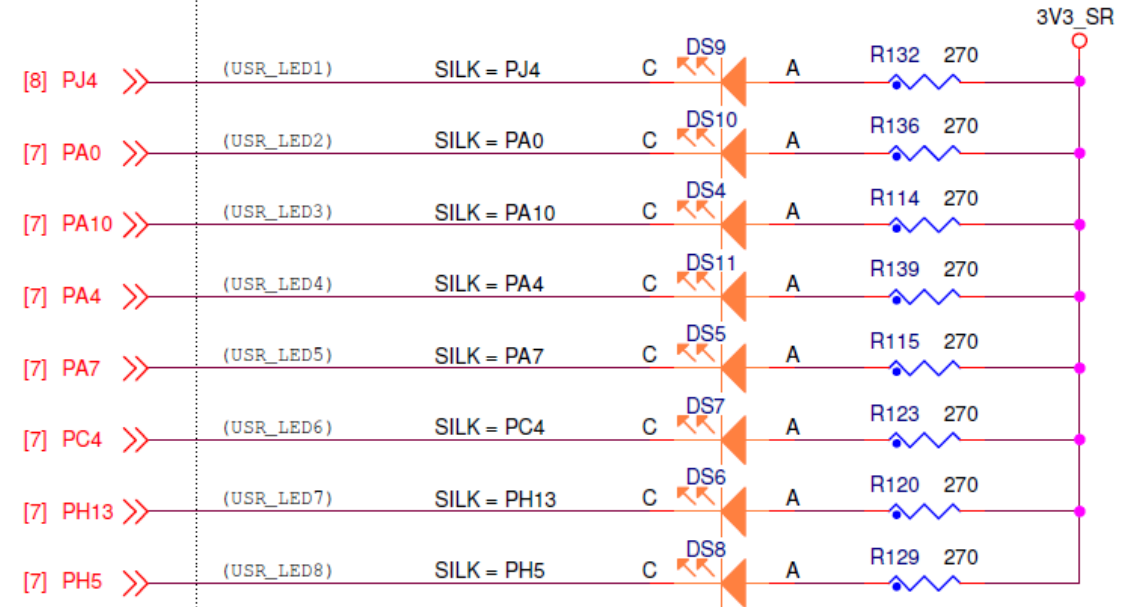
- In this lab you will learn:
 - About the GPIOs structure in MPC574x
 - How interrupts works on MPC574x
 - How to create a new SDK project with S32DS.
 - How to set a pin as output/input with SDK
 - How the use the PIT peripheral
 - Set up an interrupt event using SDK API
 - Blink an LED every 0.5 sec using the PIT interrupt

Hands-on – Blinking LED: Resources

- Resources to be used:
 - on-board user LEDs (hardwired to GPIOs)

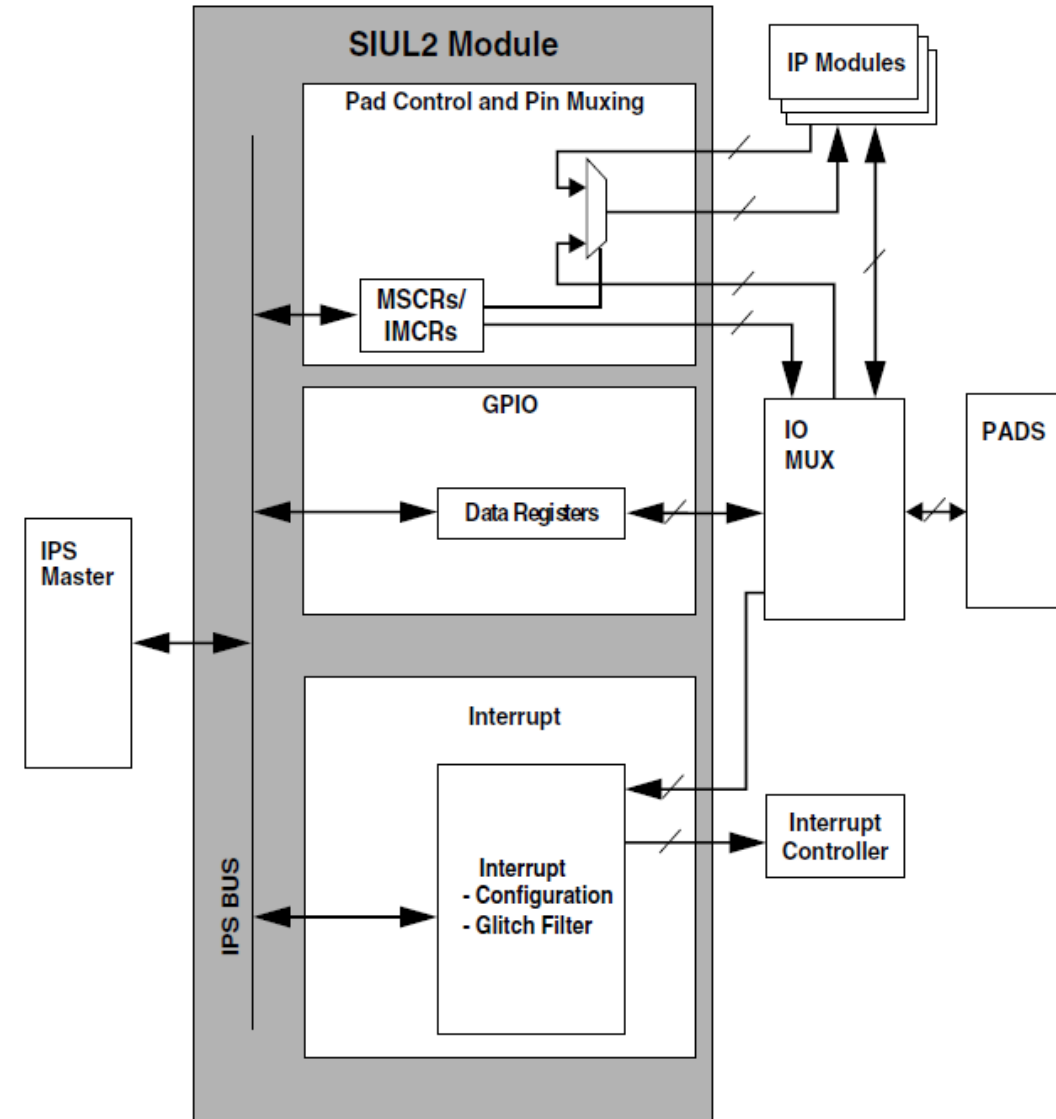


User LED's (Active Low)



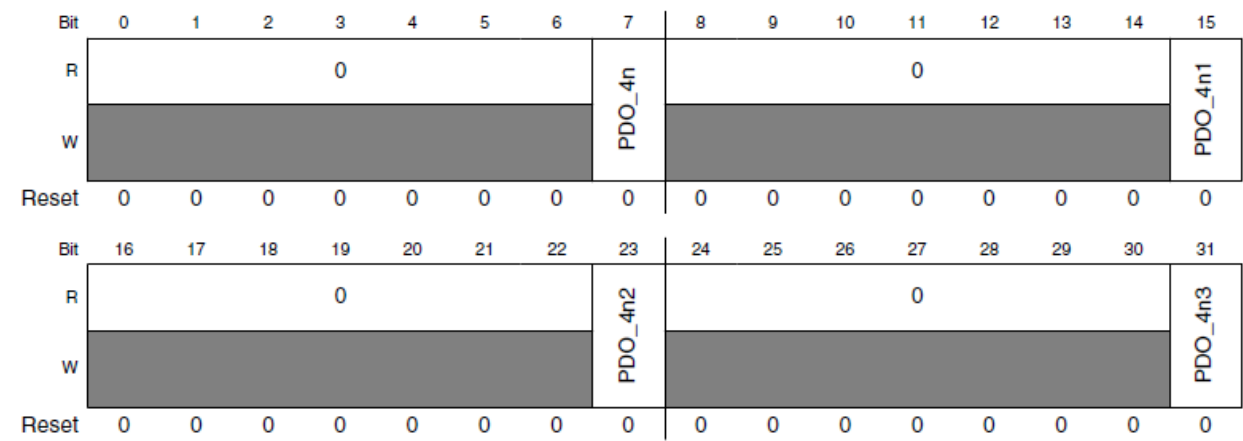
Hands-on – Blinking LED: Theory

- System Integration Unit (SIUL2):
 - Up to 264 GPIOs in MPC5748G, grouped in 16-bit ports
 - 146 (176 LQFP)
 - 196 (256 BGA)
 - 264 (324 BGA)
 - External interrupt request support
 - 0 to 31 external interrupt sources mapped to 0 to 3 interrupt vectors
 - Input/output signals multiplexing
 - Electrical parameters configuration
 - Drive strength
 - Open drain/source output enable
 - Slew rate control
 - Hysteresis control
 - Internal pull control and pull selection
 - Safe mode behavior configuration

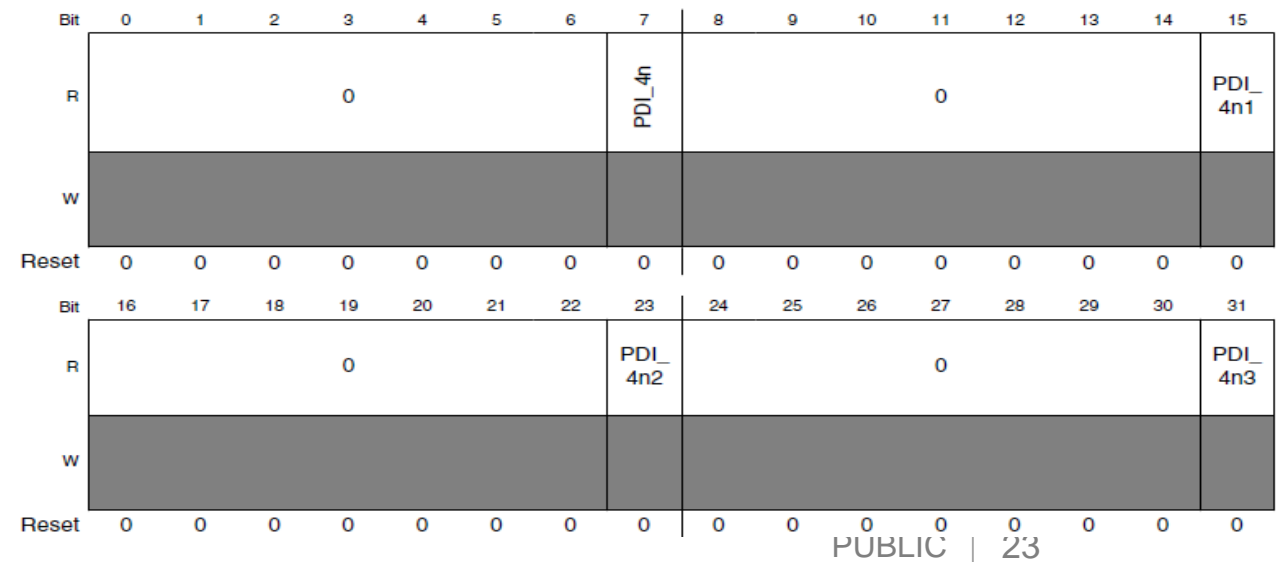


Hands-on – Blinking LED: Theory

- System Integration Unit (SIUL2) – GPIO data read/write:
 - GPIO Pad Data Output register:

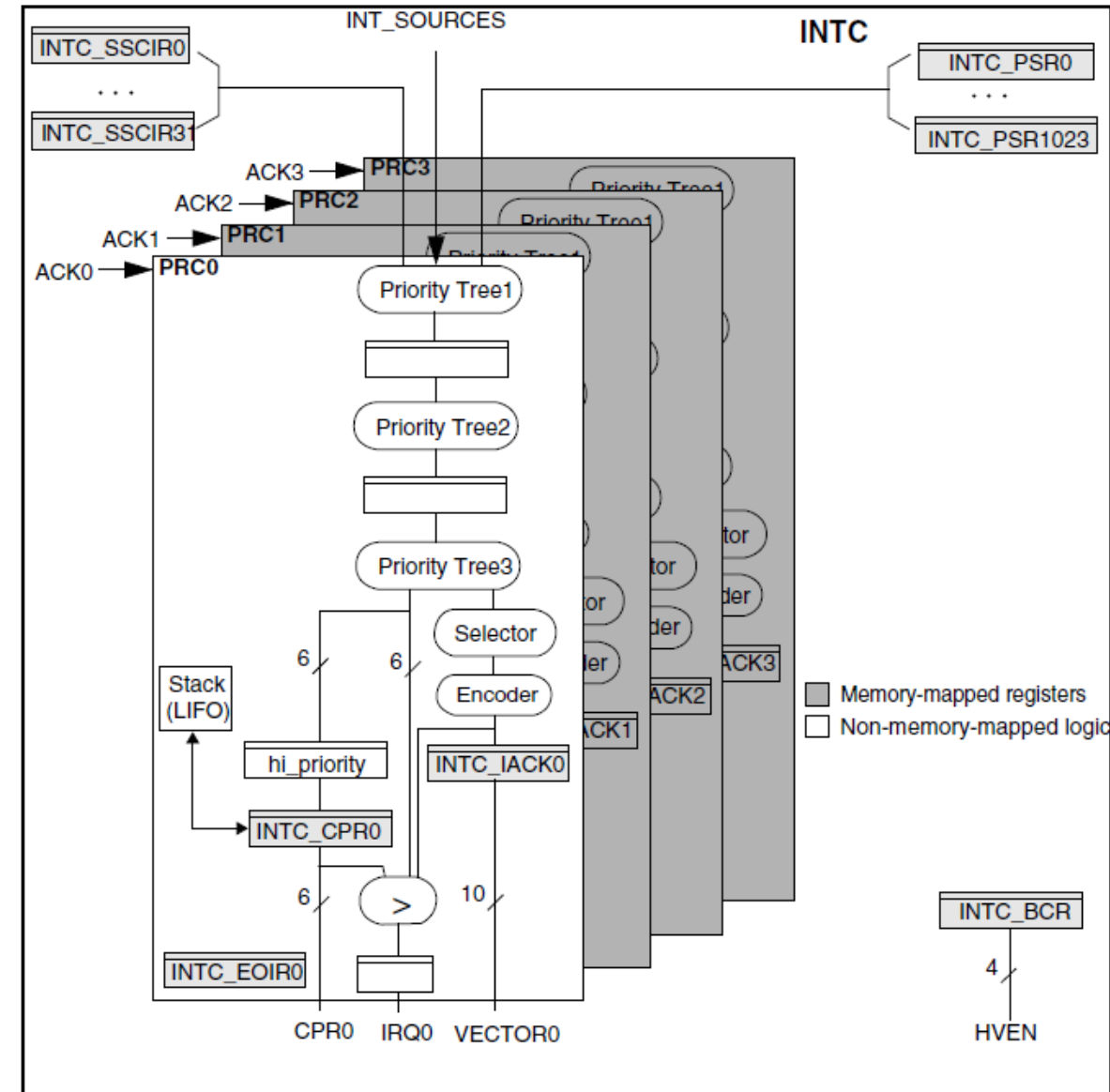


– GPIO Pad Data Input Register:



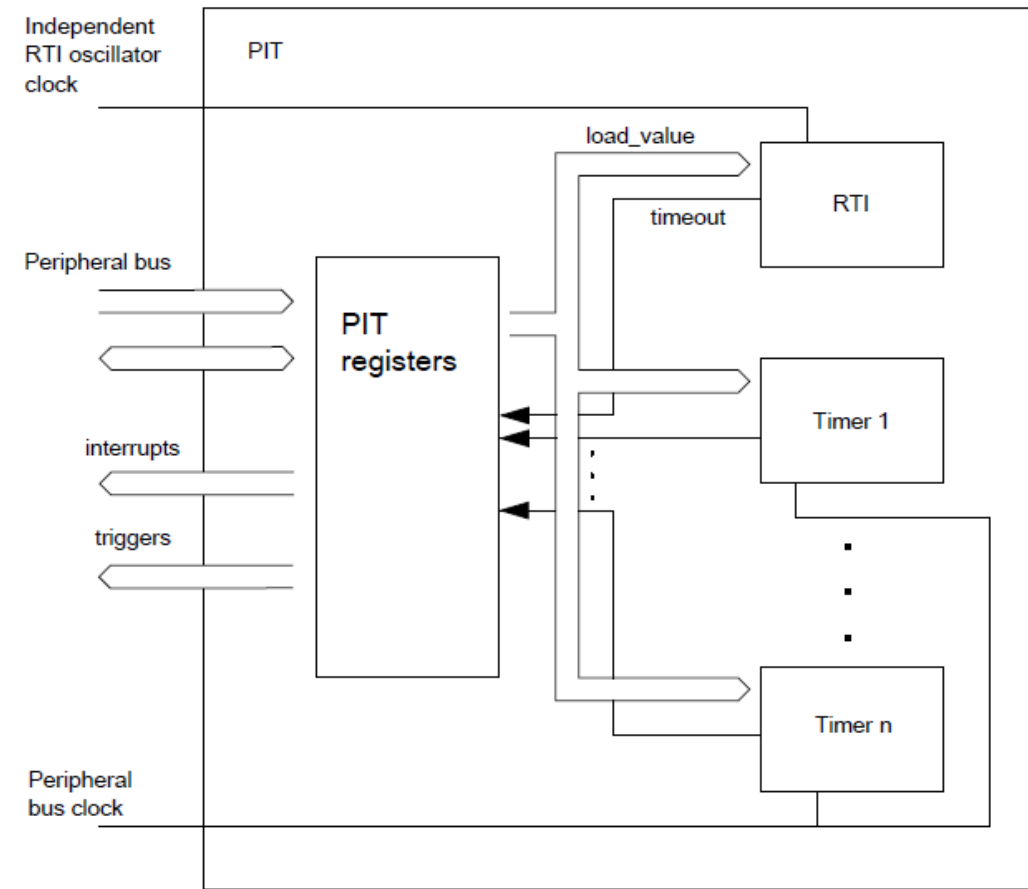
Hands-on – Blinking LED: Theory

- Interrupt Controller (INTC):
 - 754 interrupts available on MPC5748G
 - Each interrupt source is software-steerable to any core
 - 24 software-settable interrupt request sources
 - 16 priority levels (preemption)
 - SW trigger
 - HW/SW vector modes



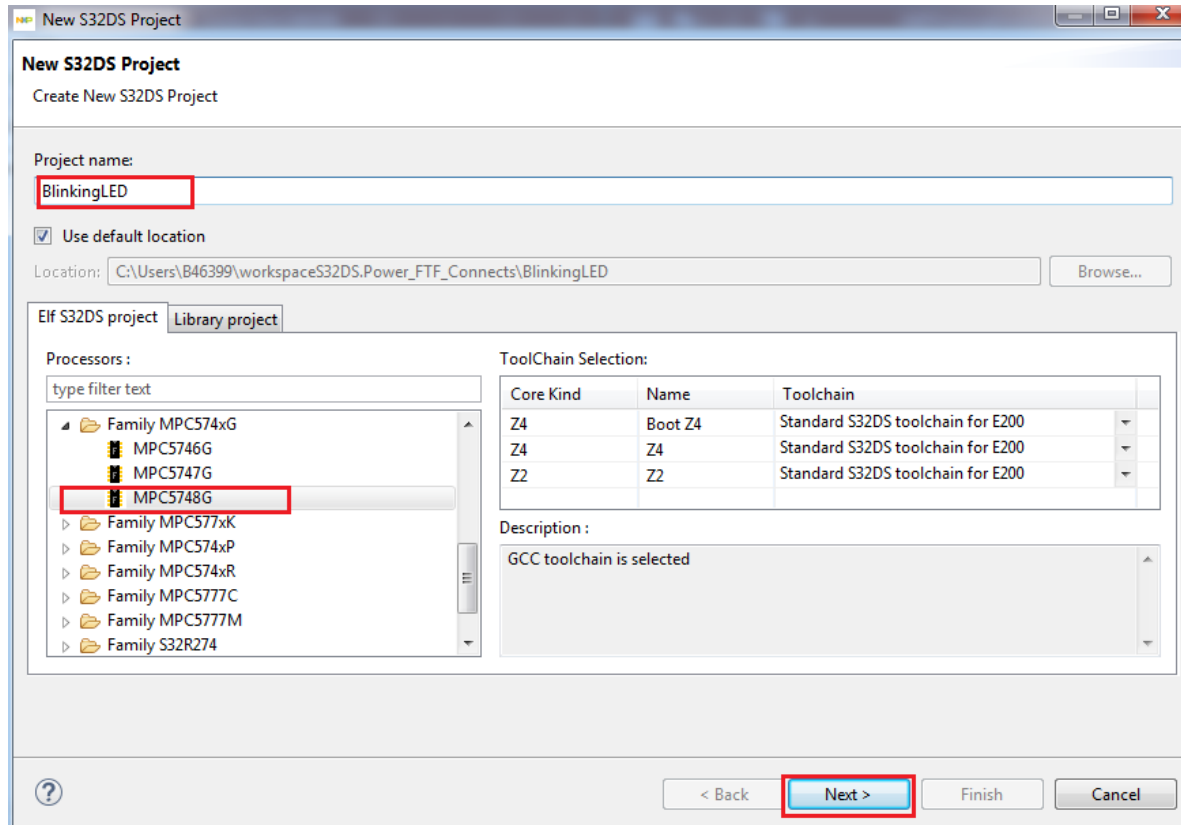
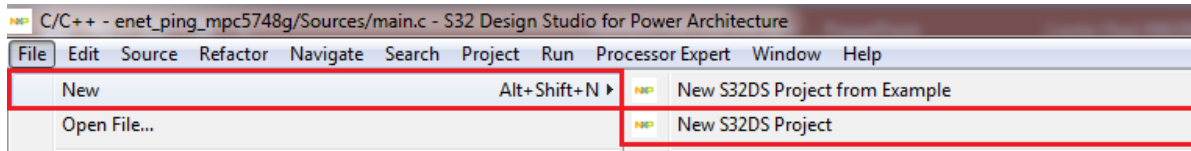
Hands-on – Blinking LED: Theory

- Periodic Interrupt Timer:
 - One RTI (Real-Time Interrupt) timer to wakeup the CPU in stop mode
 - 16 timer channels
 - Ability of timers to generate trigger pulses
 - Ability of timers to generate interrupts
 - Maskable interrupts
 - Option to raise RTI interrupt, even when the bus clock is switched off
 - Independent timeout periods for each timer
 - Chained mode



Hands-on – Blinking LED: New Project

- Create a new S32DS project for MPC5748G:



Hands-on – Blinking LED: New Project

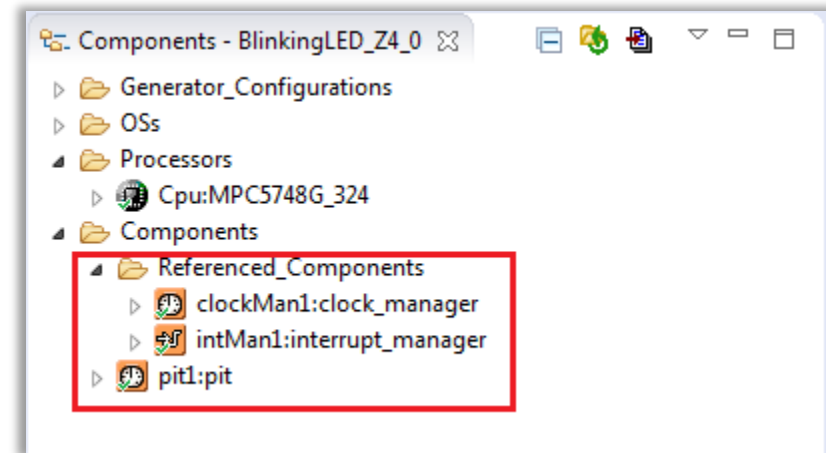
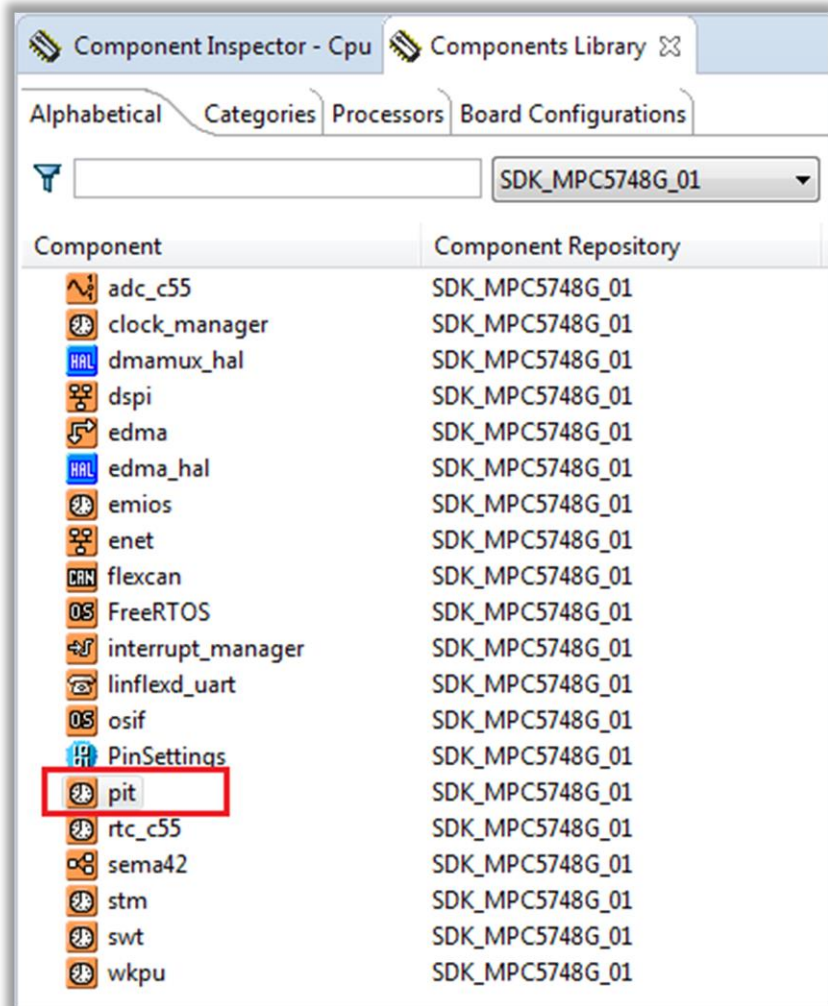
- Check only the 'Boot z4' core (application will be running on the boot core)
- Select SDK support for the new project

New S32DS Project for MPC5748G
Select required cores and parameters for them.

	BlinkingLED_Z4_0	BlinkingLED_Z4_1	BlinkingLED_Z2
Project Name	BlinkingLED_Z4_0	BlinkingLED_Z4_1	BlinkingLED_Z2
Core	<input checked="" type="checkbox"/> Boot Z4	<input type="checkbox"/> Z4	<input type="checkbox"/> Z2
FLASH Start Address	0x1000000	0x11d0000	0x11d0000
FLASH Size, KB	1856	0	0
Unused FLASH, KB	3776		
RAM Start Address	0x40000000	0x40040000	0x40040000
RAM Size, KB	256	0	0
Unused RAM, KB	512		
Language	C	C	C
SDKs	MPC5748G_SDK		
Library	EWL	EWL	EWL
Debugger	PE Micro GDB server		

Hands-on – Blinking LED: Configuration

- From 'Components Library' view, double-click 'pit' component to add it the project
- 'clock_manager' & 'interrupt_manager' dependencies will be automatically added



Hands-on – Blinking LED: Configuration

- Clicking the 'pit' component opens UI configuration in 'Component Inspector' view
- Default channel 0 with interrupt enabled – change period to 0.5 sec

Component name: pit1
Device: PIT
Component version: S32_SDK_C55

Configurations Shared components

Global configuration

☒ Enable Standard timers The standard timer clock frequency is 40000000 [Hz]!
☐ Enable RTI timer
☐ Timer Stop In Debug Mode

Channel configurations list - 1 + ^ v

#	Configuration	Name	Read only	Channel	Period units	Timer period	Channel chain	Interrupt enable
0	<input checked="" type="checkbox"/>	pit1_ChnConfig0	<input type="checkbox"/>	Channel 0	Microsecond unit	1000000	<input type="checkbox"/>	<input checked="" type="checkbox"/>

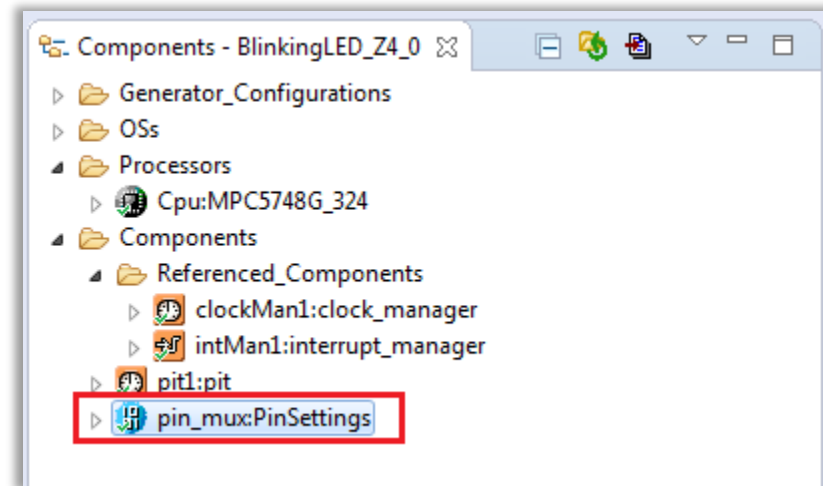
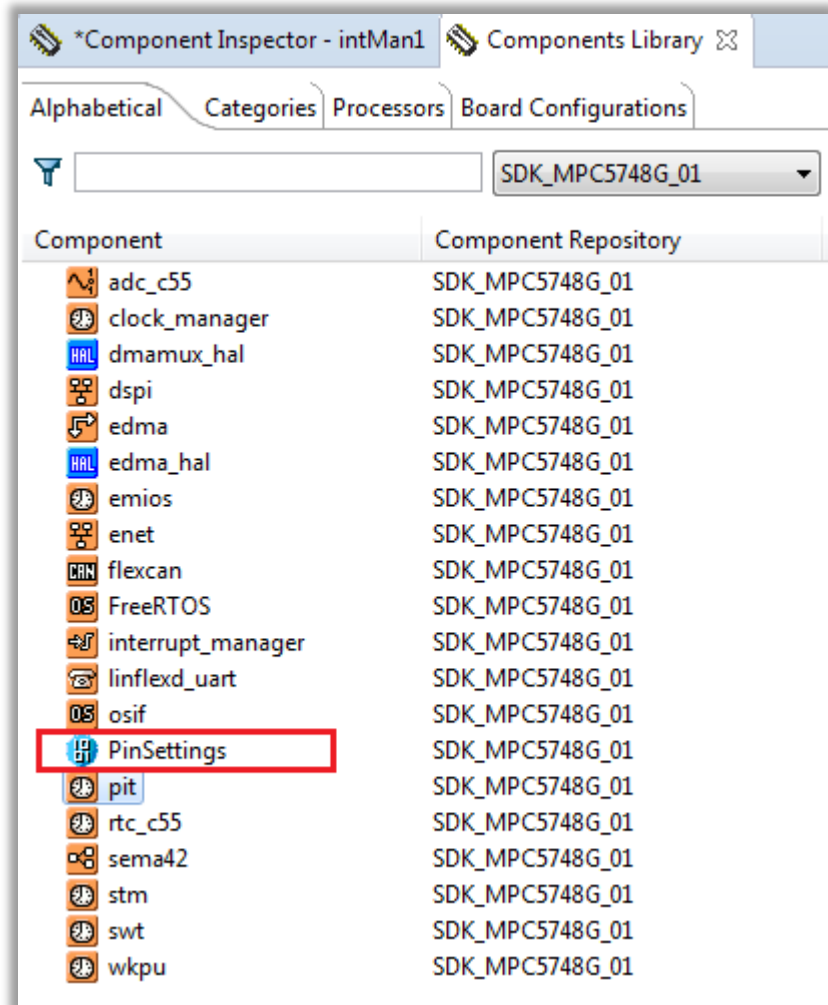
Details for selected row:

☒ Configuration 0

Name: pit1_ChnConfig0
Read only: ☐
Channel: Channel 0
Period units: Microsecond unit
Timer period: 500000
Channel chain: ☐
Interrupt enable: ☒

Hands-on – Blinking LED: Configuration

- From 'Components Library' view, double-click 'PinSettings' component to add it the project



Hands-on – Blinking LED: Configuration

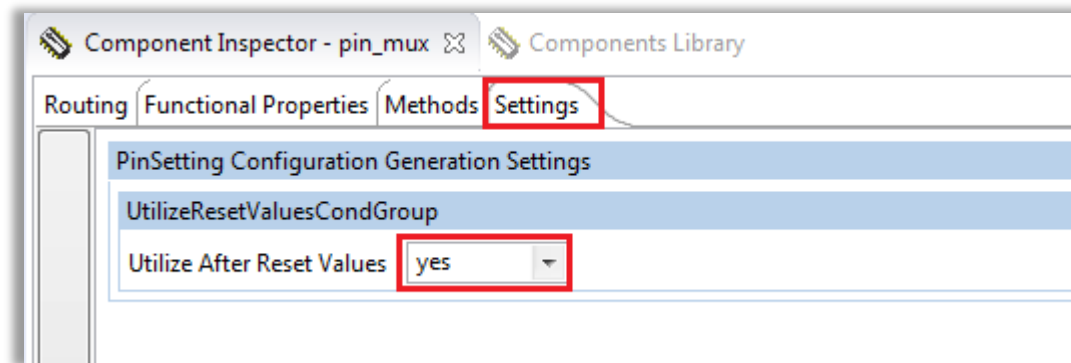
- Open 'pin_mux' component in 'Component Inspector' to configure pin routing
- SIUL2 tab -> GPIO 0 -> select the pin (one option) + direction output
 - PA0 is internally connected to user LED 2 on the board

The screenshot shows the 'Component Inspector - pin_mux' window with the 'Routing' tab selected. The 'SIUL2' component is highlighted in the top navigation bar. The 'Signals' table below shows the configuration for various signals, with 'GPIO 0' selected and highlighted by a red box. The 'Direction' column for 'GPIO 0' is set to 'Output', and the 'Selected Pin/Signal Name' is 'PA[0]'. The 'Options' section shows 'Show Only Configurable Signals' is unchecked. The 'Generate Report' button is visible in the top right corner.

Signals	Pin/Signal Selection	Direction	Selected Pin/Signal Name
GPI 55	PD[7]	Input	PD[7]
GPI 56	PD[8]	Input	PD[8]
GPI 57	PD[9]	Input	PD[9]
GPI 58	PD[10]	Input	PD[10]
GPI 59	PD[11]	Input	PD[11]
GPIO 0	PA[0]	Output	PA[0]
GPIO 1	No pin routed	No pin routed	
GPIO 2	No pin routed	No pin routed	
GPIO 3	No pin routed	No pin routed	

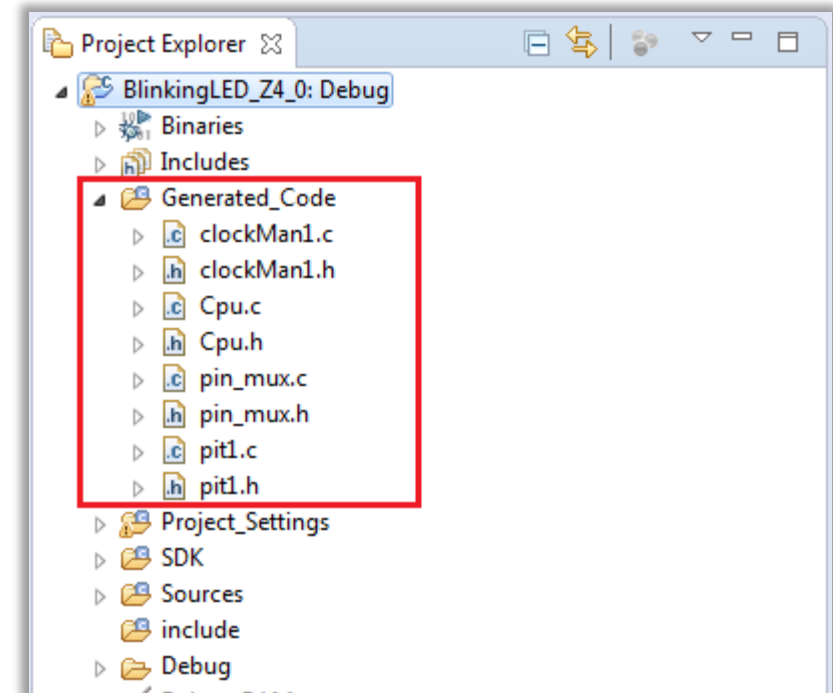
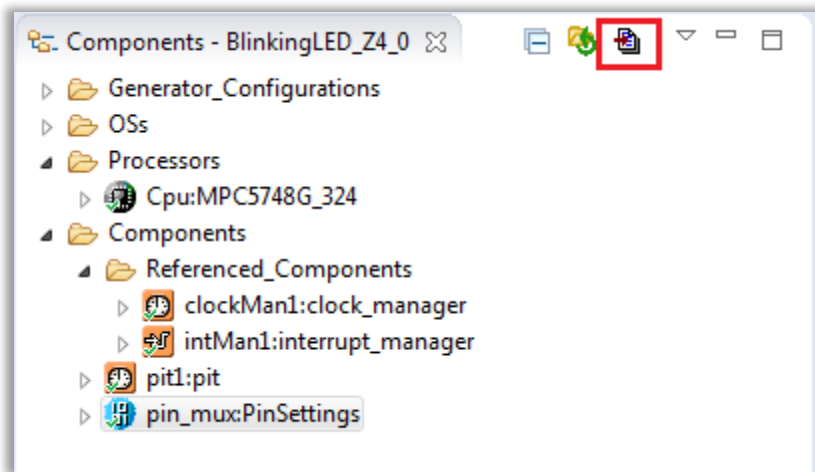
Hands-on – Blinking LED: Configuration

- In 'pin_mux' component → Settings tab – select 'Utilize After Rest Values – yes'
 - this option will enable code generation only for the modified pins (generating one configuration structure for each pin will result in a very large array residing in target memory)



Hands-on – Blinking LED: Configuration

- Once the configuration is done, hit the 'Generate code' button
- Configuration structures are generated in 'Generated_Code' project folder

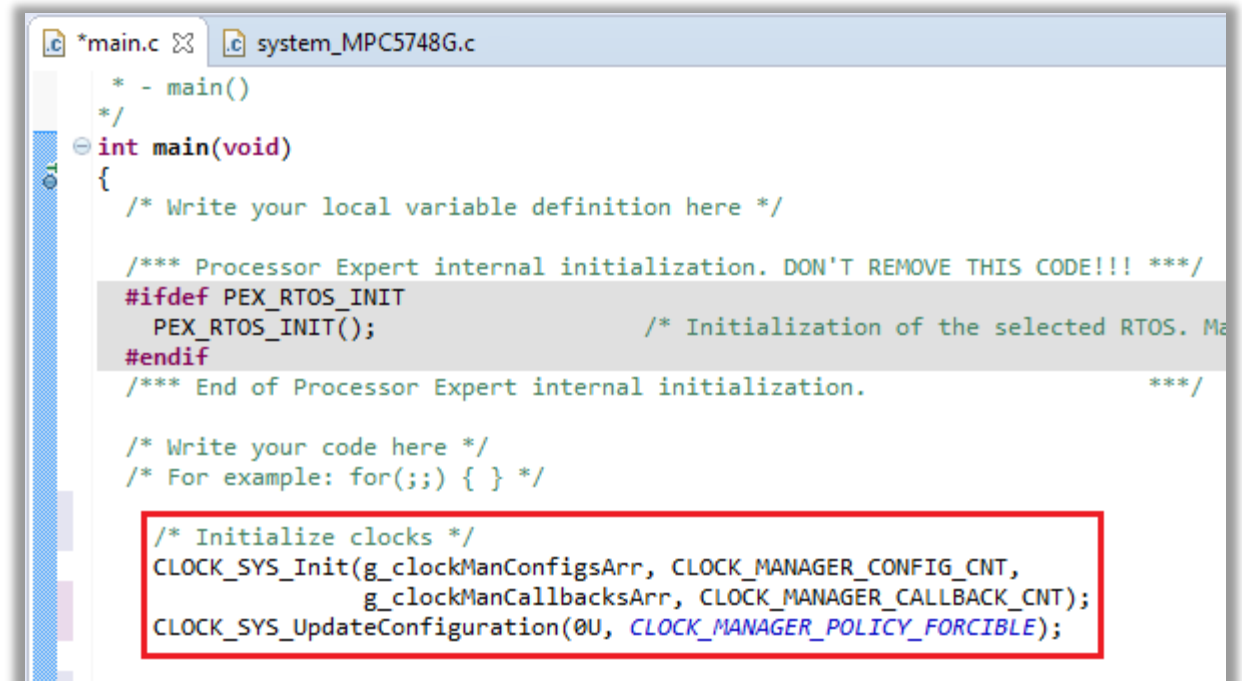
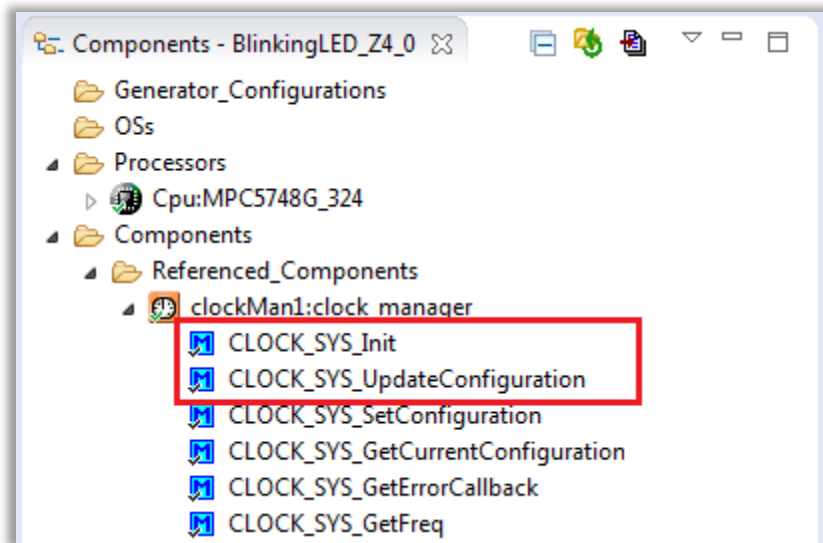


Hands-on – Blinking LED: Application Code

- Open the main.c file in text editor view

1) Initialize clocks

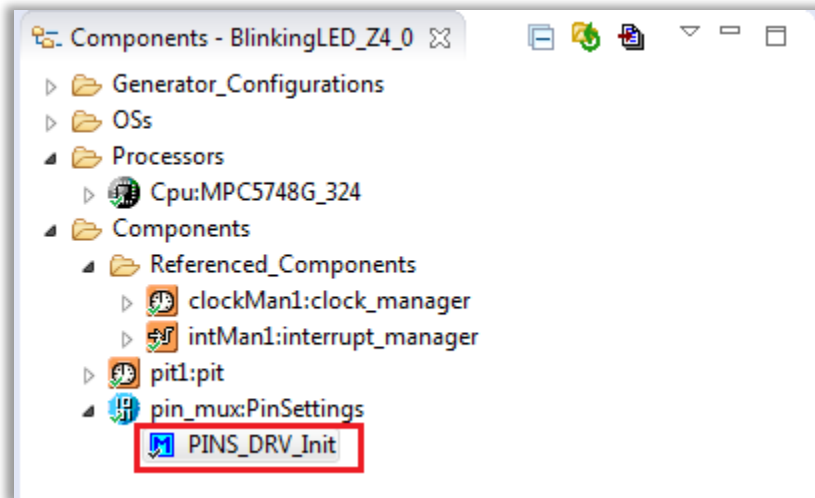
- expand 'clock_manager' component -> drag and drop **CLOCK_SYS_Init** & **CLOCK_SYS_UpdateConfiguration** functions in 'main()'
- fill in the parameters for the second function: **0U, CLOCK_MANAGER_POLICY_FORCIBLE**



Hands-on – Blinking LED: Application Code

2) Initialize pins

- expand 'pin_mux' component -> drag and drop **PINS_DRV_Init** in 'main()'



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

    /** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! */
    #ifdef PEX_RTOS_INIT
        PEX_RTOS_INIT(); /* Initialization of the selected RTOS. M
    #endif
    /** End of Processor Expert internal initialization. */

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

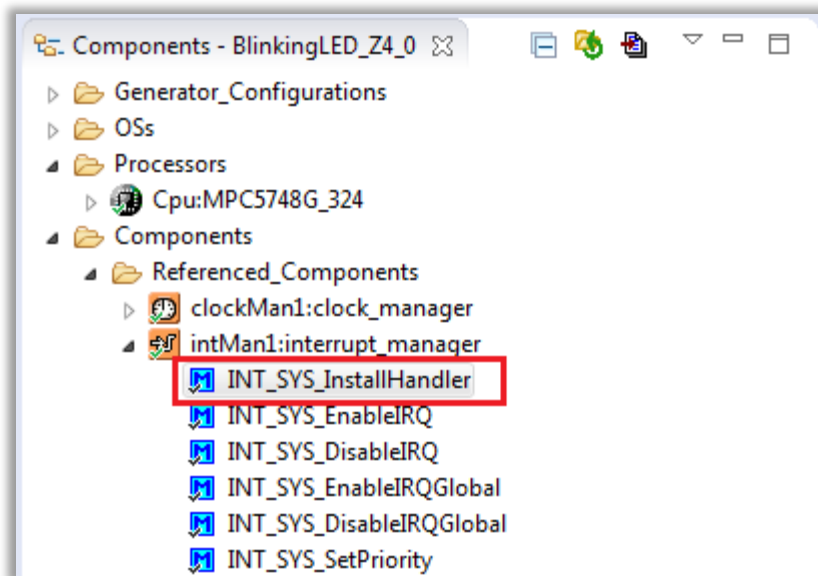
    /* Initialize clocks */
    CLOCK_SYS_Init(g_clockManConfigsArr, CLOCK_MANAGER_CONFIG_CNT,
                  g_clockManCallbacksArr, CLOCK_MANAGER_CALLBACK_CNT);
    CLOCK_SYS_UpdateConfiguration(0U, CLOCK_MANAGER_POLICY_FORCIBLE);

    /* Initialize pins */
    PINS_DRV_Init(NUM_OF_CONFIGURED_PINS, g_pin_mux_InitConfigArr);
}
```

Hands-on – Blinking LED: Application Code

3) Install the PIT channel 0 interrupt handler

- expand 'interrupt_manager' component -> drag and drop ***INT_SYS_InstallHandler*** in 'main()'
- fill in the parameters: ***PIT_Ch0_IRQn, &pitCh0Handler, NULL***



```
/* Initialize clocks */
CLOCK_SYS_Init(g_clockManConfigsArr, CLOCK_MANAGER_CONFIG_CNT,
               g_clockManCallbacksArr, CLOCK_MANAGER_CALLBACK_CNT);
CLOCK_SYS_UpdateConfiguration(0U, CLOCK_MANAGER_POLICY_FORCIBLE);

/* Initialize pins */
PINS_DRV_Init(NUM_OF_CONFIGURED_PINS, g_pin_mux_InitConfigArr);

/* Install PIT ch0 handler */
INT_SYS_InstallHandler(PIT_Ch0_IRQn, &pitCh0Handler, NULL);
```

Hands-on – Blinking LED: Application Code

4) Add the implementation of the handler

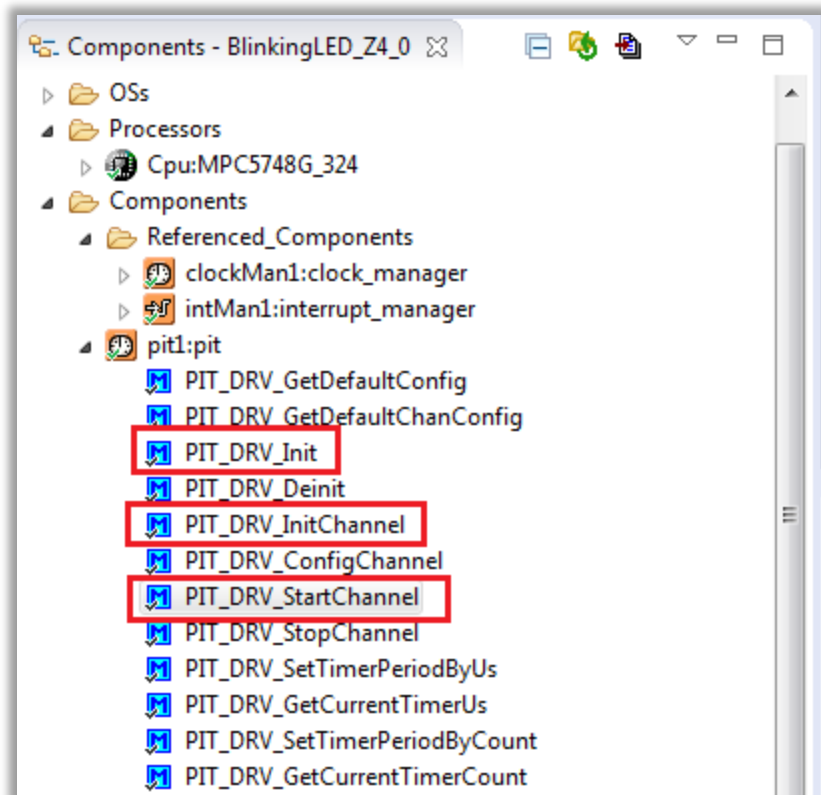
- create a function called ***pitCh0Handler***
- in the function body: clear the interrupt flag + toggle LED

```
/* IRQ handler for PIT ch0 interrupt */
void pitCh0Handler(void)
{
    /* Clear PIT channel 0 interrupt flag */
    PIT_DRV_ClearStatusFlags(INST_PIT1, 0U);
    /* Toggle LED (GPIO 0 connected to user LED 2) */
    SIUL2->GPDO[0] ^= SIUL2_GPDO_PDO_4n_MASK;
}
```

Hands-on – Blinking LED: Application Code

5) Initialize PIT and start the timer

- expand 'pit' component -> drag and drop **PIT_DRV_Init**, **PIT_DRV_InitChannel** & **PIT_DRV_StartChannel** in 'main()'
- for the last function, fill in the second parameter: **0U** - channel number



```
/* Initialize clocks */
CLOCK_SYS_Init(g_clockManConfigsArr, CLOCK_MANAGER_CONFIG_CNT,
               g_clockManCallbacksArr, CLOCK_MANAGER_CALLBACK_CNT);
CLOCK_SYS_UpdateConfiguration(0U, CLOCK_MANAGER_POLICY_FORCIBLE);

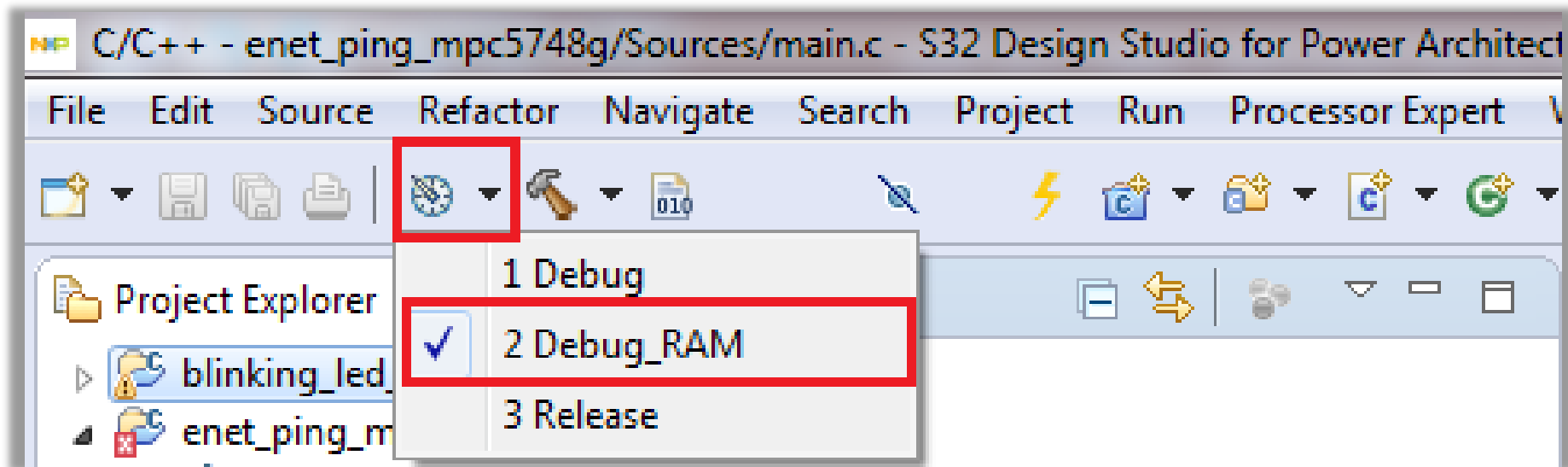
/* Initialize pins */
PINS_DRV_Init(NUM_OF_CONFIGURED_PINS, g_pin_mux_InitConfigArr);

/* Install PIT ch0 handler */
INT_SYS_InstallHandler(PIT_Ch0_IRQn, &pitCh0Handler, NULL);

/* Initialize the timer module */
PIT_DRV_Init(INST_PIT1, &pit1_InitConfig);
/* Initialize timer channel 0 */
PIT_DRV_InitChannel(INST_PIT1, &pit1_Ch0Config);
/* Start the counter for channel 0 */
PIT_DRV_StartChannel(INST_PIT1, 0U);
```

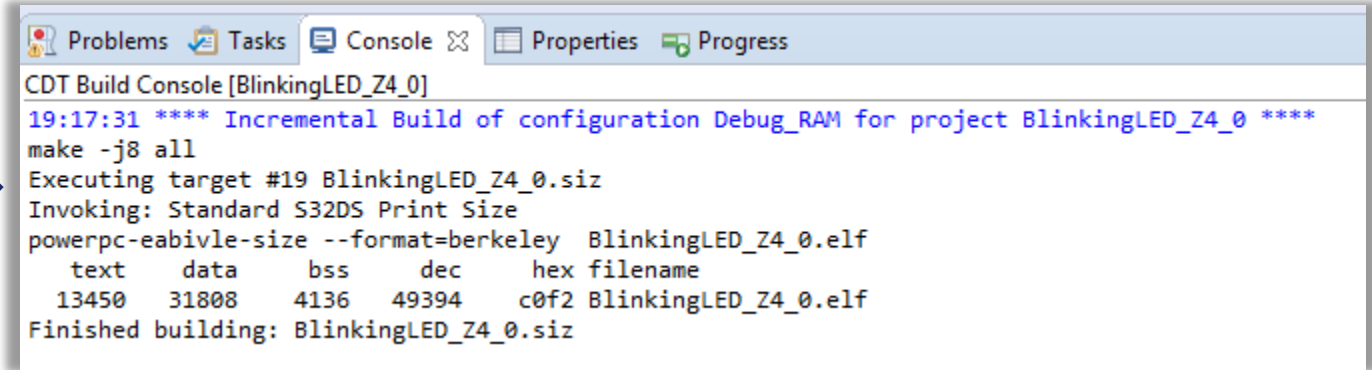
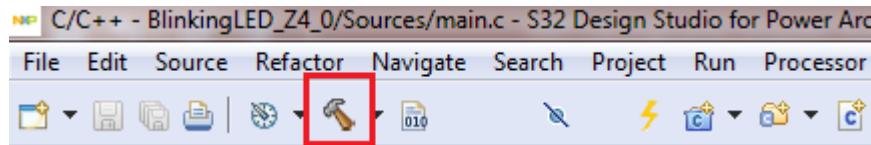
Hands-on – Blinking LED: Build configuration

- Click the 'manage configurations' button and select Debug_RAM build configuration (more easy to debug)

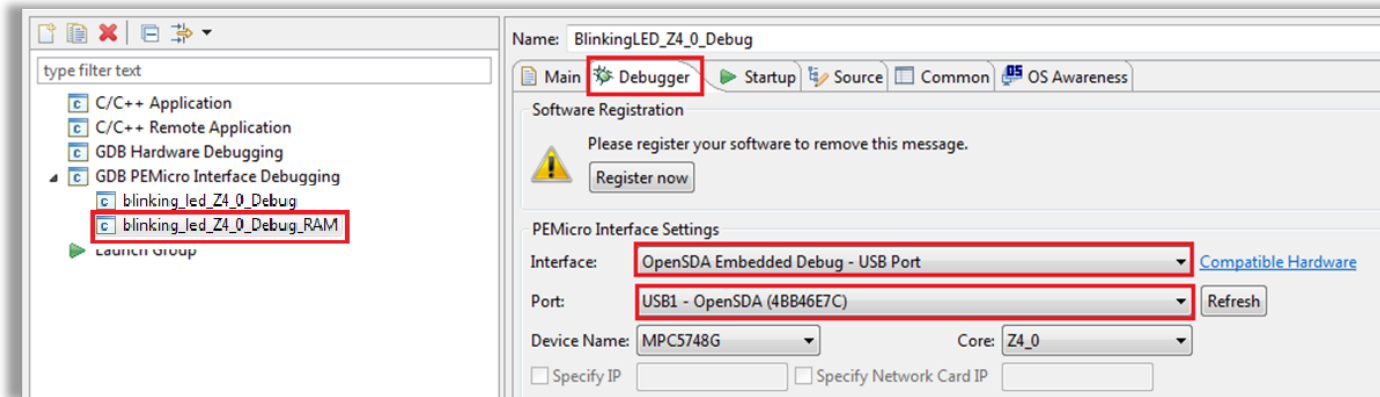
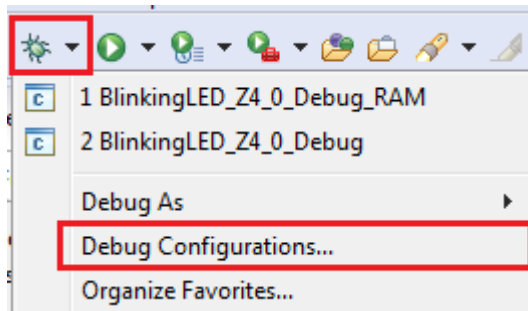


Hands-on – Blinking LED: Build and Debug

- Click the 'build project' button – make sure there are no compilation errors

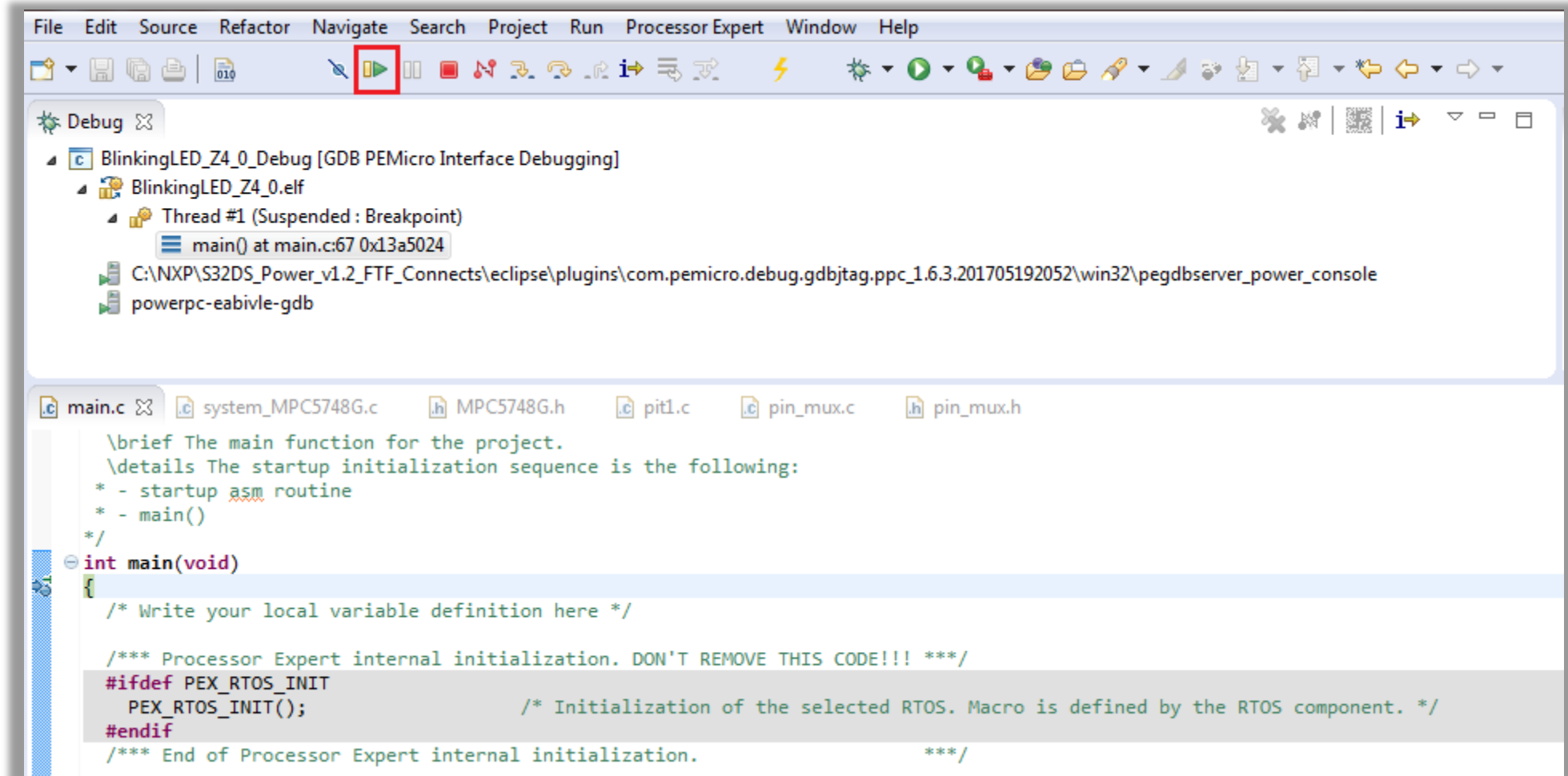


- Select the correct debug configuration and interface to debug the application



Hands-on – Blinking LED: Build and Debug

- In the debug perspective, click the 'Resume' button or press F8 to run the application
- User LED2 should blink every 0.5 seconds





03.

Hands-on – ENET - CAN

Hands-on – ENET–CAN: Objective

- In this lab you will learn:
 - About the features of the Ethernet MAC module on MPC5748G
 - About the features of FlexCAN module on MPC5748G
 - How to import example projects provided with SDK
 - How to configure ENET driver for rx/tx
 - How to configure FlexCAN driver for rx/tx
 - How to initiate a simple communication between:
 - PC and board via ENET
 - Two boards via CAN

Hands-on – ENET–CAN: ENET Theory

- Implements the full 802.3 specification
- Dynamically configurable to support 10/100-Mbit/s operation
- Seamless interface to commercial ethernet PHY:
 - a 4-bit Media Independent Interface (MII) operating at 2.5/25 MHz
 - a 4-bit non-standard MII-Lite operating at 2.5/25 MHz
 - a 2-bit Reduced MII (RMII) operating at 50 MHz
- Supports VLAN-tagged frames according to IEEE 802.1Q
- Programmable MAC address
- Programmable promiscuous mode support to omit MAC destination address checking
- Multicast and unicast address filtering on receive based on 64-entry hash table
- MDIO master interface for PHY device configuration and management
- Supports IPv4 and IPv6
- Automatic IP-header and payload checksum calculation and verification
- 2 ENET IP instances on MPC5748G

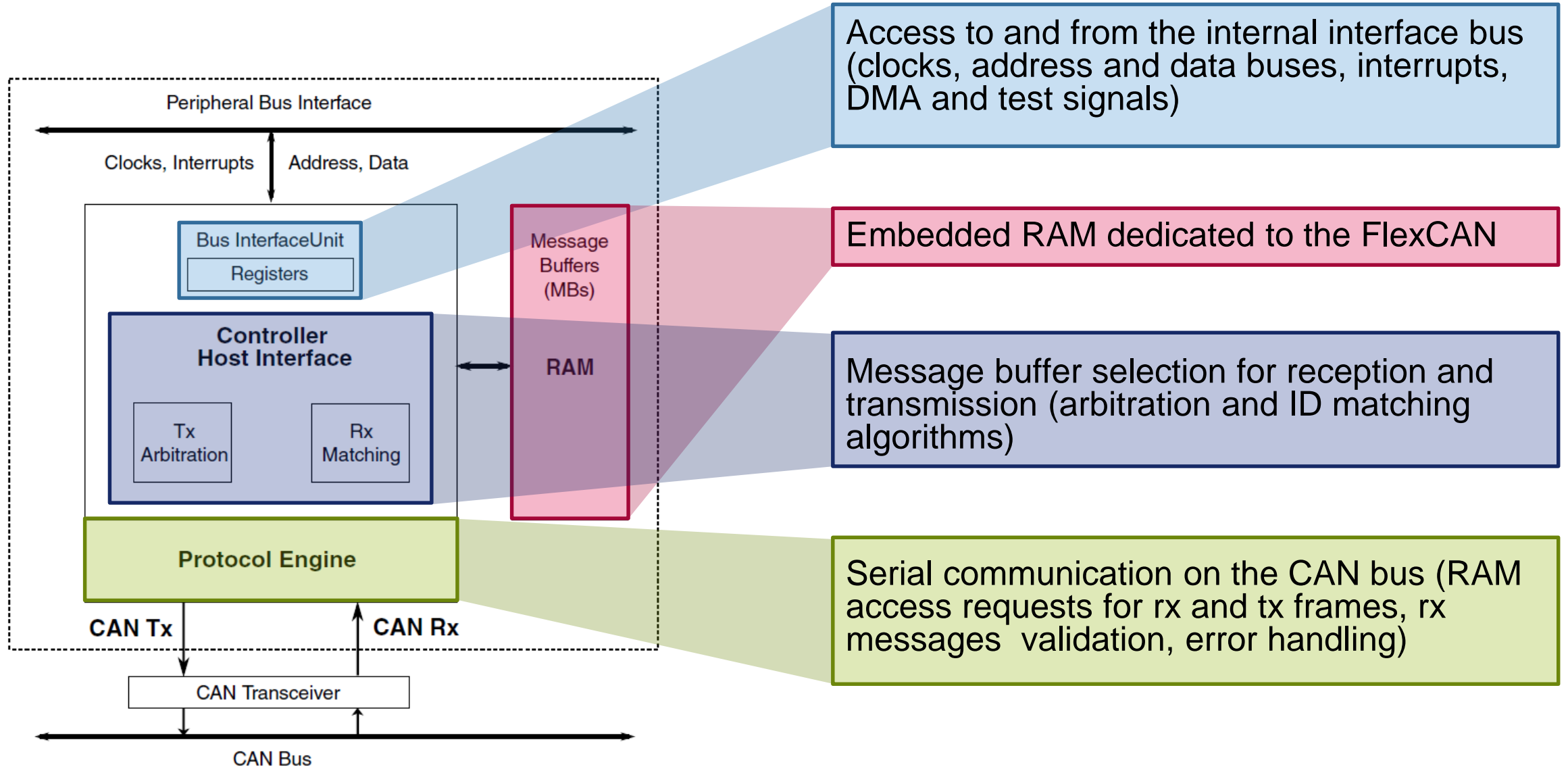


Hands-on – ENET–CAN: CAN Theory



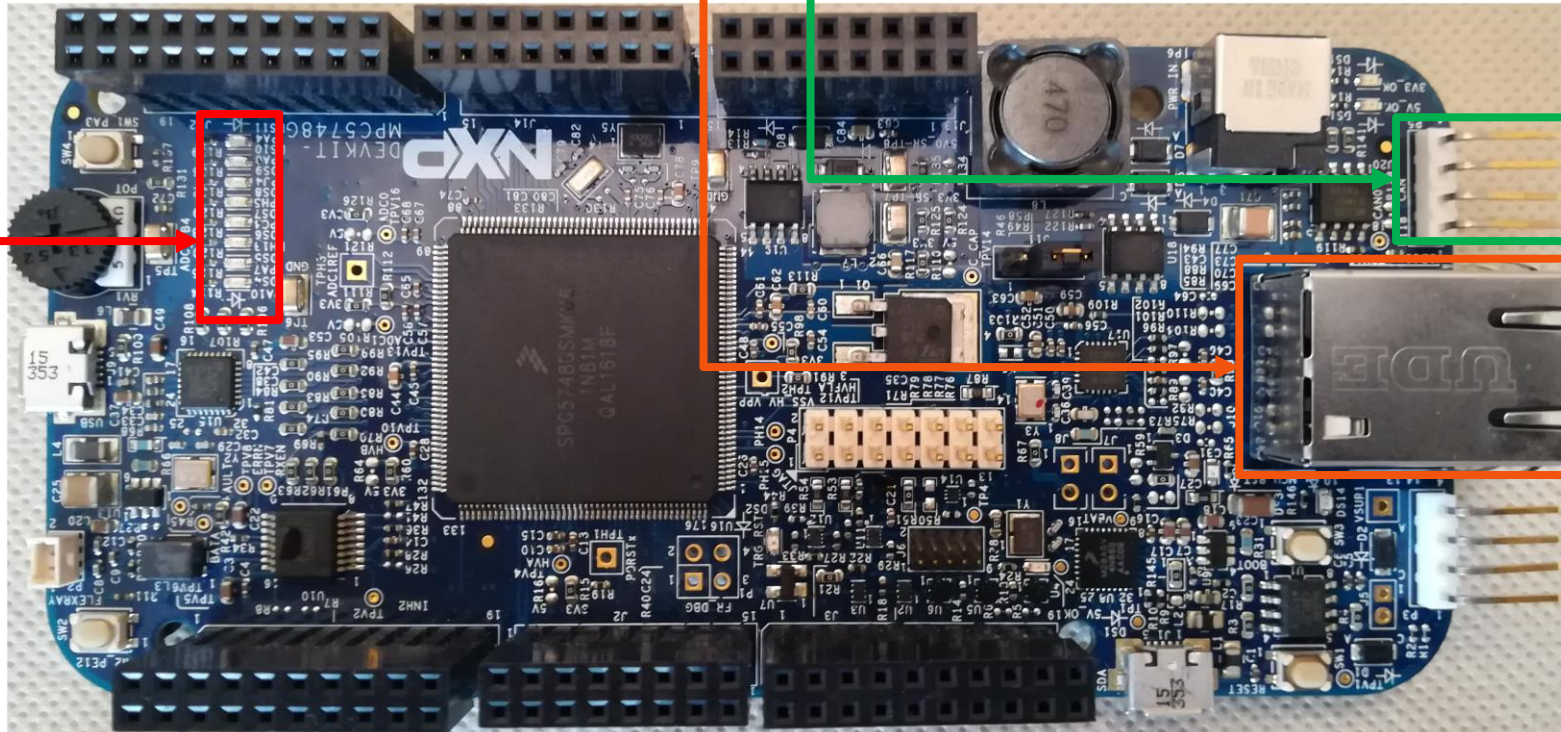
- Full implementation of the CAN FD & CAN 2.0 B
 - data field bitrate up to 8Mbps
- Flexible mailboxes (0/8/16/32/64 bytes data length)
- Listen-Only mode capability
- Programmable Loop-Back mode supporting self-test operation
- Programmable transmission priority scheme
- Independence from the transmission medium
- CRC status for transmitted message
- Full featured Rx FIFO with storage capacity for 6 frames
- DMA request for Rx FIFO
- Programmable clock source to the CAN Protocol Interface, either bus clock or crystal oscillator
- 100% backward compatibility with previous FlexCAN version
- 8 FlexCAN instances

Hands-on – ENET–CAN: CAN Theory



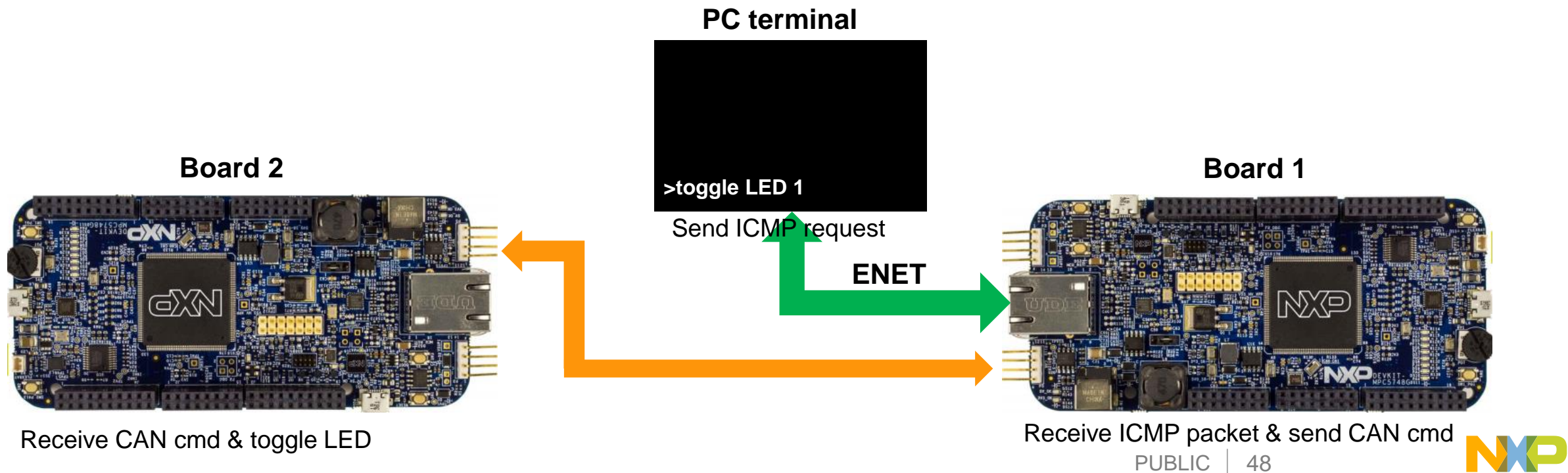
Hands-on – ENET–CAN: Resources

- Resources to be used:
 - on-board user LEDs (hardwired to GPIOs)
 - CAN connector
 - Ethernet port



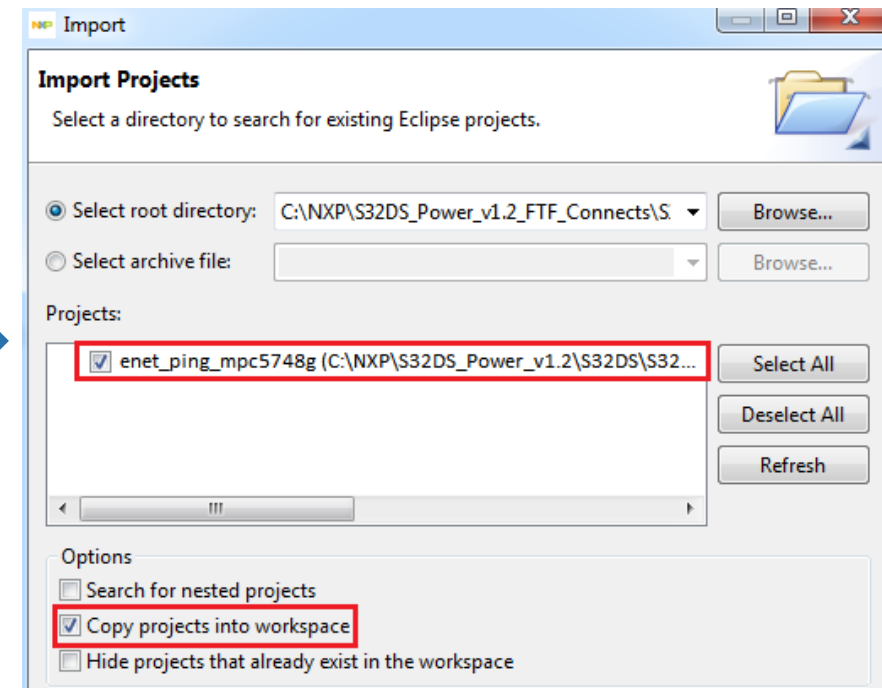
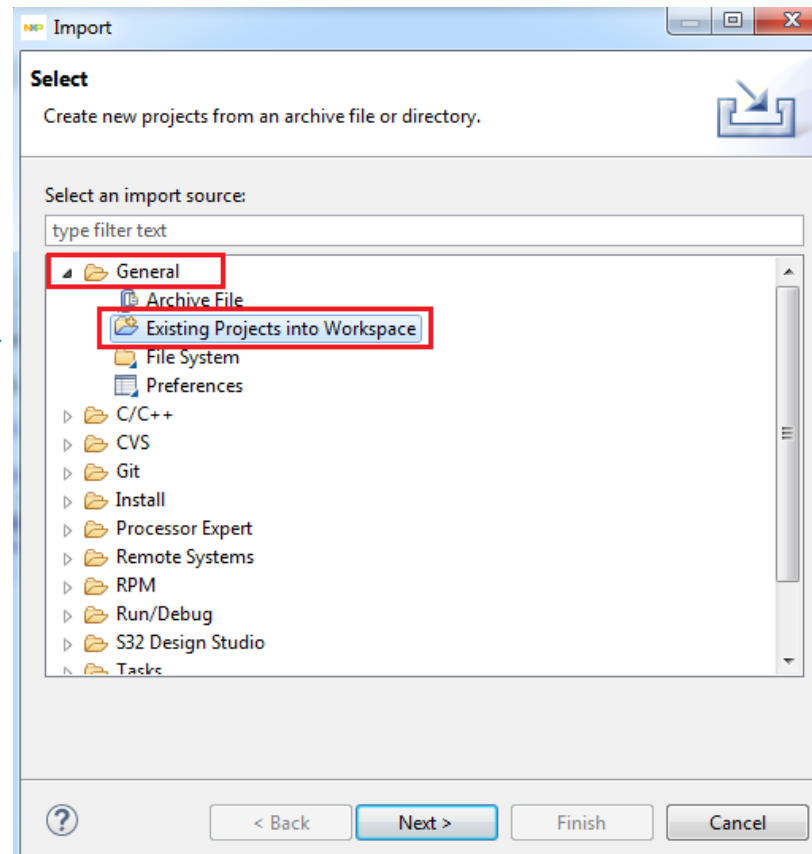
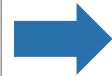
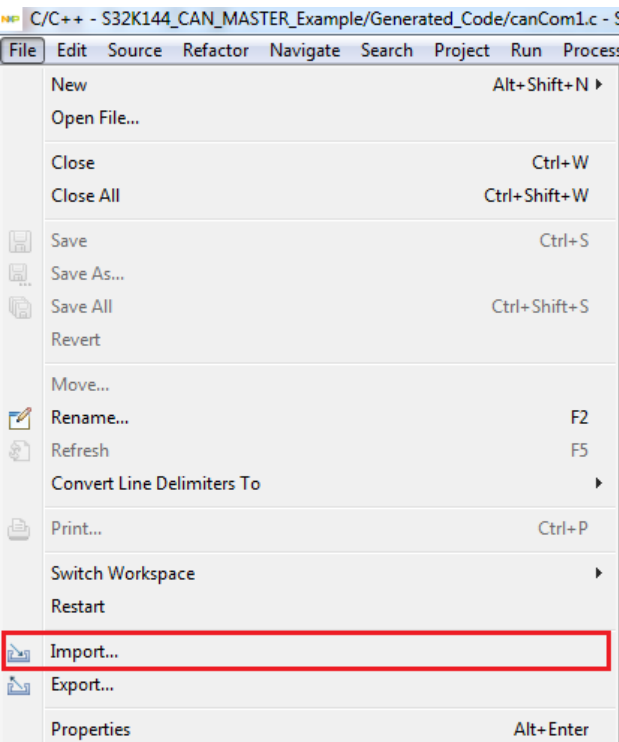
Hands-on – ENET–CAN: Lab Preview

- Board1 connected to PC via Ethernet
- Board1 and Board2 connected via CAN bus
- Command sent from PC to board1 (ICMP frames with variable payload length)
- Command sent from board1 to board2 (CAN message)
- Board2 toggle LEDs upon successful command receipt



Hands-on – ENET–CAN: Import Example Project

- Import 'enet_ping' example provided with the SDK:
 - File->Import->General->Existing Projects into Workspace->Select root directory
 - Select:
{DS_InstallationFolder}\S32DS\S32_SDK_MPC5748G_EAR_0.8.0\examples\MPC5748G\driver_examples\communication\enet_ping
 - Make sure 'Copy projects into workspace' is checked, so SDK example remains clean



Hands-on – ENET–CAN: Configuration

- ENET configuration
 - Receive callback function
 - Receive interrupt enabled
 - Insertion of IP + ICMP checksum
 - Set MAC address for frames

☒ Configuration 0

Name

Read only ☐

Maximum frame length (bytes)

Reception ring size

Transmission ring size

Callback

Interrupts

☒ Receive

☐ Transmit

☐ Error

☐ Wakeup

☐ Parser

☐ Timer

MII configuration

Mode

Speed

Duplex

Transmission acceleration options

☐ TX FIFO Shift-16

☒ Insertion of IP header checksum

☒ Insertion of protocol checksum

Reception acceleration options

☐ Padding Removal For Short IP Frames

☐ Discard Of Frames With Wrong IPv4 Header Checksum

☐ Discard Of Frames With Wrong Protocol Checksum

☐ Discard Of Frames With MAC Layer Errors

☐ RX FIFO Shift-16

Reception special options

☐ Payload Length Check

☐ Strip Received CRC

☐ Forward Pause Frames

☐ Padding Remove On Receive

☐ Flow Control Enable

☐ Broadcast Frame Reject

☐ Promiscuous Mode

☐ MII Loopback

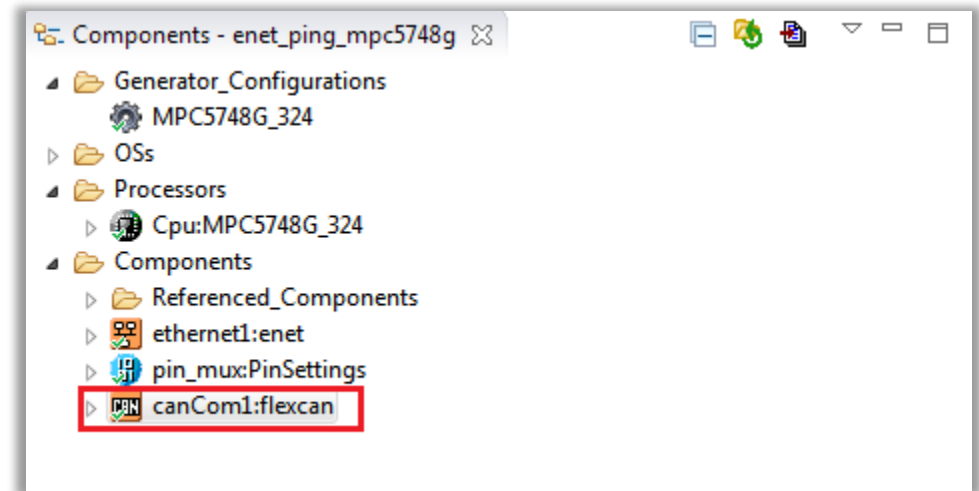
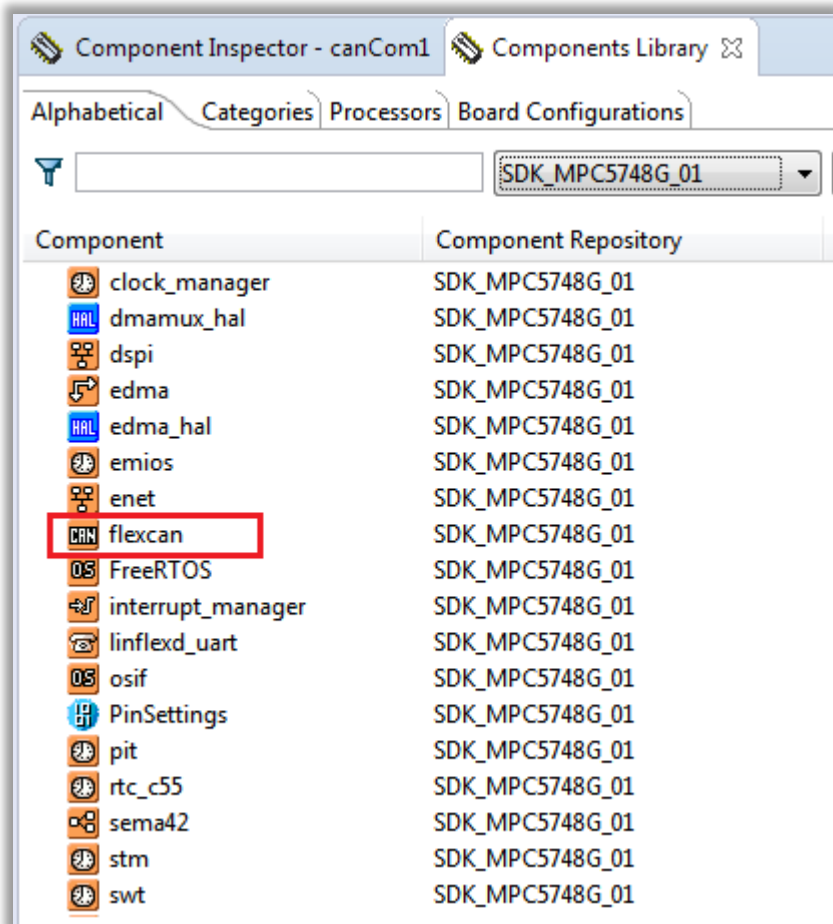
Transmission special options

☐ Do not append CRC

☒ Set MAC Address On Transmit

Hands-on – ENET–CAN: Configuration

- From 'Components Library' view, double-click 'flexcan' component to add it the project



Hands-on – ENET–CAN: Configuration

- No need to change default configuration for CAN:
 - standard CAN (no FD), minimum payload, 500 kbps

☒ Configuration 0

Name: canCom1_InitConfig0

Read only: ☒

Enable FD: ☐

PE clock source: FXOSC

MBs number: 16

RxFIFO ID filters number: 8 Rx FIFO Filters

Use Rx FIFO: ☐

RxFIFO transfer type: Using interrupts

RxFIFO transfer DMA channel number: Channel 0

Operation Mode: Normal

Payload Size: FLEXCAN_PAYLOAD_SIZE_8

Module clock: 80 MHz

PE clock: 40 MHz

Bitrate to time segments: ☒

Bitrate configuration							
Item	Propagation segment	Phase segment 1	Phase segment 2	Prescaler Division Factor	Resync jump width	Bitrate [kbit/s]	Sampling point [%]
Arbitration Phase	7	4	1	4	1	500	87.5
Data Phase	7	4	1	4	1	500	87.5

Hands-on – ENET–CAN: Configuration

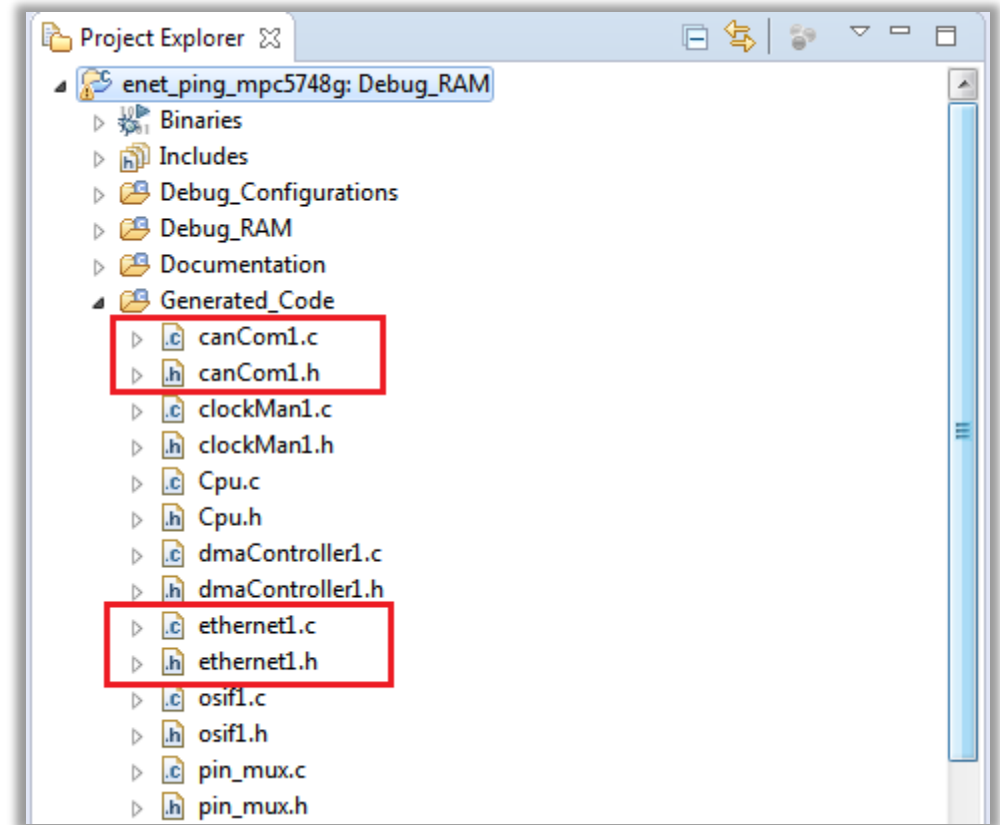
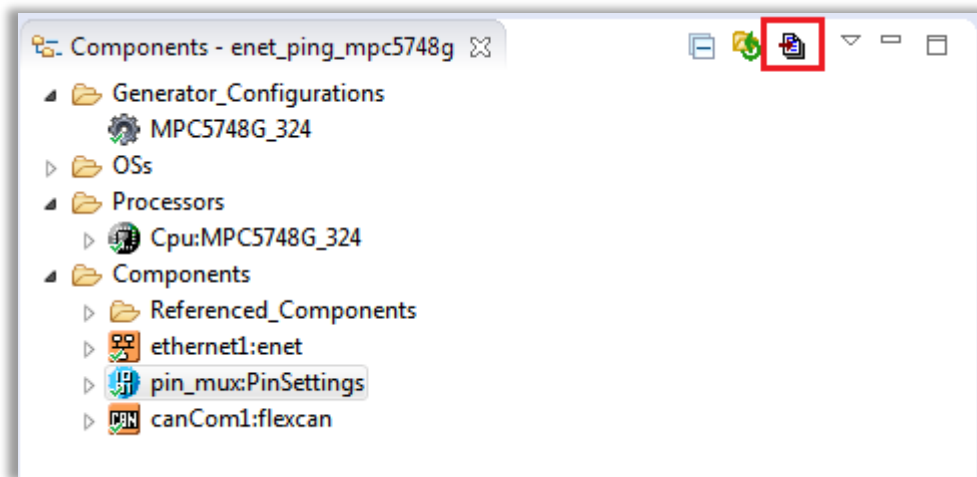
- From 'pin_mux' component:
 - Select PB0 & PB1 for CAN0 (internally connected to the board CAN connector)
 - GPIOs PA0 & PA10 as output (internally connected to board LEDs)

ADC CAN CMP DCI DSPI ENET FCCU FlexRay I2C LINFlexD MLB Platform PowerAndGround SIUL2 S									
Signals	Pin/Signal Selection		Direction						
▲ CAN_0									
Receiver Input	PB[1]		Input						
Transmitter Output	PB[0]		Output						
▲ CAN_1									

ADC CAN CMP DCI DSPI ENET FCCU FlexRay I2C LINFlexD MLB Platform PowerAndGround SIUL2 SPI									
Signals	Pin/Signal Selection		Direction						
GPI 58	PD[10]		Input						
GPI 59	PD[11]		Input						
GPIO 0	PA[0]		Output						
GPIO 1	No pin routed		No pin routed						
GPIO 2	No pin routed		No pin routed						
GPIO 3	PA[3]		Not Specified						
GPIO 4	No pin routed		No pin routed						
GPIO 5	No pin routed		No pin routed						
GPIO 6	No pin routed		No pin routed						
GPIO 7	PA[7]		Not Specified						
GPIO 8	PA[8]		Not Specified						
GPIO 9	PA[9]		Input						
GPIO 10	PA[10]		Output						
GPIO 11	PA[11]		Not Specified						
GPIO 12	No pin routed		No pin routed						

Hands-on – ENET–CAN: Configuration

- Once the configuration is done, hit the 'Generate code' button
- Configuration structures are generated in 'Generated_Code' project folder



Hands-on – ENET–CAN: Application Code

- Open the main.c file in text editor view
 - 1) Below ENET driver init function, call the FlexCAN init driver function and define a 'flexcan_data_info_t' structure needed by the CAN driver to send data:

```
/* Initialize FlexCAN driver */
FLEXCAN_DRV_Init(INST_CANCOM1, &canCom1_State, &canCom1_InitConfig0);

/* Set information about the data to be sent */
flexcan_data_info_t dataInfo =
{
    .data_length = 1U,                /* 1 byte in length */
    .msg_id_type = FLEXCAN_MSG_ID_STD, /* Standard message ID */
    .enable_brs  = false,             /* Bit rate switch disabled */
    .fd_enable   = false,             /* Flexible data rate disabled */
    .fd_padding  = 0U                 /* Use zeros for FD padding */
};
```


Hands-on – ENET–CAN: Application Code

2) Add the following macros and global variables above the 'copy_buff' function:

```
#define MB 1U          /* Index of the CAN message buffer */
#define MSG_ID 1U      /* CAN message ID */

uint8_t led1FrameLen = 100U; /* ENET frame length for LED1 toggle */
uint8_t led2FrameLen = 101U; /* ENET frame length for LED2 toggle */
volatile bool toggleLed1 = false; /* volatile flag updated in ISR */
volatile bool toggleLed2 = false; /* volatile flag updated in ISR */
```

```
uint8_t g_macAddr[6] = {0x11, 0x22, 0x33, 0x44, 0x55, 0x66};

#define MB 1U
#define MSG_ID 1U

uint8_t led1FrameLen = 100U;
uint8_t led2FrameLen = 101U;
volatile bool toggleLed1 = false;
volatile bool toggleLed2 = false;

void copy_buff(uint8_t *dest, uint8_t *src, uint8_t len)
```

Hands-on – ENET–CAN: Application Code

3) Inside the ENET rx callback function, add the following code, signaling the application that a new command should be sent via CAN (**only for applications sending data via CAN**):

```
if (buffList[0].length == led1FrameLen)
{
    /* frame length for LED1 toggle */
    toggleLed1 = true;
}
else if (buffList[0].length == led2FrameLen)
{
    /* frame length for LED2 toggle */
    toggleLed2 = true;
}
```

```
void rx_callback(uint8_t instance, enet_event_t event, enet_buf_t *buffList)
{
    if (event == ENET_RX_EVENT)
    {
        if (buffListSize > 0U)
        {
            if (buffList[0].length < FRAME_ICMP_TYPE_OFFSET)
                return;

            if (buffList[0].length == led1FrameLen)
            {
                toggleLed1 = true;
            }
            else if (buffList[0].length == led2FrameLen)
            {
                toggleLed2 = true;
            }
        }
    }
}
```

/* Check that the received frame was an ICMP echo

Hands-on – ENET–CAN: Application Code

- 4) Inside the 'main' function, replace the contents of the infinite loop with the following code
(**only for the application sending data via CAN**):

```
if (toggleLed1)
{
    /* Configure TX message buffer with index MB and message ID MSG_ID */
    FLEXCAN_DRV_ConfigTxMb(INST_CANCOM1, MB, &dataInfo, MSG_ID);
    /* Execute send non-blocking */
    FLEXCAN_DRV_Send(INST_CANCOM1, MB, &dataInfo, MSG_ID, &led1FrameLen);
    /* Wait until the previous FlexCAN send is completed */
    while (FLEXCAN_DRV_GetTransferStatus(INST_CANCOM1, MB) == STATUS_BUSY);
    /* Reset the global flag */
    toggleLed1 = false;
}
if (toggleLed2)
{
    /* Configure TX message buffer with index MB and message ID MSG_ID */
    FLEXCAN_DRV_ConfigTxMb(INST_CANCOM1, MB, &dataInfo, MSG_ID);
    /* Execute send non-blocking */
    FLEXCAN_DRV_Send(INST_CANCOM1, MB, &dataInfo, MSG_ID, &led2FrameLen);
    /* Wait until the previous FlexCAN send is completed */
    while (FLEXCAN_DRV_GetTransferStatus(INST_CANCOM1, MB) == STATUS_BUSY);
    /* Reset the global flag */
    toggleLed2 = false;
}
```

Hands-on – ENET–CAN: Application Code

4) The new 'main' code (**only for the application sending data via CAN**):

```
ENET_DRV_Init(INST_ETHERNET1, &ethernet1_State, &ethernet1_InitConfig0, &ethernet1_buffConfig0, g_macAddr);

/* Initialize FlexCAN driver */
FLEXCAN_DRV_Init(INST_CANCOM1, &canCom1_State, &canCom1_InitConfig0);

/* Set information about the data to be sent */
flexcan_data_info_t dataInfo =
{
    .data_length = 1U,                /* 1 byte in length */
    .msg_id_type = FLEXCAN_MSG_ID_STD, /* Standard message ID */
    .enable_brs = false,              /* Bit rate switch disabled */
    .fd_enable = false,               /* Flexible data rate disabled */
    .fd_padding = 0U                 /* Use zeros for FD padding */
};

for (;;)
{
    if (toggleLed1)
    {
        /* Configure TX message buffer with index TX_MSG_ID and TX_MAILBOX*/
        FLEXCAN_DRV_ConfigTxMb(INST_CANCOM1, MB, &dataInfo, MSG_ID);
        /* Execute send non-blocking */
        FLEXCAN_DRV_Send(INST_CANCOM1, MB, &dataInfo, MSG_ID, &led1FrameLen);
        /* Wait until the previous FlexCAN send is completed */
        while (FLEXCAN_DRV_GetTransferStatus(INST_CANCOM1, MB) == STATUS_BUSY);
        /* Reset the global flag */
        toggleLed1 = false;
    }
    if (toggleLed2)
    {
        /* Configure TX message buffer with index TX_MSG_ID and TX_MAILBOX*/
        FLEXCAN_DRV_ConfigTxMb(INST_CANCOM1, MB, &dataInfo, MSG_ID);
        /* Execute send non-blocking */
        FLEXCAN_DRV_Send(INST_CANCOM1, MB, &dataInfo, MSG_ID, &led2FrameLen);
        /* Wait until the previous FlexCAN send is completed */
        while (FLEXCAN_DRV_GetTransferStatus(INST_CANCOM1, MB) == STATUS_BUSY);
        /* Reset the global flag */
        toggleLed2 = false;
    }
}
```

Hands-on – ENET–CAN: Application Code

5) Inside the 'main' function, replace the contents of the infinite loop with the following code (**only for the application receiving data via CAN**):

```
/* Define receive buffer */
flexcan_msgbuff_t recvBuff;

/* Configure RX message buffer with index MB and message ID MSG_ID */
FLEXCAN_DRV_ConfigRxMb(INST_CANCOM1, MB, &dataInfo, MSG_ID);

/* Start receiving data in rx mailbox. */
FLEXCAN_DRV_Receive(INST_CANCOM1, MB, &recvBuff);

/* Wait until the previous FlexCAN receive is completed */
while(FLEXCAN_DRV_GetTransferStatus(INST_CANCOM1, MB) == STATUS_BUSY);

if (recvBuff.data[0] == led1FrameLen)
{
    SIUL2->GPDO[0] ^= SIUL2_GPDO_PDO_4n_MASK; /* toggle LED 1 */
}
else if (recvBuff.data[0] == led2FrameLen)
{
    SIUL2->GPDO[2] ^= SIUL2_GPDO_PDO_4n2_MASK; /* toggle LED 2 */
}
```

Hands-on – ENET–CAN: Application Code

5) The new ‘main’ code (**only for the application receiving data via CAN**):

```
ENET_DRV_Init(INST_ETHERNET1, &ethernet1_State, &ethernet1_InitConfig0, &ethernet1_buffConfig0, g_macAddr);

/* Initialize FlexCAN driver */
FLEXCAN_DRV_Init(INST_CANCOM1, &canCom1_State, &canCom1_InitConfig0);

/* Set information about the data to be sent */
flexcan_data_info_t dataInfo =
{
    .data_length = 1U,           /* 1 byte in length */
    .msg_id_type = FLEXCAN_MSG_ID_STD, /* Standard message ID */
    .enable_brs = false,        /* Bit rate switch disabled */
    .fd_enable = false,         /* Flexible data rate disabled */
    .fd_padding = 0U            /* Use zeros for FD padding */
};

for (;;)
{
    /* Define receive buffer */
    flexcan_msgbuff_t recvBuff;

    /* Configure RX message buffer with index MB and message ID MSG_ID */
    FLEXCAN_DRV_ConfigRxMb(INST_CANCOM1, MB, &dataInfo, MSG_ID);

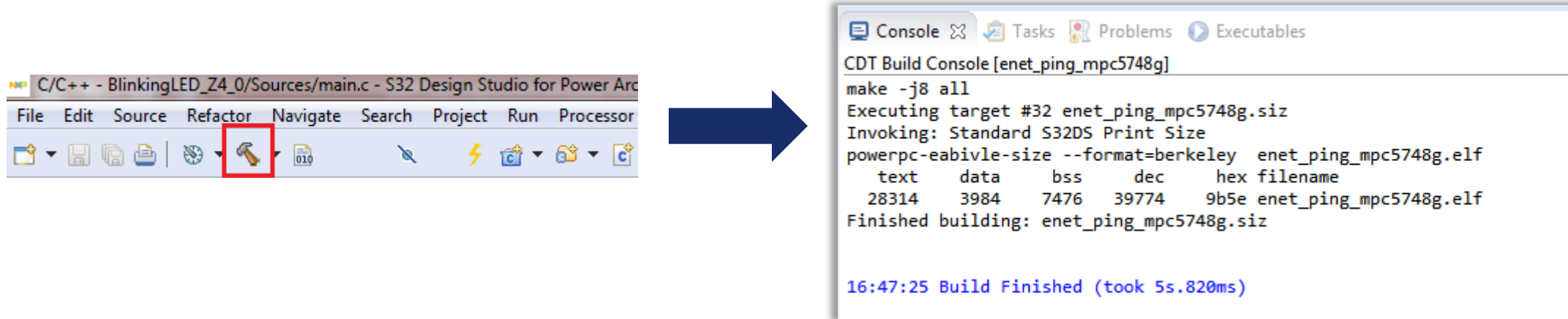
    /* Start receiving data in rx mailbox. */
    FLEXCAN_DRV_Receive(INST_CANCOM1, MB, &recvBuff);

    /* Wait until the previous FlexCAN receive is completed */
    while(FLEXCAN_DRV_GetTransferStatus(INST_CANCOM1, MB) == STATUS_BUSY);

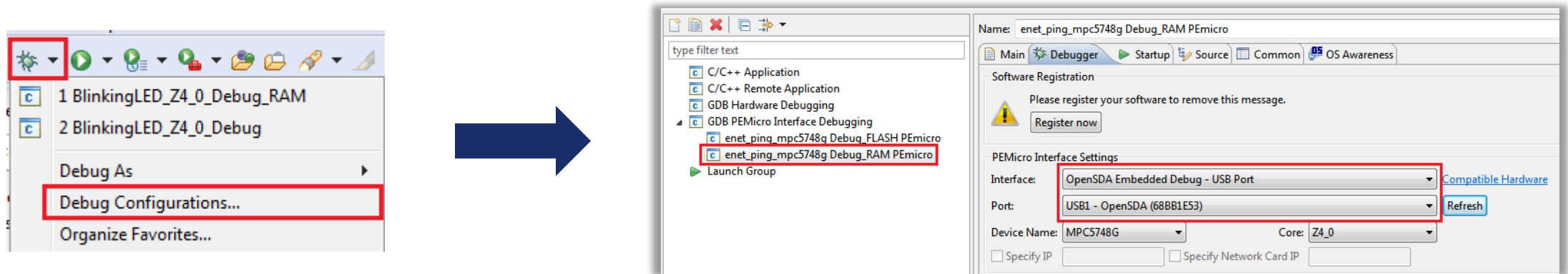
    if (recvBuff.data[0] == led1FrameLen)
    {
        SIUL2->GPD0[0] ^= SIUL2_GPDO_PDO_4n_MASK; /* toggle LED 1 */
    }
    else if (recvBuff.data[0] == led2FrameLen)
    {
        SIUL2->GPD0[2] ^= SIUL2_GPDO_PDO_4n2_MASK; /* toggle LED 2 */
    }
}
```

Hands-on – ENET–CAN: Build and Debug

- Click the 'build project' button – make sure there are no compilation errors

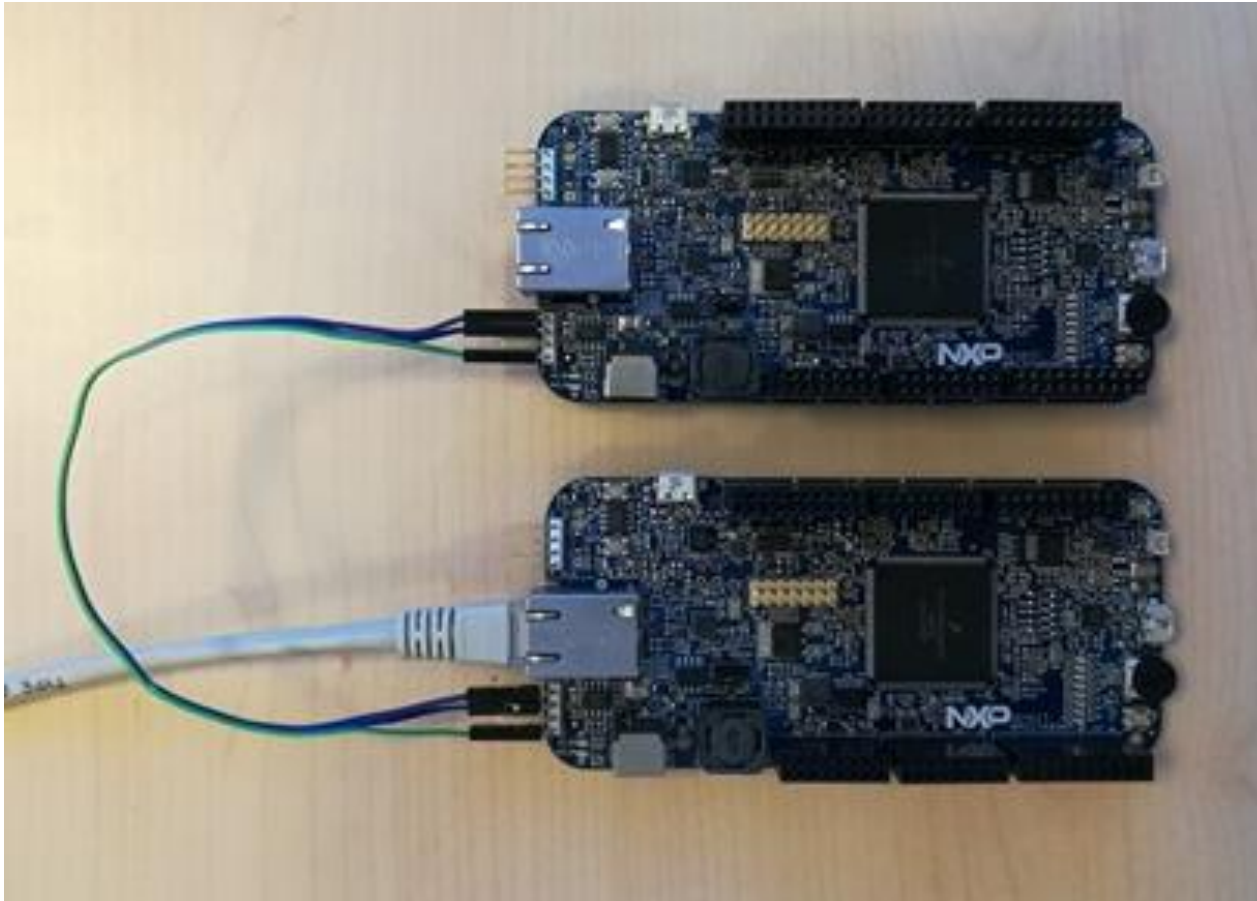


- Select the correct debug configuration and interface to debug the application



Hands-on – ENET–CAN: Board Setup

- Connect board1 to the PC using the Ethernet cable
- Connect board1 to board2 using the CAN connectors



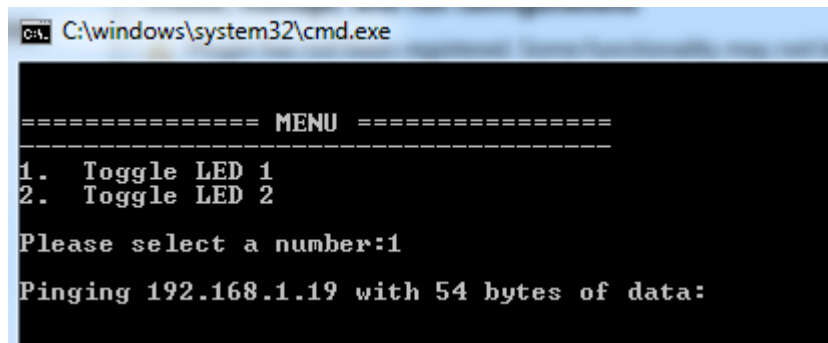
Hands-on – ENET–CAN: PC Setup

- In the ENET SDK example, the MAC address of the board is **11-22-33-44-55-66** and the IP is **192.168.1.19** (hardcoded values)
- Add this entry to your ARP table using the following command:

```
arp -s 192.168.1.19 11-22-33-44-55-66
```
- Set the IP address of appropriate ETH interface of the PC to be in the same network with the board:
 - **Control Panel > Network and Internet > Network Connections > Local Area Connection**
 - Select **Internet Protocol Version 4** and click **Properties**; configure the IP address and subnet mask. Configuration example: **IP - 192.168.1.5**, subnet mask - **255.255.255.0**

Hands-on – ENET–CAN: PC CLI

- Open 'demo.bat' script
- Use the available options to send commands to the board via ENET
 - Toggle led1 option sends an ICMP echo request with 54 bytes payload (100 bytes total Ethernet frame length)
 - Toggle led2 option sends an ICMP echo request with 55 bytes payload (101 bytes total Ethernet frame length)
- Selecting '1' will toggle the first LED on the second board, '2' will toggle the second LED on the second board



```
C:\windows\system32\cmd.exe

===== MENU =====
1.  Toggle LED 1
2.  Toggle LED 2
Please select a number:1
Pinging 192.168.1.19 with 54 bytes of data:
```

```
IF /I '%INPUT%'=='1' GOTO Selection1
IF /I '%INPUT%'=='2' GOTO Selection2

ECHO Please select a valid option!
GOTO MENU

:Selection1
ping -l 54 -n 1 192.168.1.19
GOTO MENU

:Selection2
ping -l 55 -n 1 192.168.1.19
GOTO MENU
```



SECURE CONNECTIONS
FOR A SMARTER WORLD