# HANDS-ON WORKSHOP: MPC574XG SDK FOR GATEWAY

## 1ST SDK FOR POWER ARCHITECTURE

JULY 2017

SECURE CONNECTIONS FOR A SMARTER WORLD

# 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

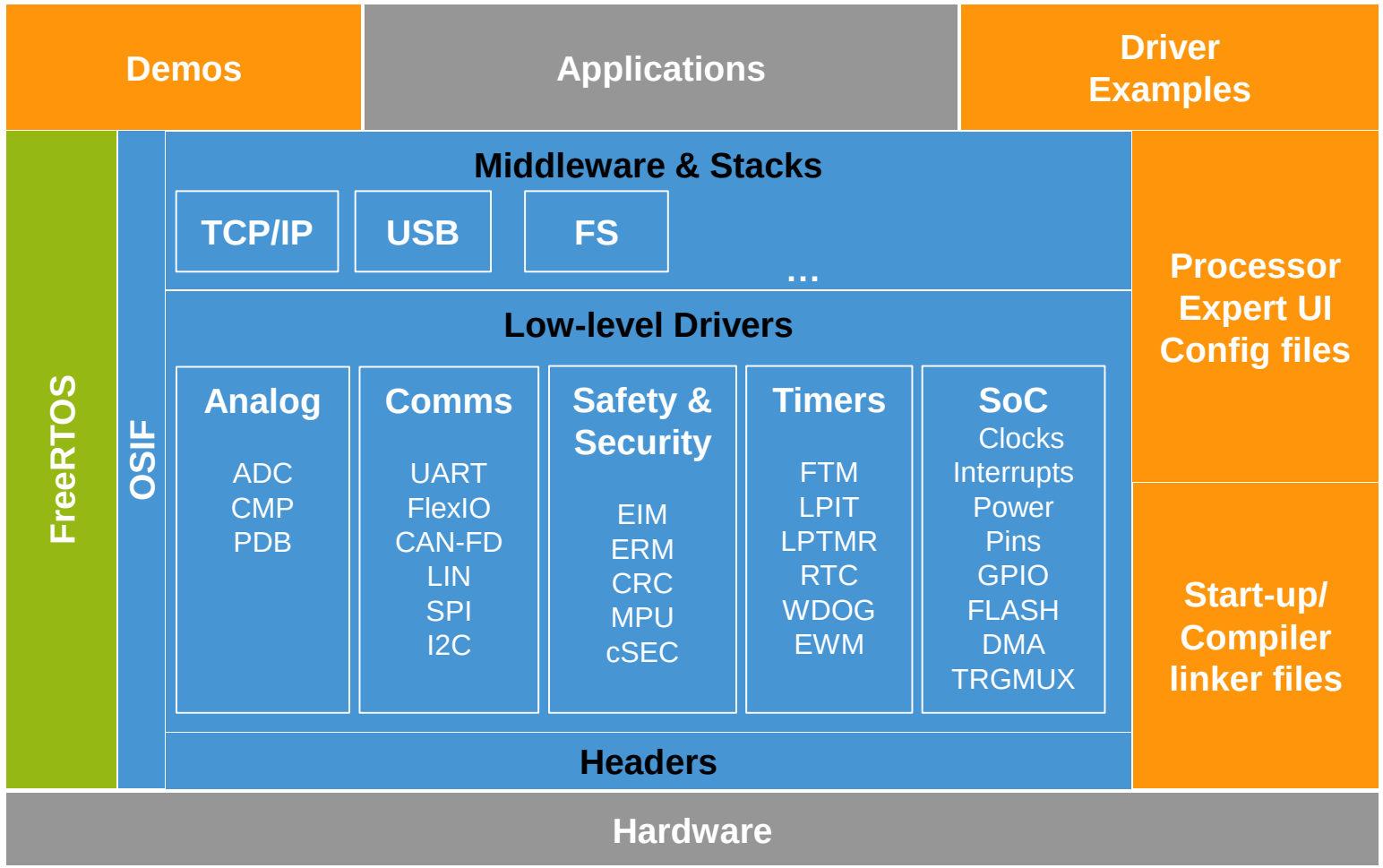

## Easy prototype

- 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

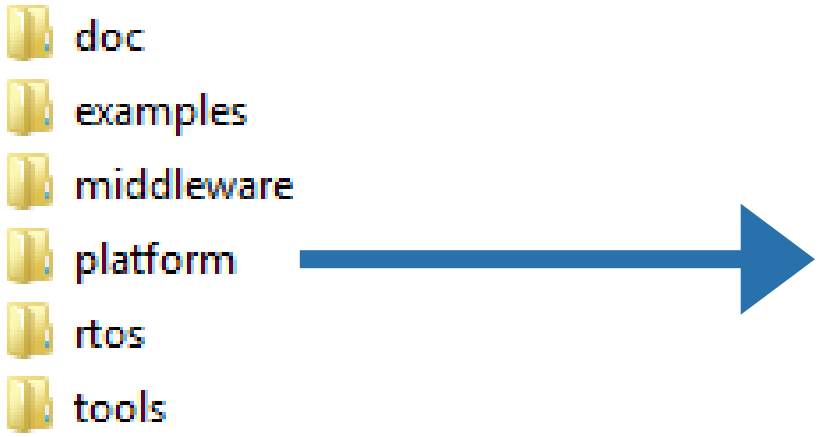



## Features

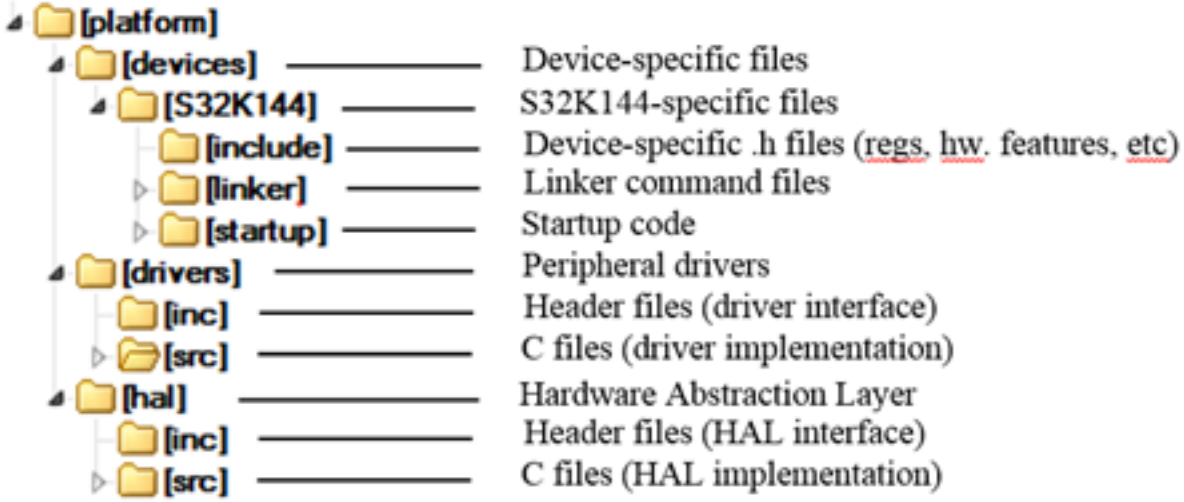
- Integrated **Non-Autosar SW Production-grade** software
- Graphical-based Configuration
- Layered Software Architecture
- Documented Source Code and Examples
- Integrated with S32 Design Studio and other IDEs
- FreeRTOS integration
- Multiple toolchains supported
- Several examples and demos

# S32 SDK – File Structure

## S32 SDK

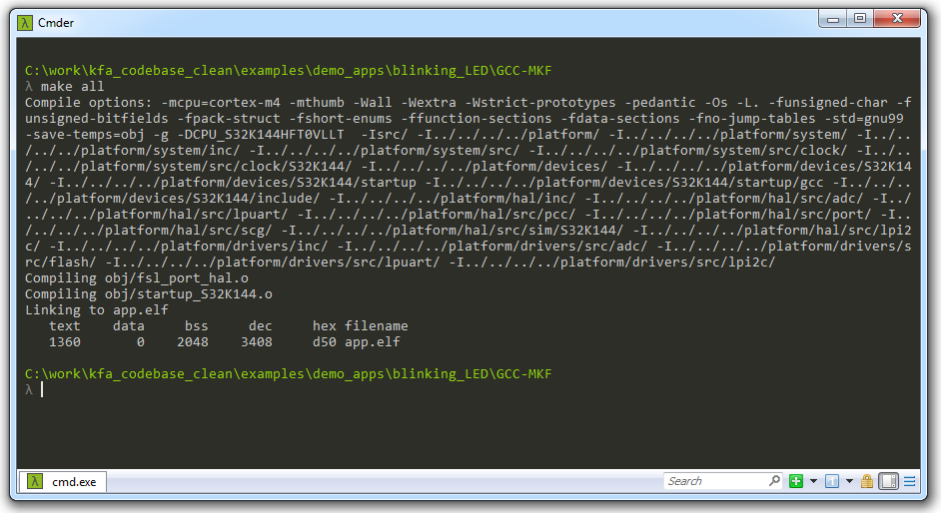


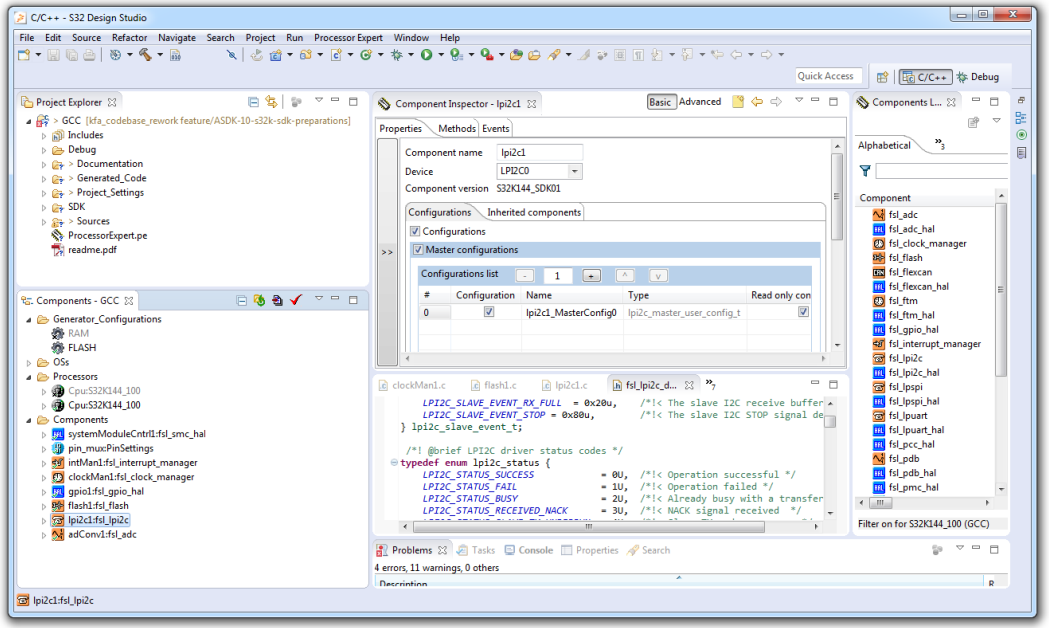

## Platform folder

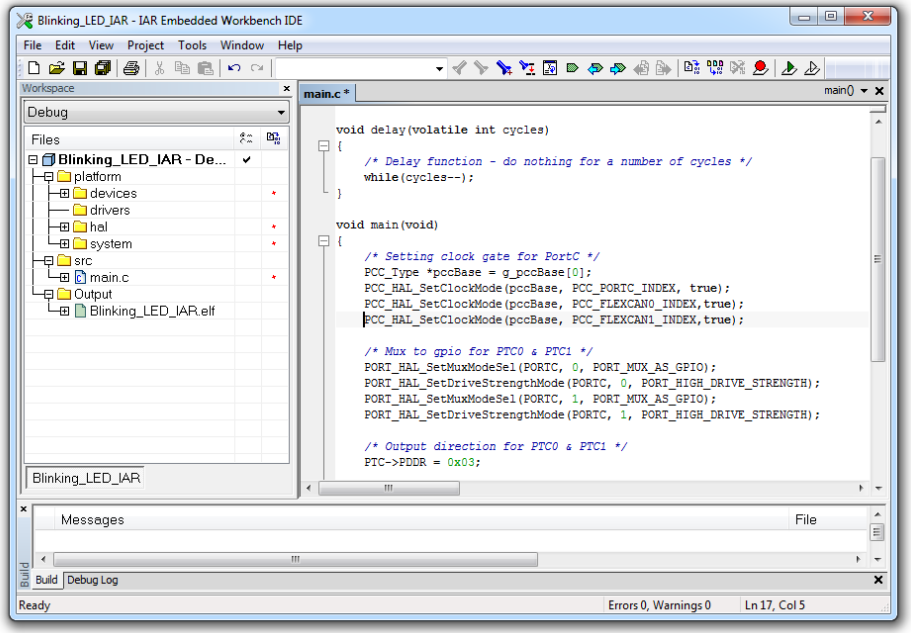


# S32 SDK – Usage

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

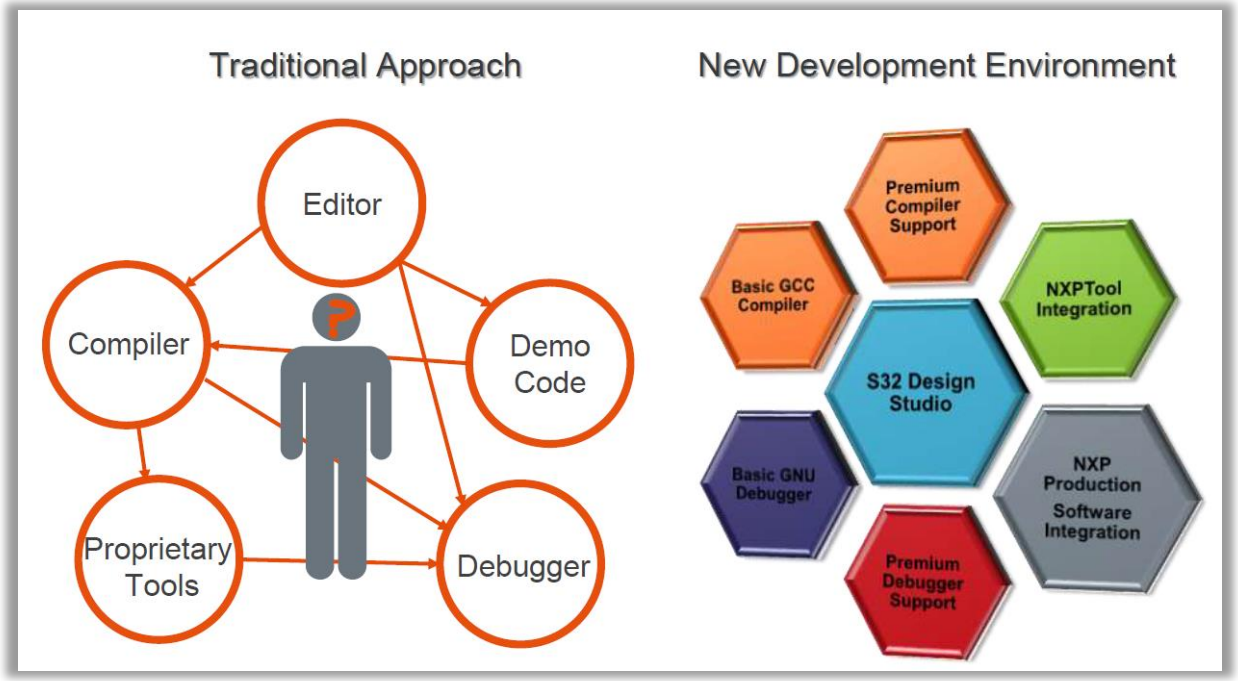


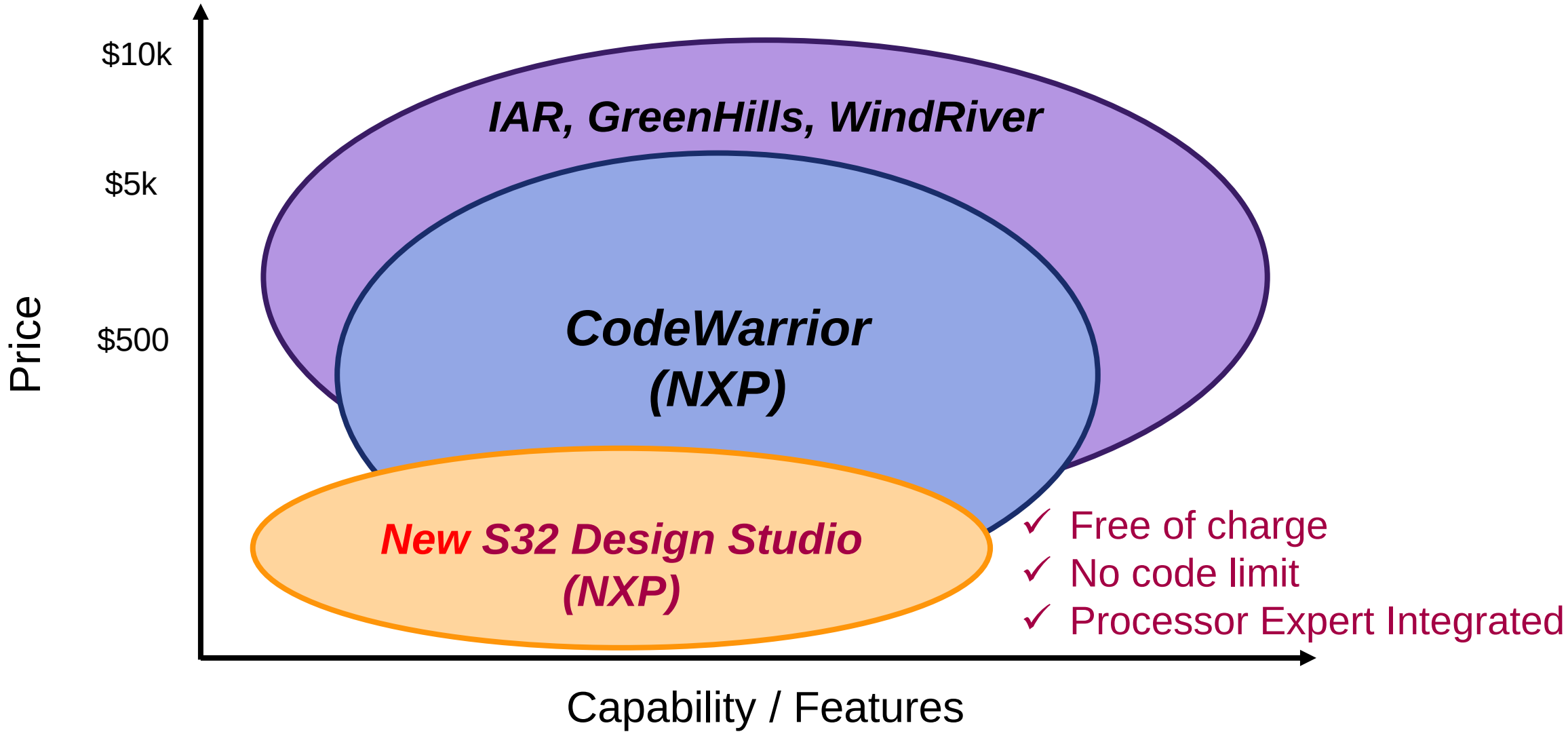



# NXP S32 Design Studio IDE [www.nxp.com/S32DS](http://www.nxp.com/S32DS)

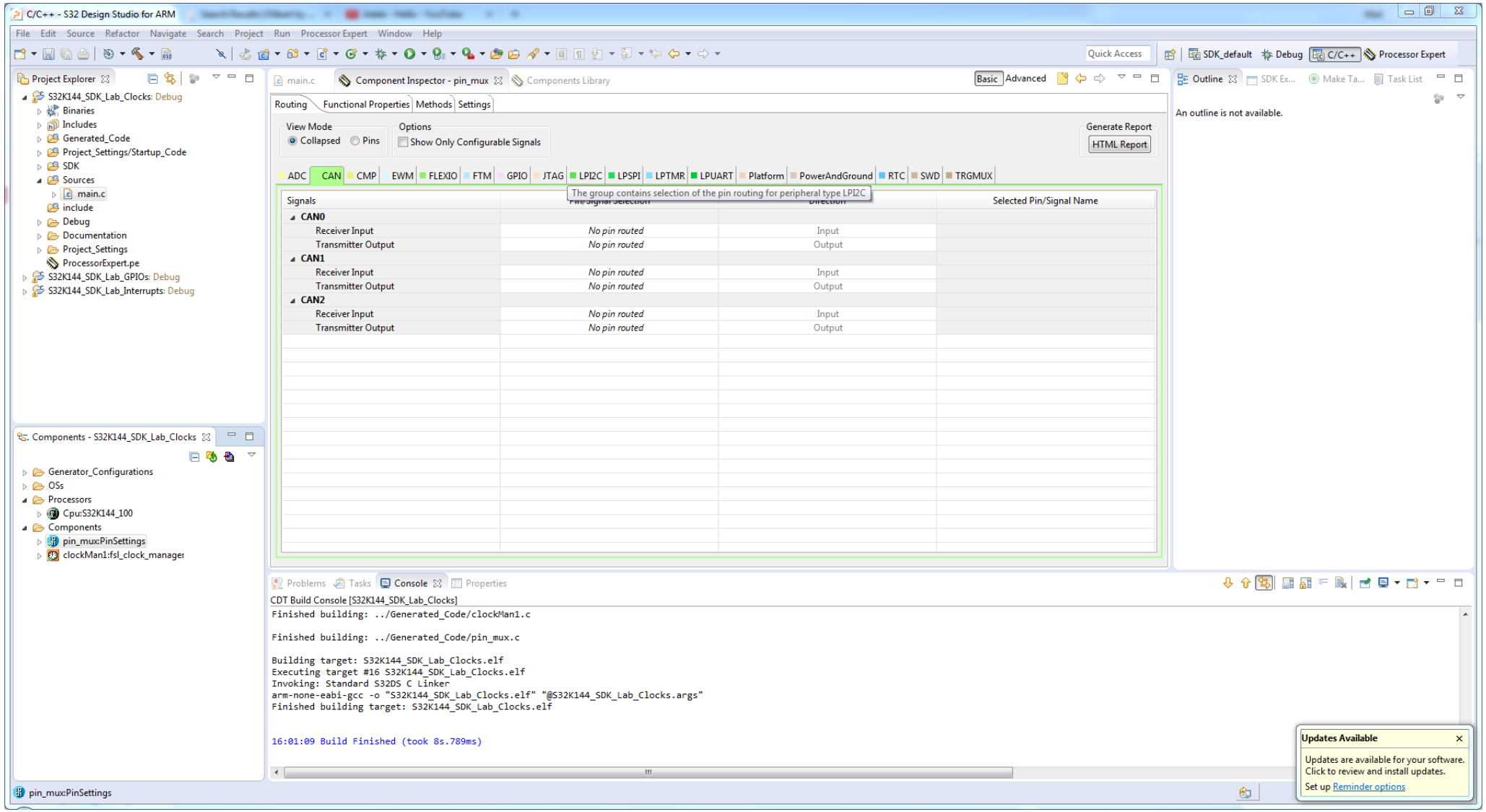
- Free of charge
- Unlimited code size
- Eclipse based environment
- GNU compiler & debugger integrated
- S32 SDK integrated (graphical configuration)
- Processor Expert integrated (automatic code generator)
- Can use with 3rd party compliers & debuggers (IAR) via Connection Utility
- Supports S32K and 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

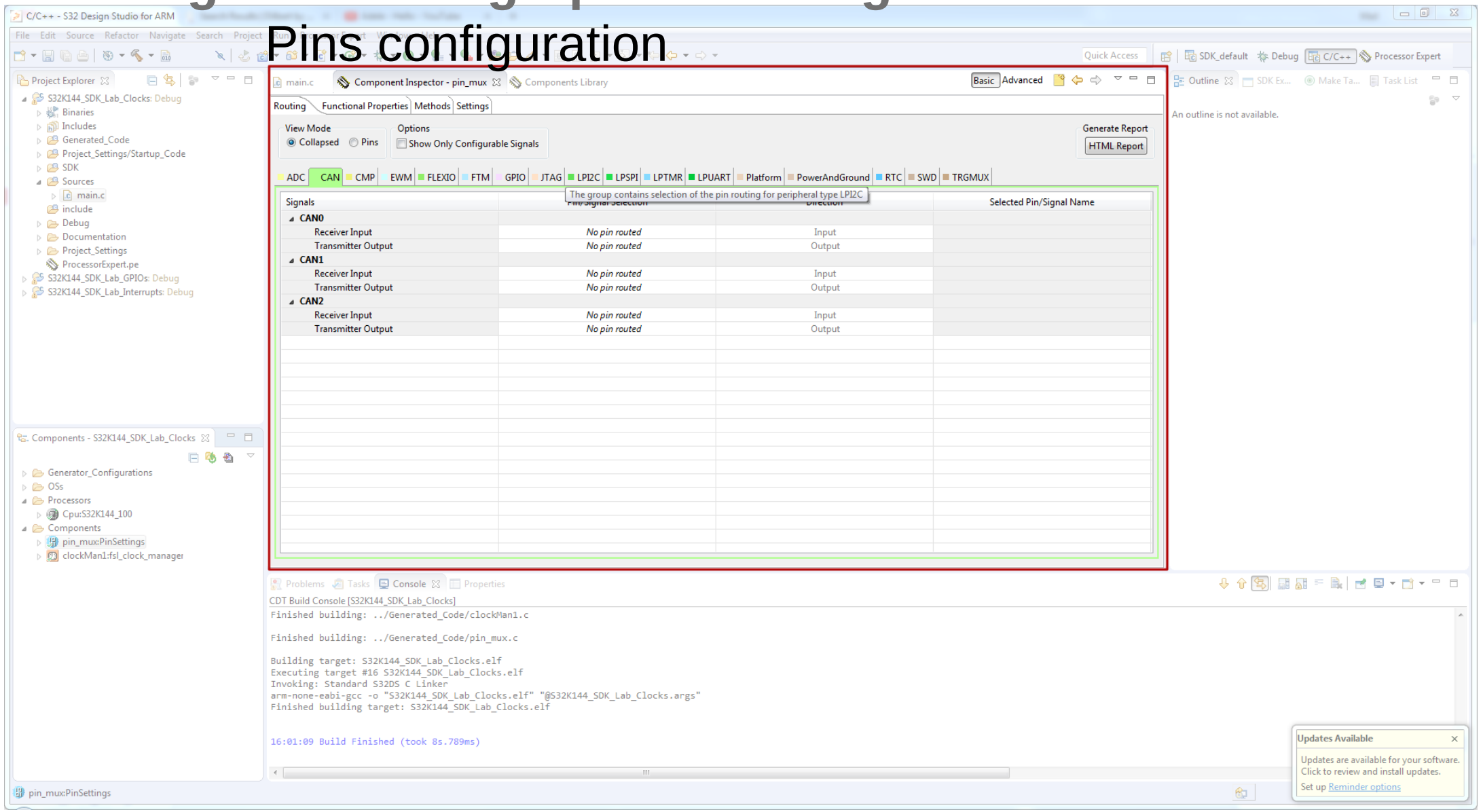


# S32 Design Studio – graphical configuration environment



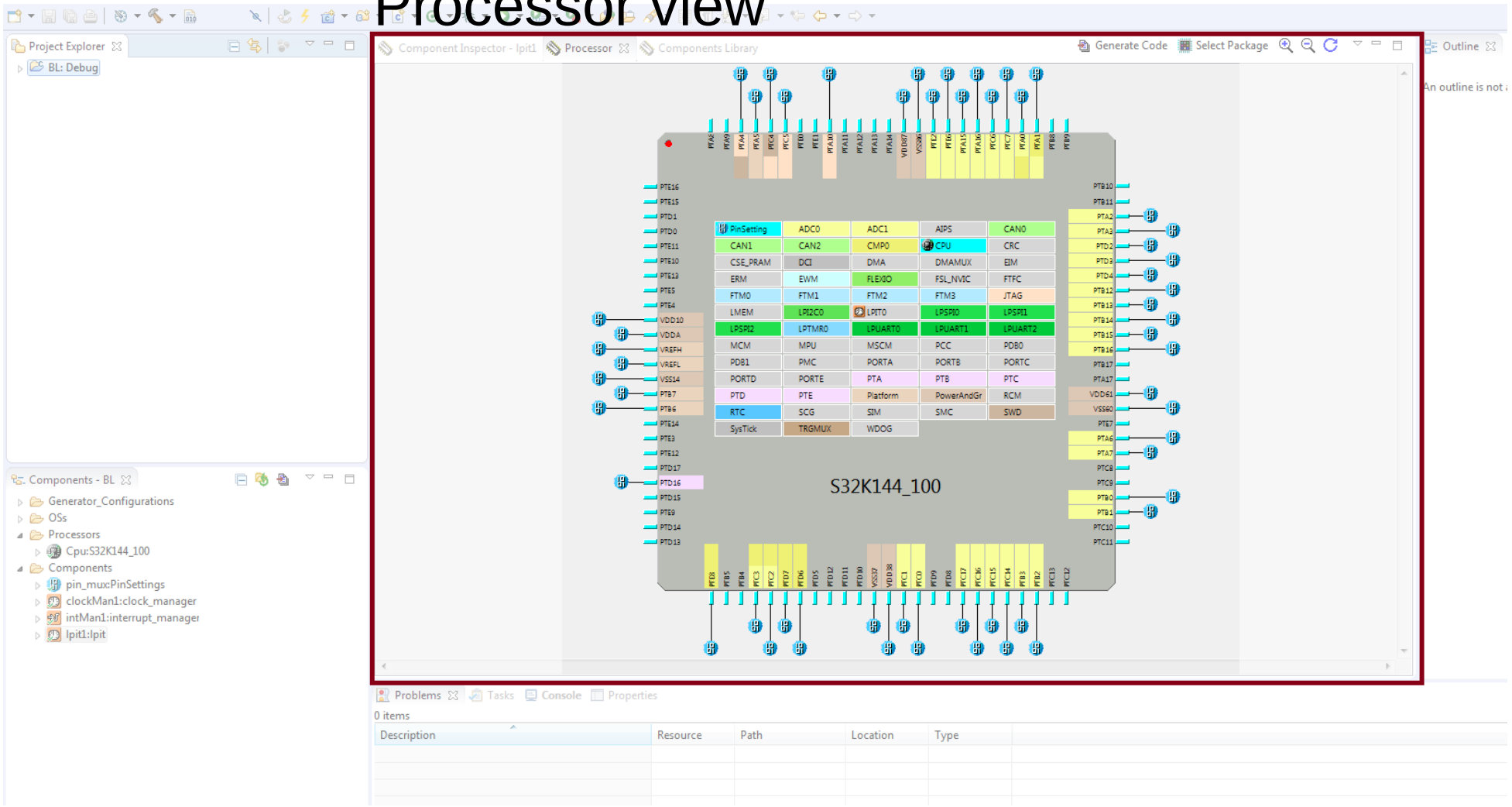
# S32 Design Studio – graphical configuration environment

## Pins configuration



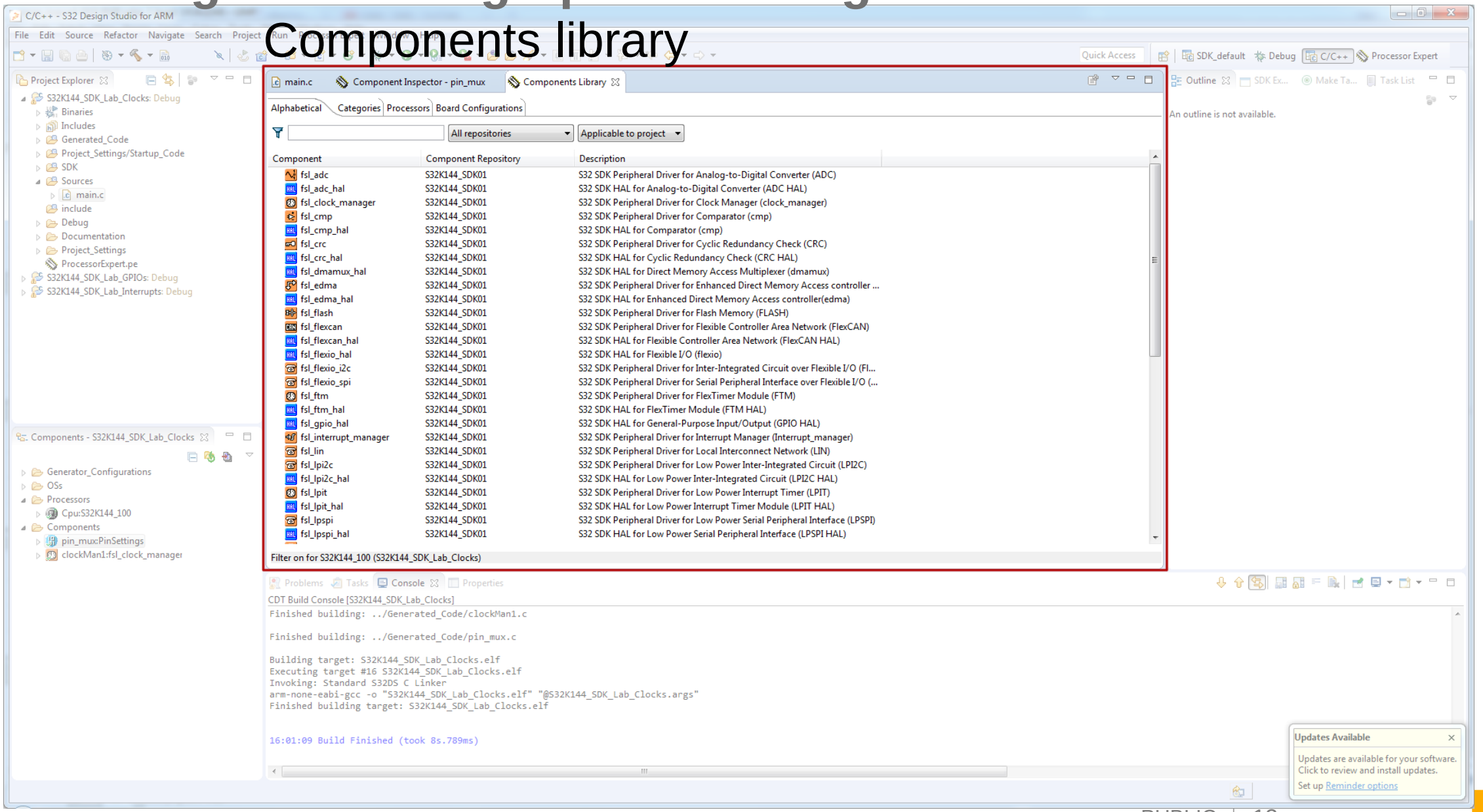
# S32 Design Studio – graphical configuration environment

## Processor view

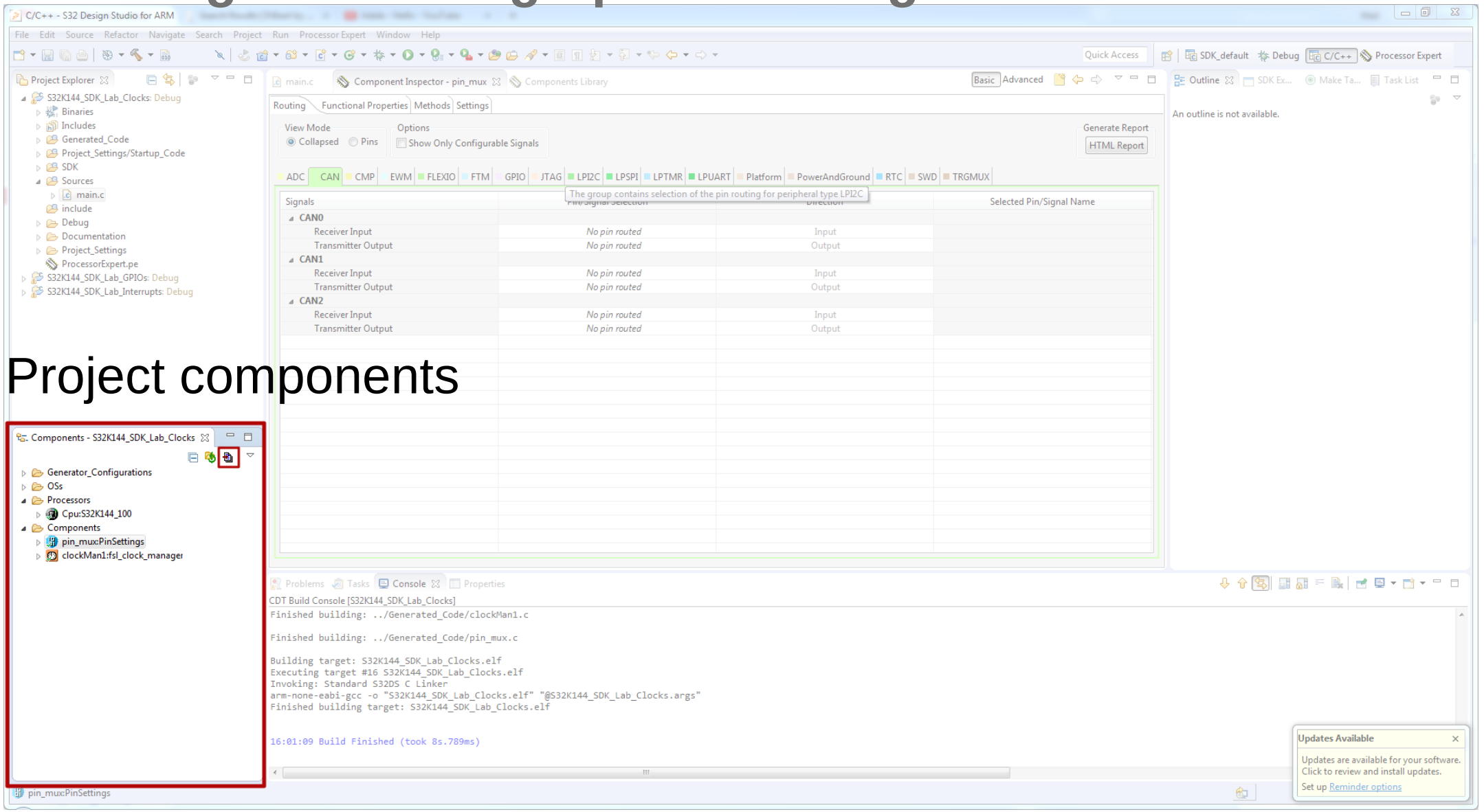


# S32 Design Studio – graphical configuration environment

## Components library



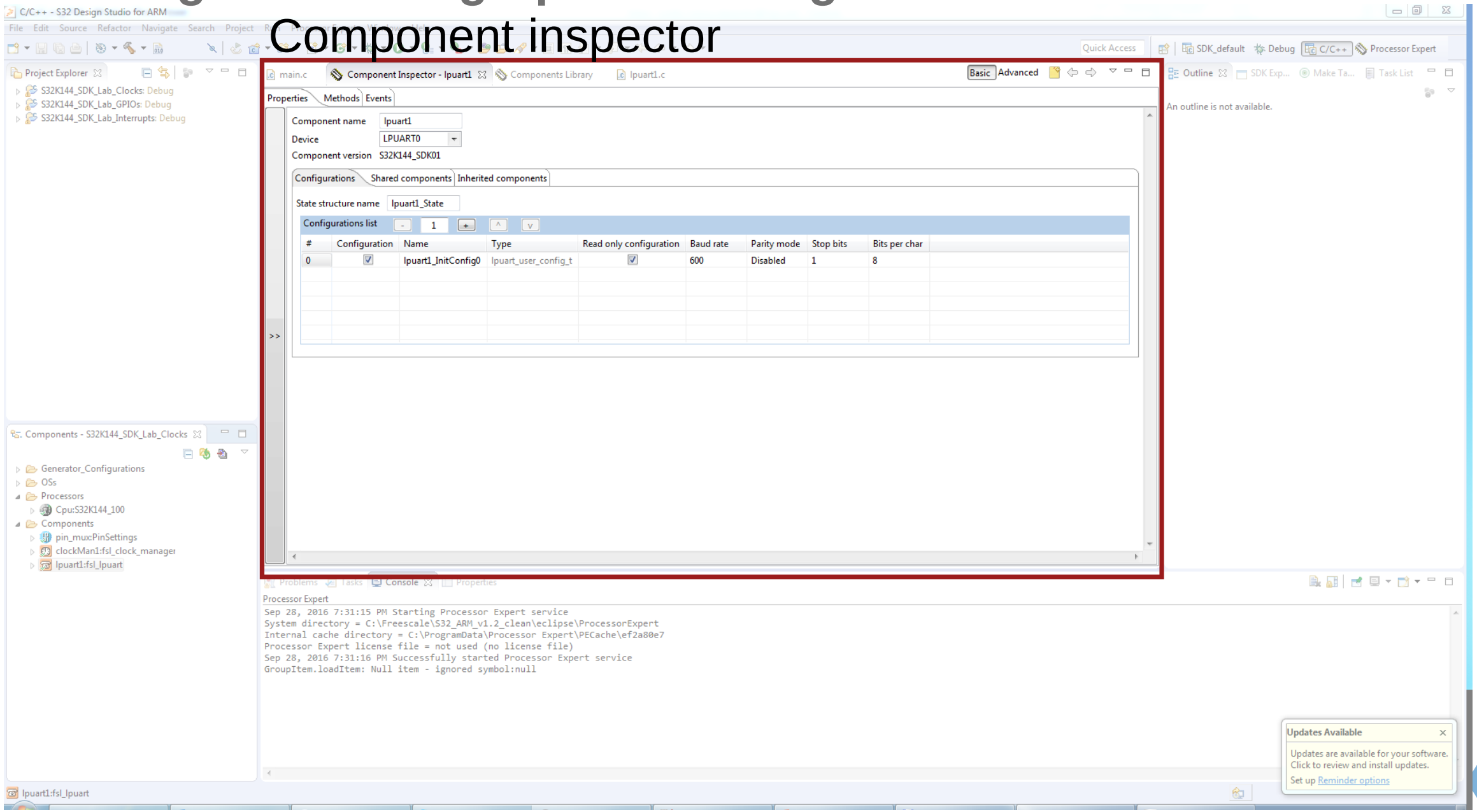
# S32 Design Studio – graphical configuration environment




Project components

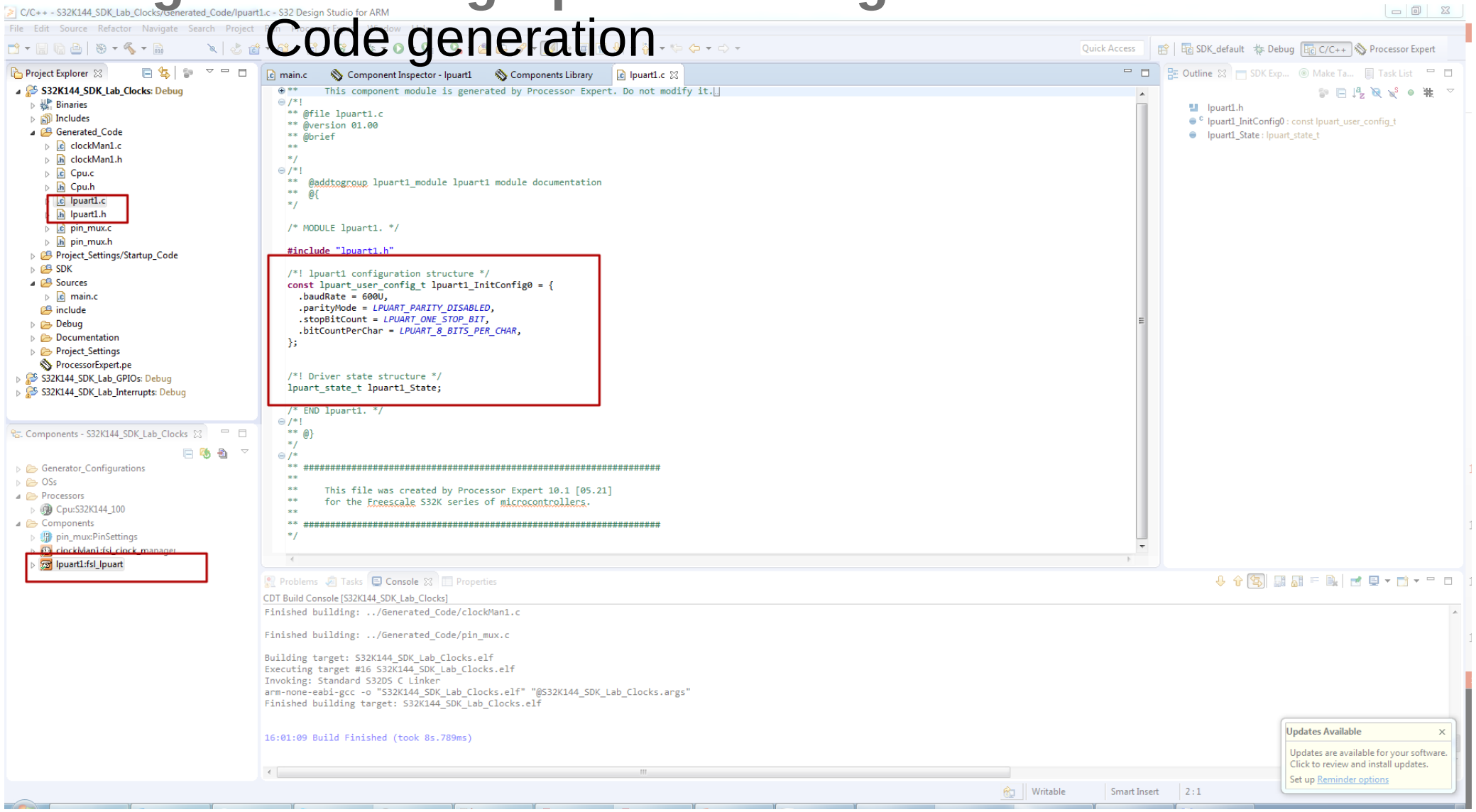
# S32 Design Studio – graphical configuration environment

## Component inspector



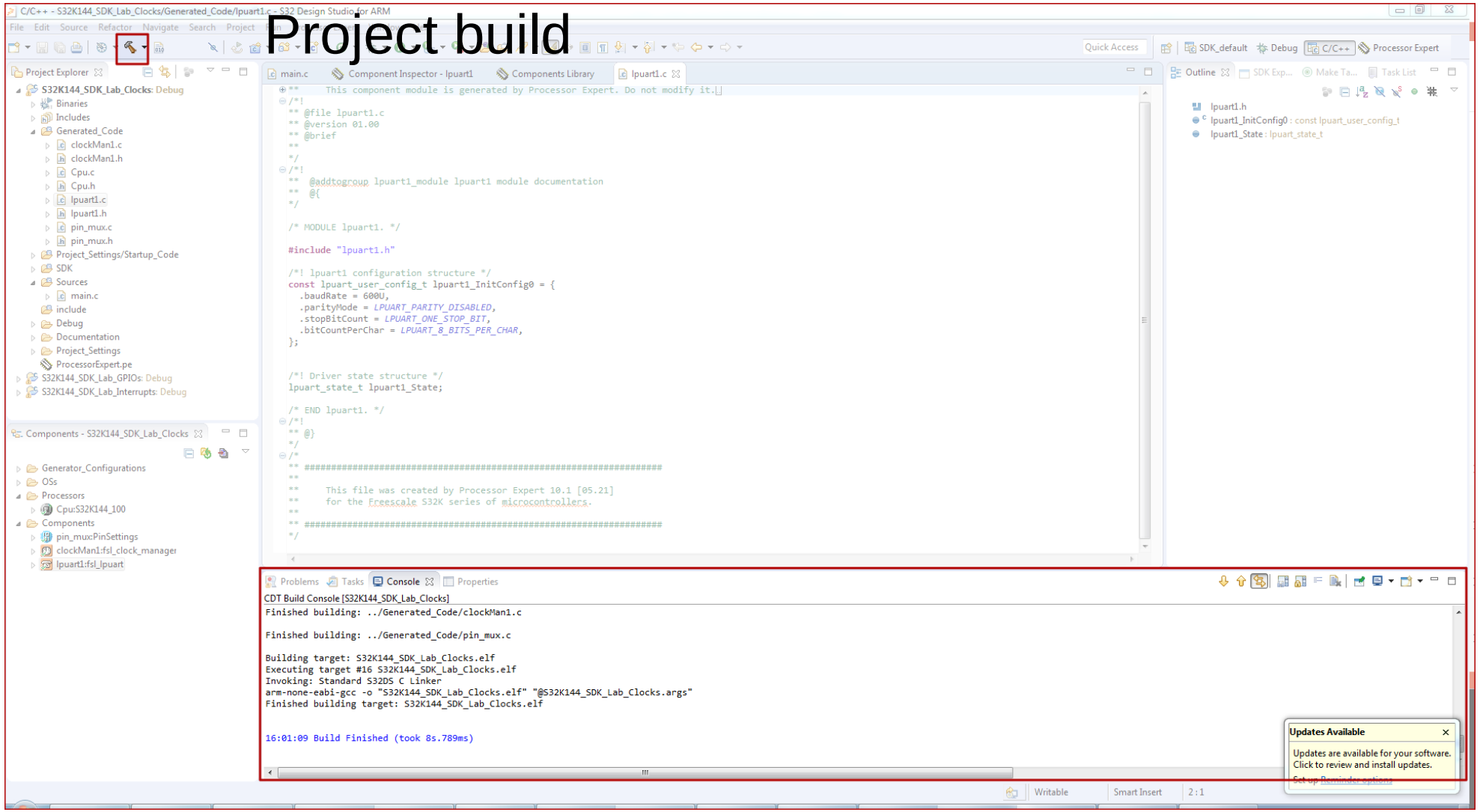
# S32 Design Studio – graphical configuration environment

## Code generation



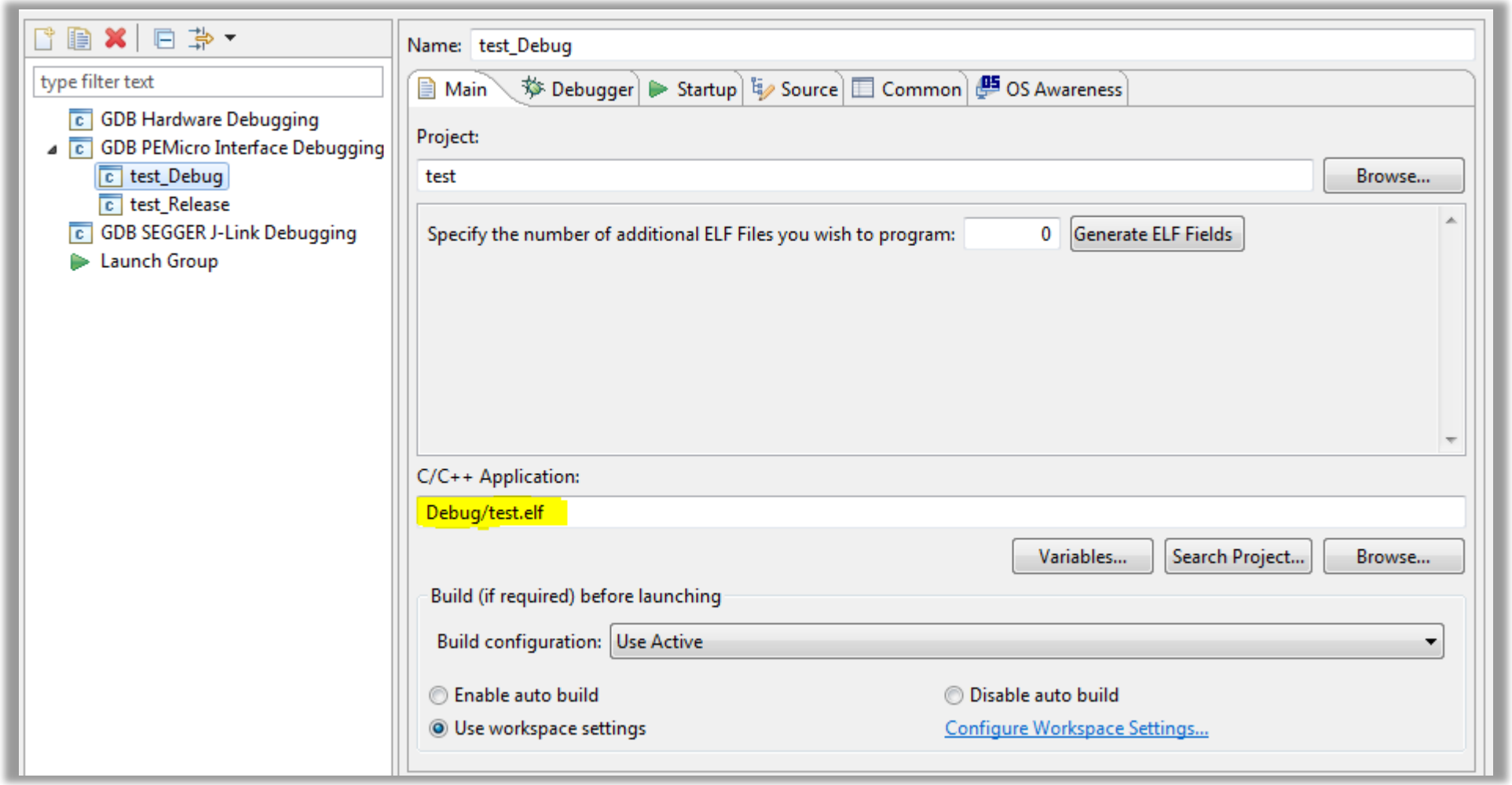
# S32 Design Studio – graphical configuration environment

## Project build



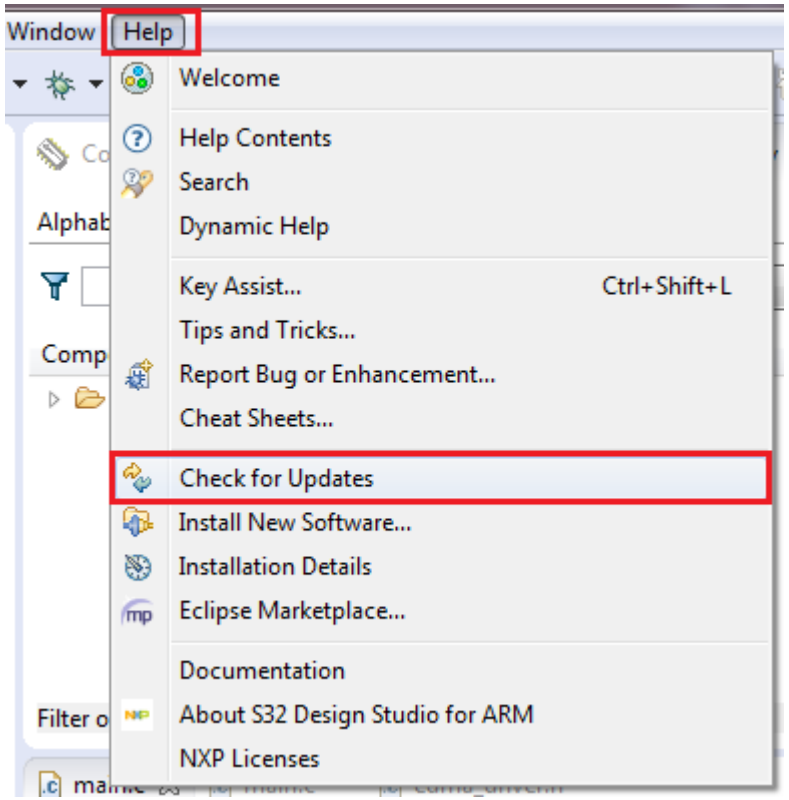
# S32 Design Studio – deploying the application

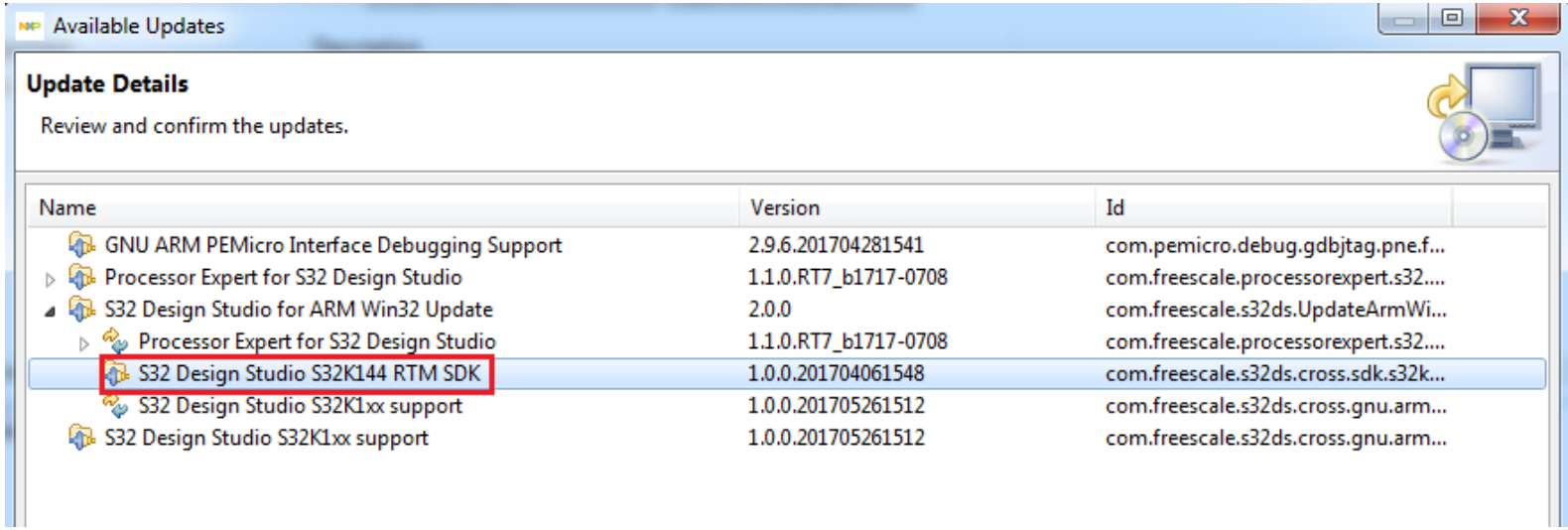
## Target debug



# S32 Design Studio – SDK Update

Newer SDK versions are provided as update sites for DS



# 02.

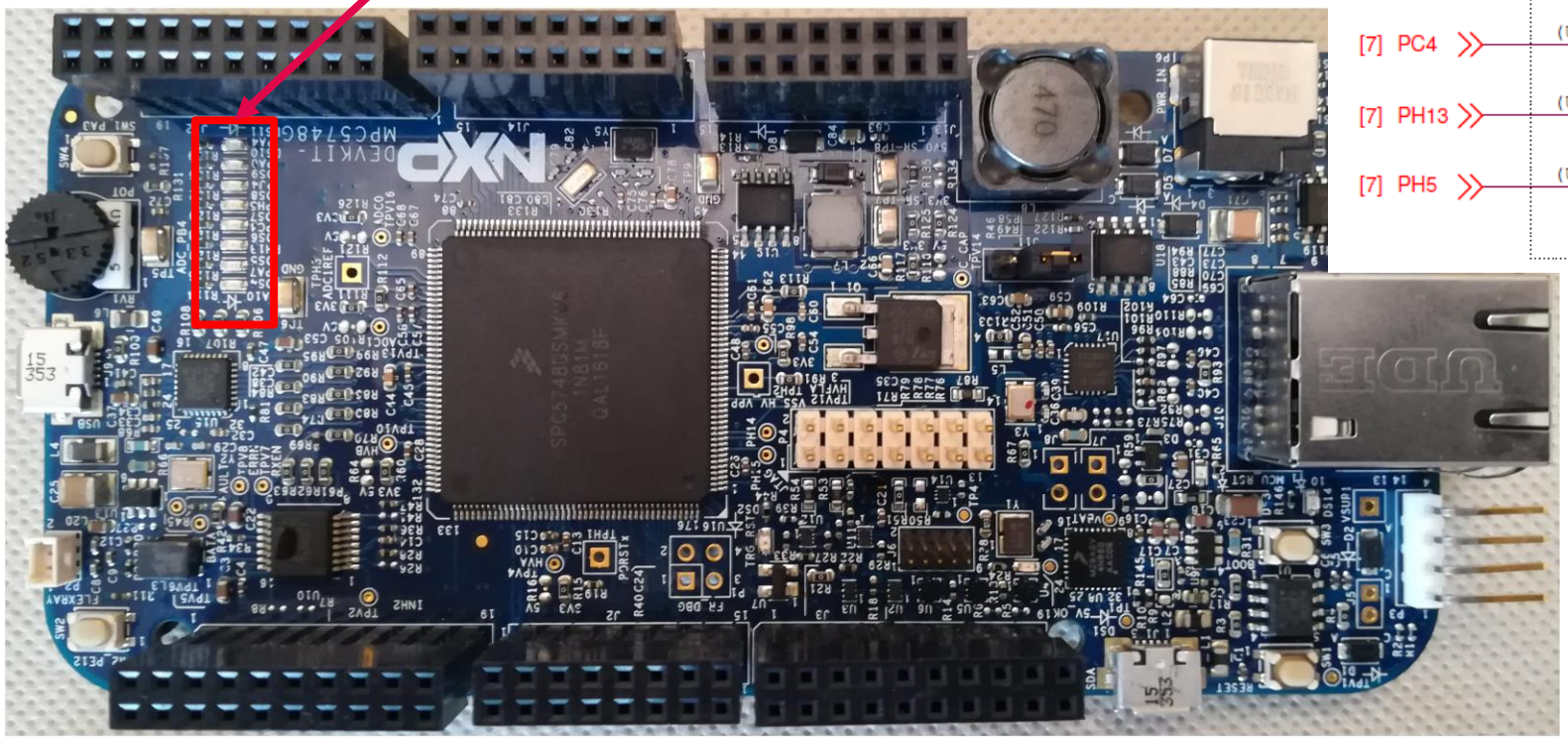
# Hands-on – Blinking LED

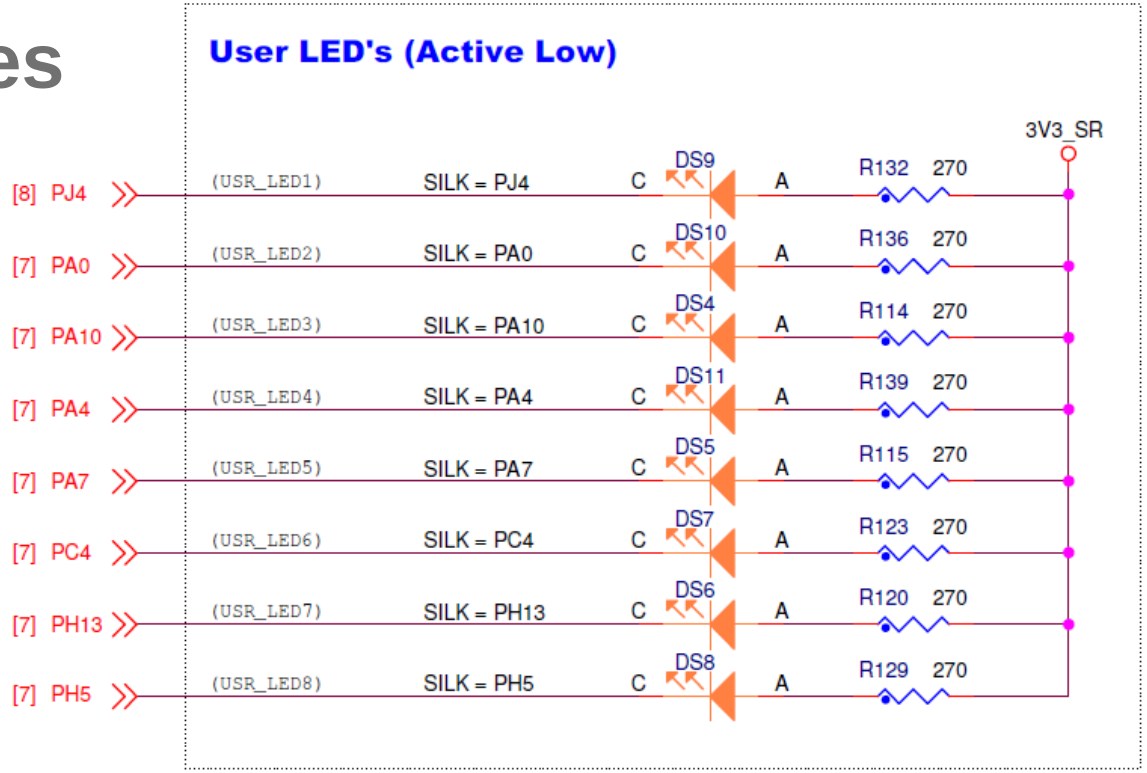
# Hands-on – Blinking LED: Objective

- 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

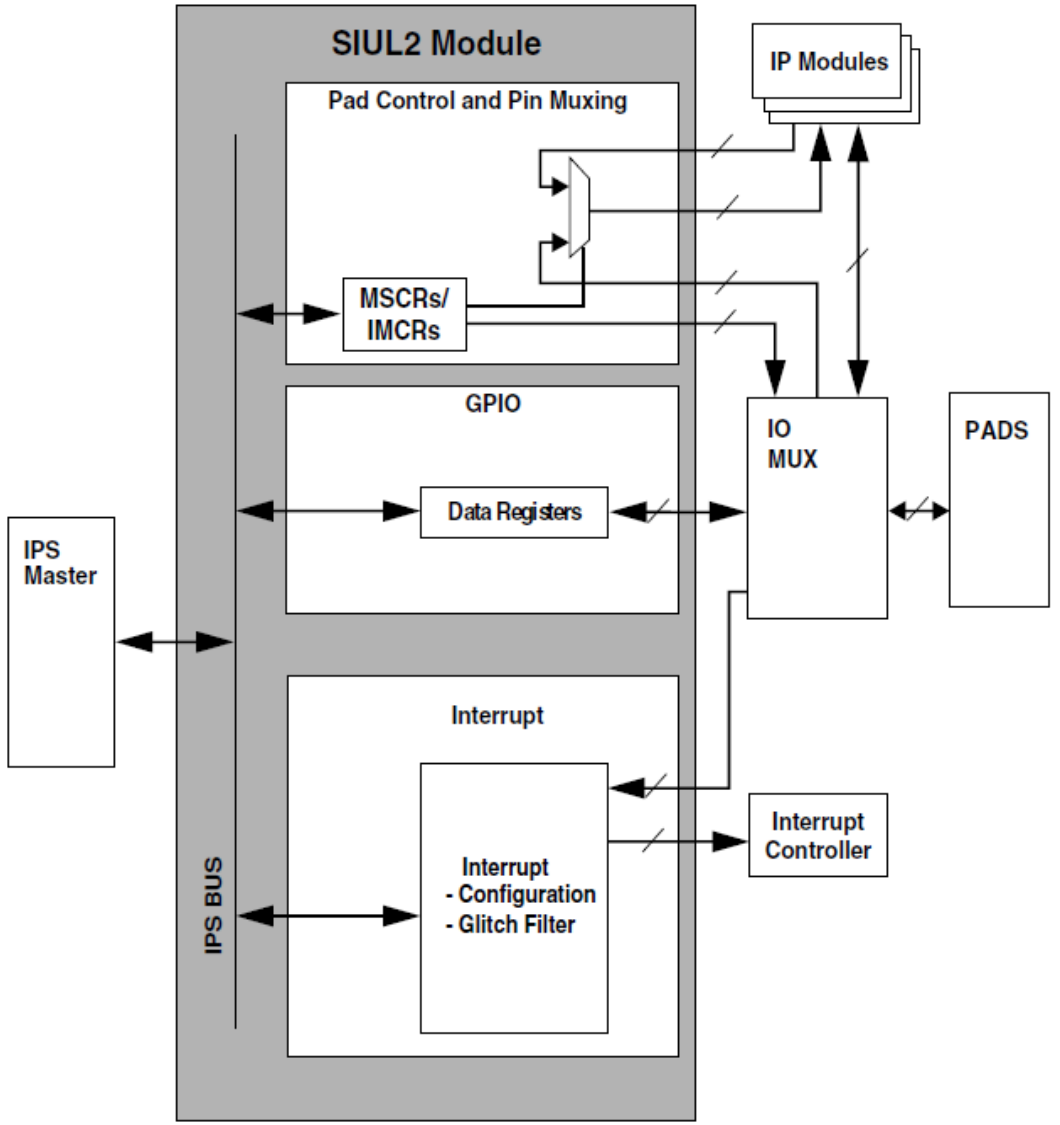
- Resources to be used:
  - on-board user LEDs (hardwired to GPIOs)





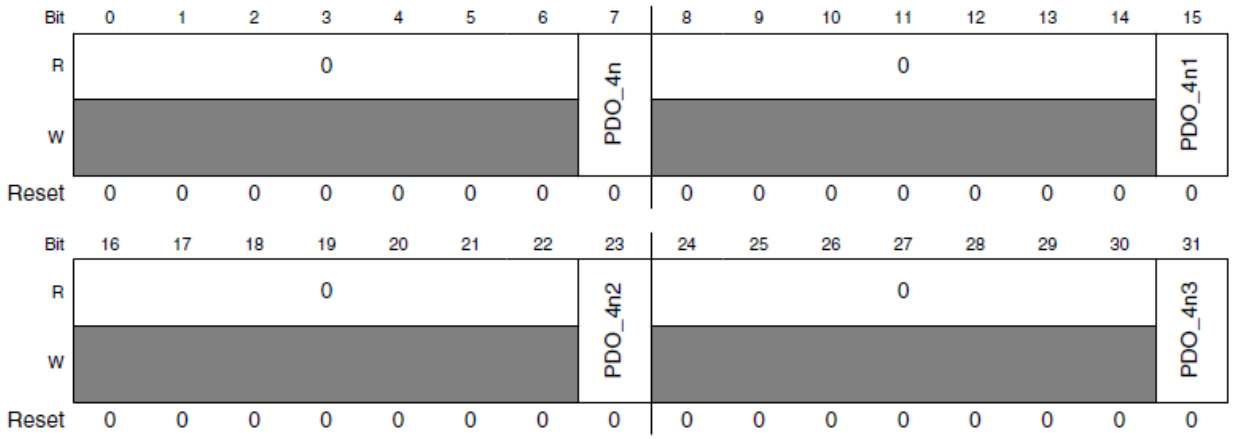
# Hands-on – Blinking LED: Theory

- 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

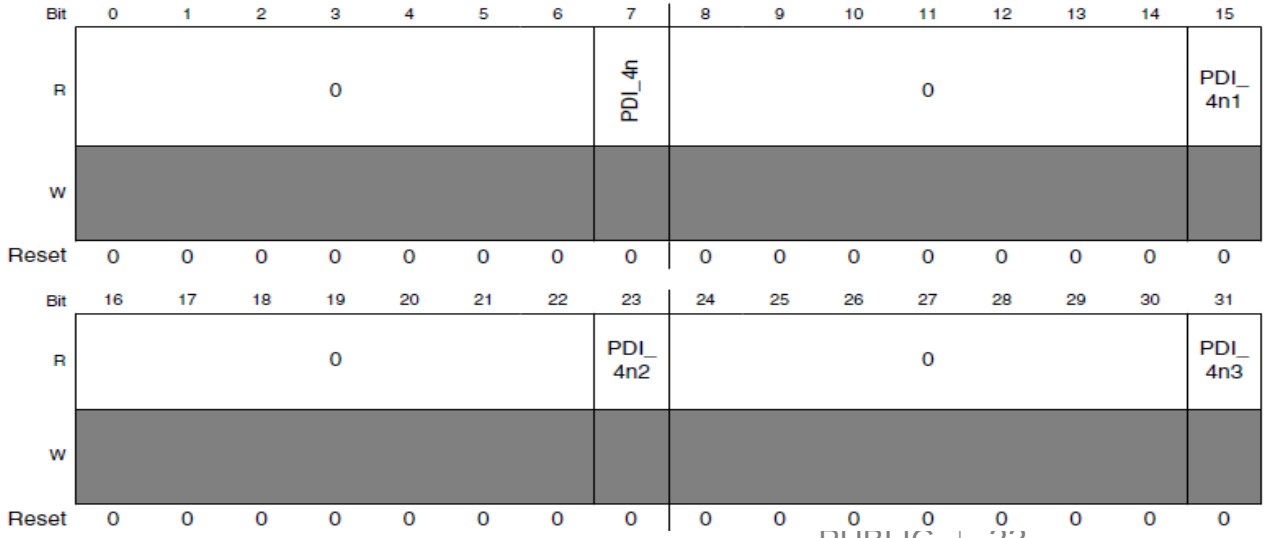


# Hands-on – Blinking LED: Theory

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

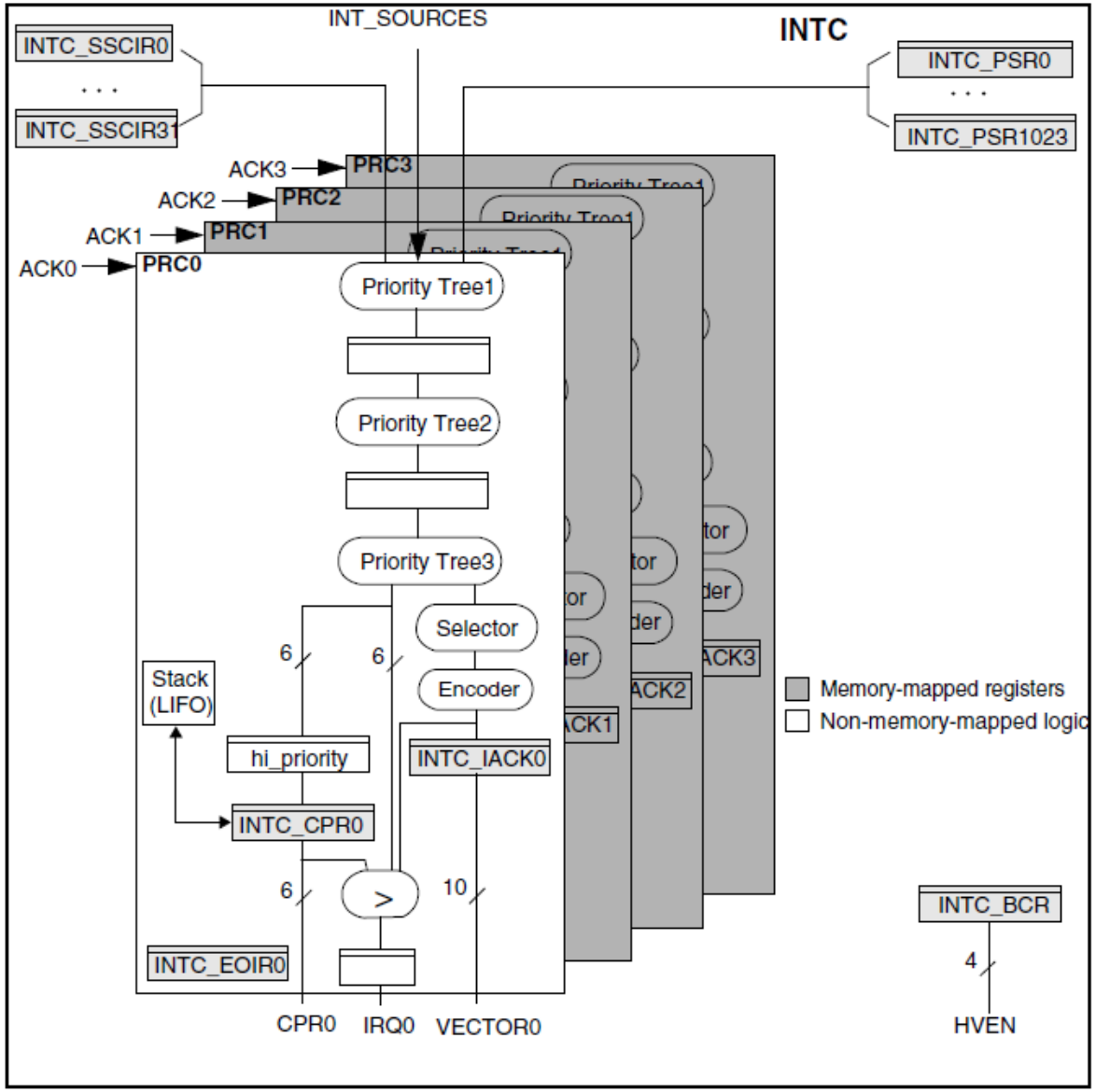



  - GPIO Pad Data Input Register:



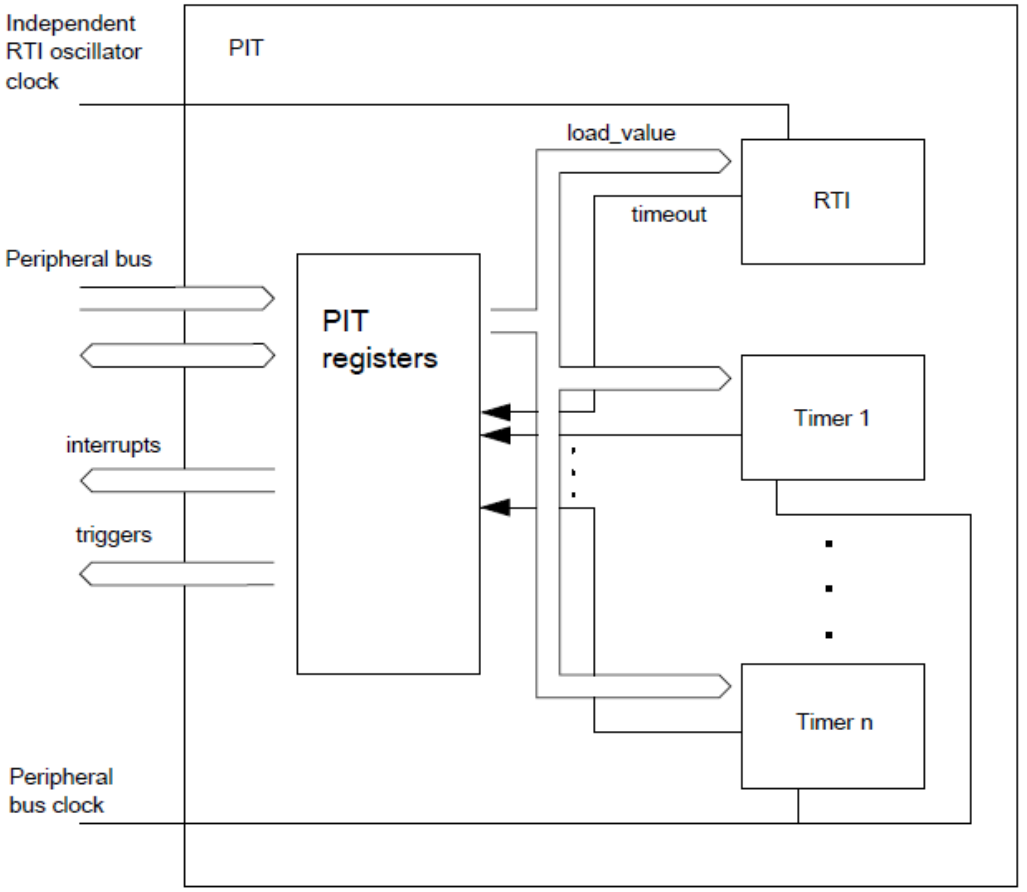
# Hands-on – Blinking LED: Theory

- 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



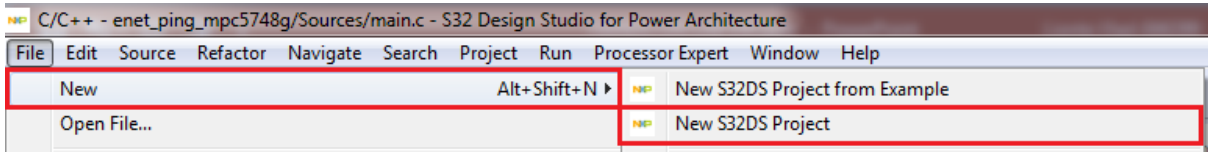
# Hands-on – Blinking LED: Theory

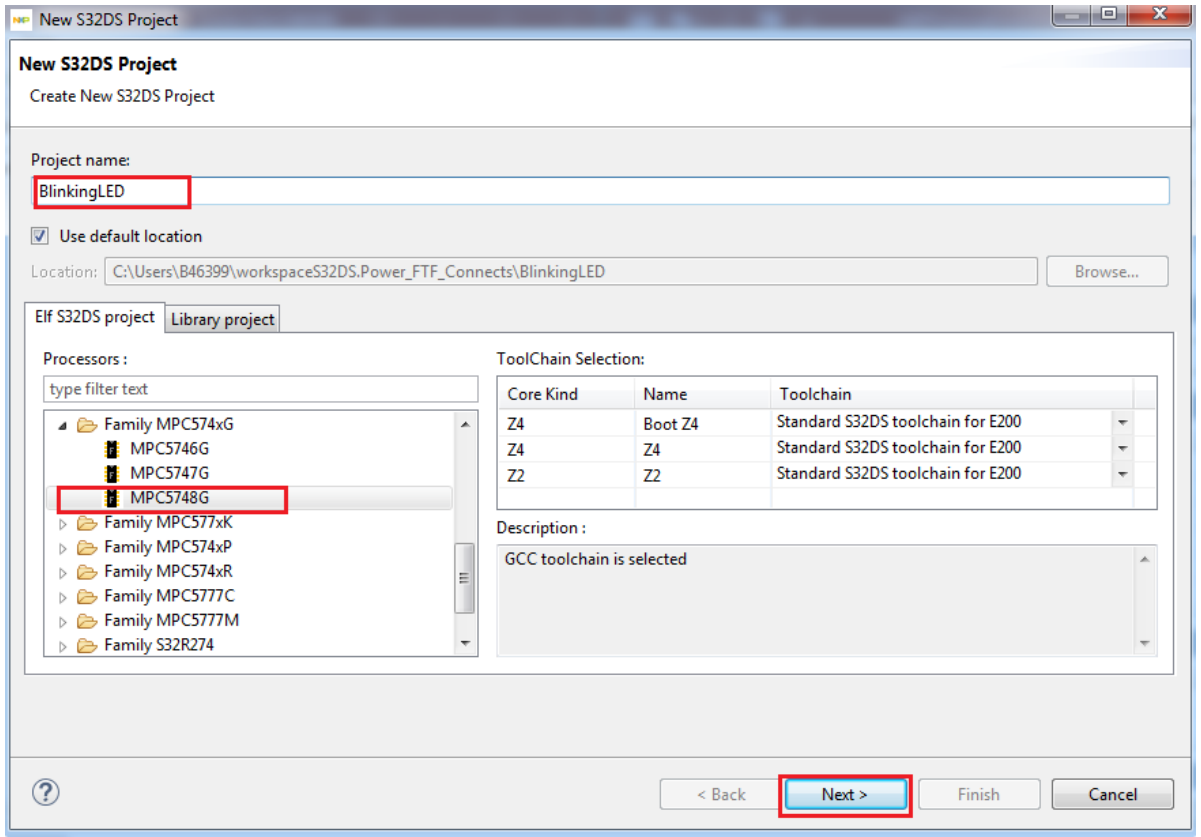
- Periodic Interrupt Timer:
  - One RTI (Real-Time Interrupt) timer to wakeup the CPU in stop mode
  - 16 timer channels
  - Ability of timers to generate trigger pulses
  - Ability of timers to generate interrupts
  - Maskable interrupts
  - Option to raise RTI interrupt, even when the bus clock is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



# Hands-on – Blinking LED: New Project

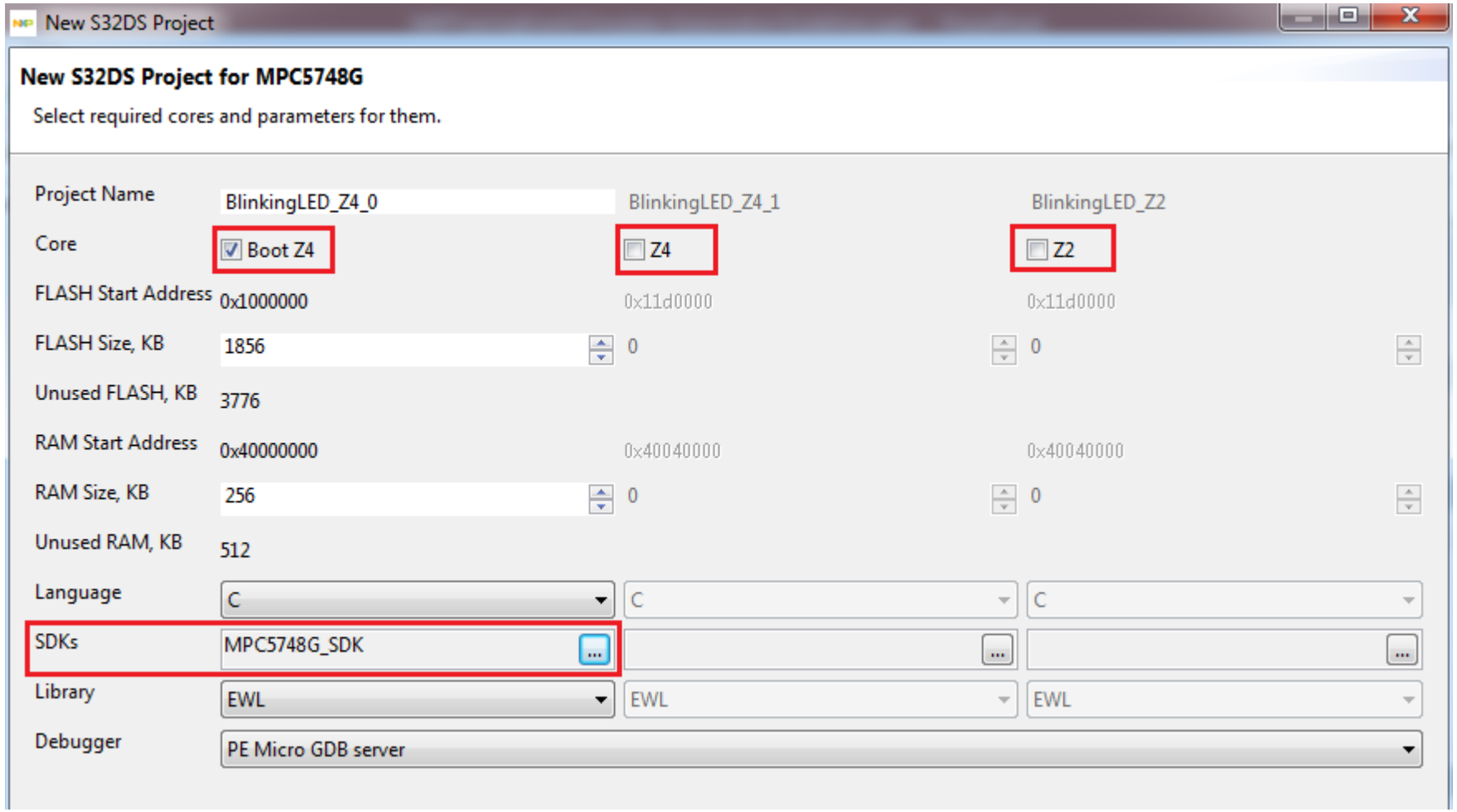
- Create a new S32DS project for MPC5748G:



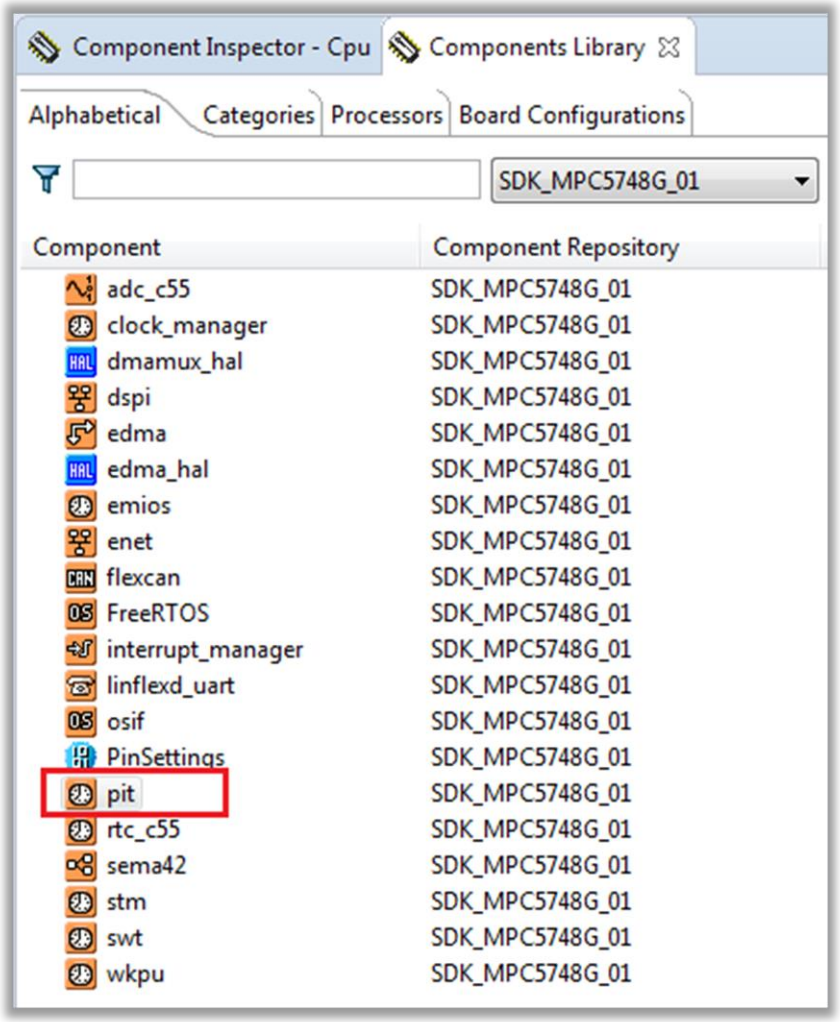
# Hands-on – Blinking LED: New Project

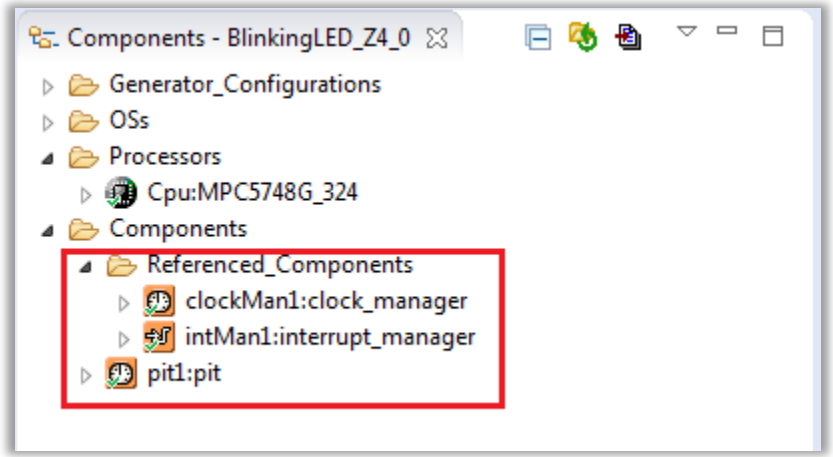
- Check only the ‘Boot z4’ core (application will be running on the boot core)
- Select SDK support for the new project



# Hands-on – Blinking LED: Configuration

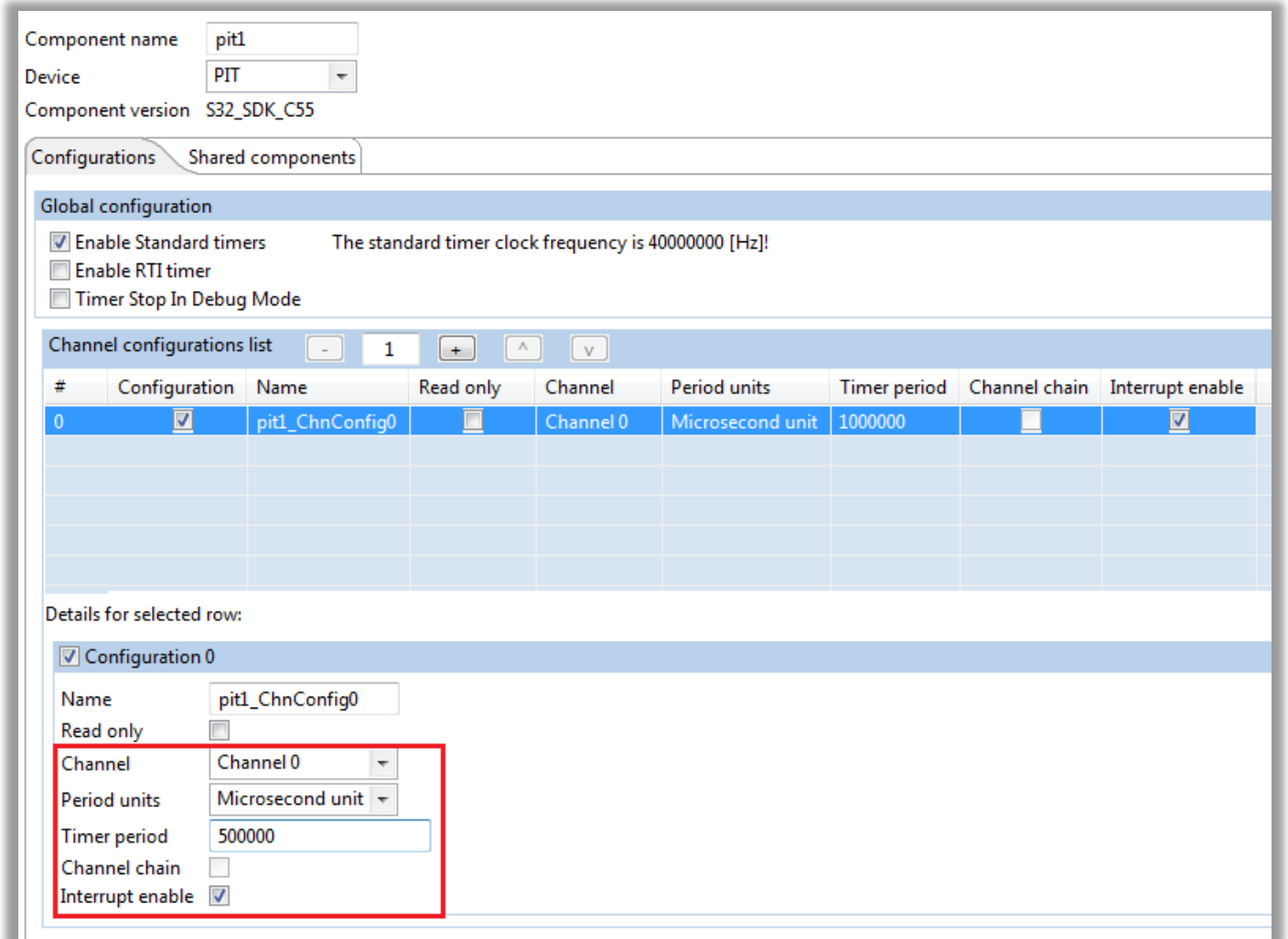
- From ‘Components Library’ view, double-click ‘pit’ component to add it the project
- ‘clock_manager’ & ‘interrupt_manager’ dependencies will be automatically added



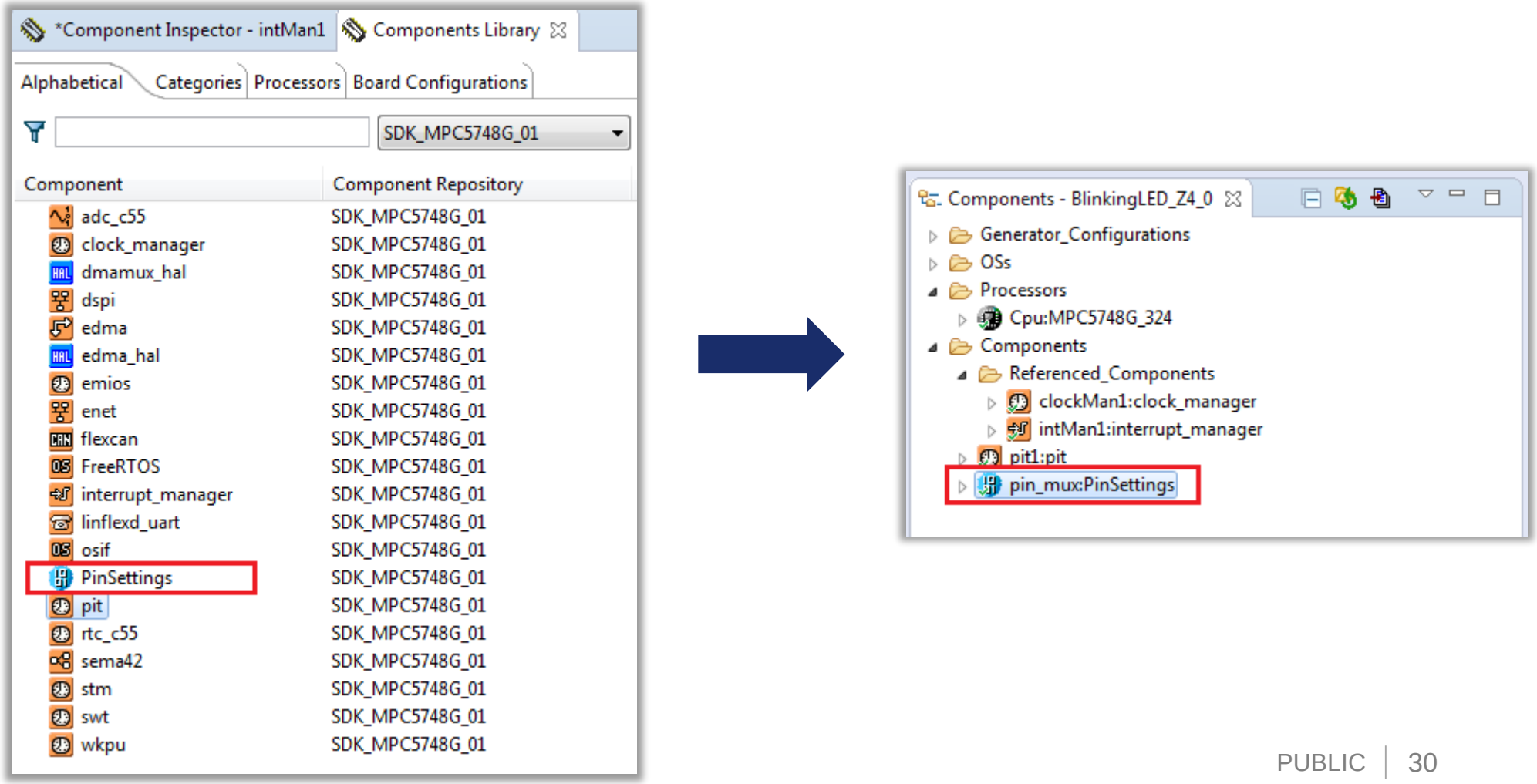
# Hands-on – Blinking LED: Configuration

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



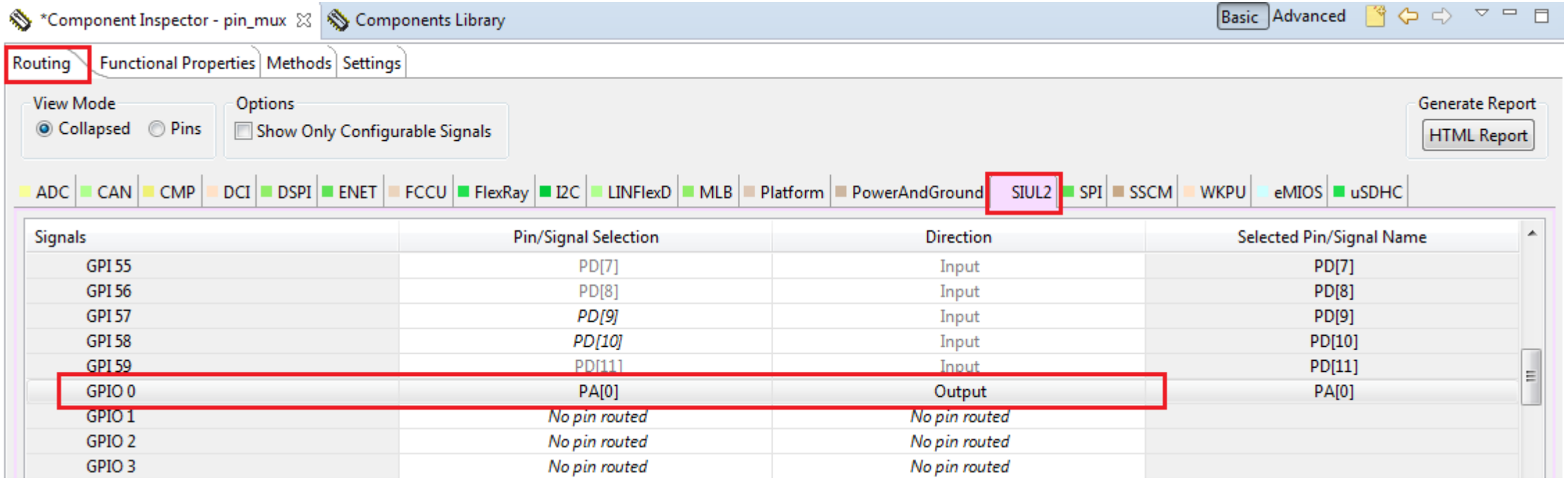
# Hands-on – Blinking LED: Configuration

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



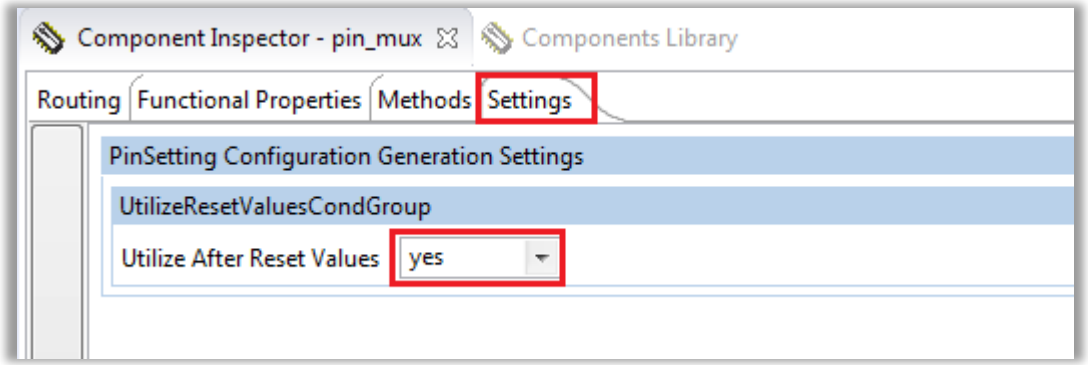
# Hands-on – Blinking LED: Configuration

- 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



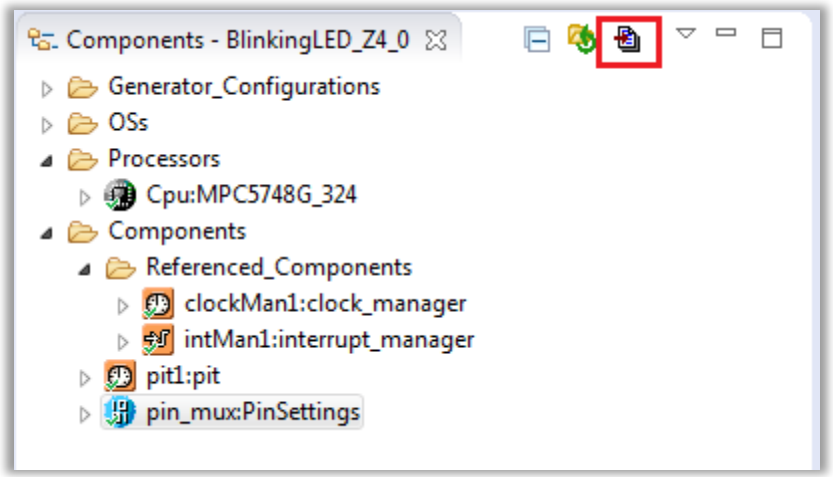
# Hands-on – Blinking LED: Configuration

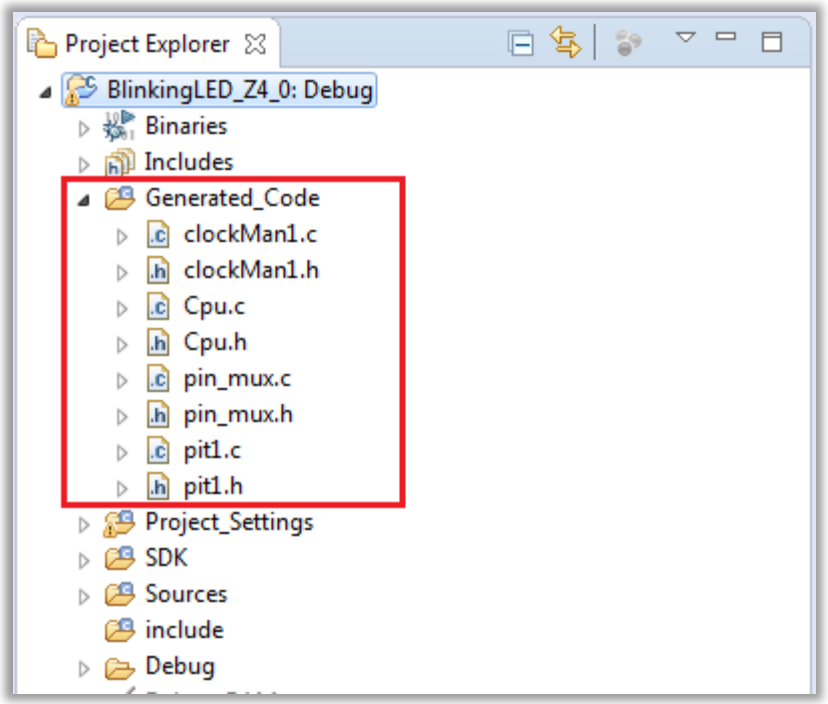
- In ‘pin_mux’ component –> Settings tab – select ‘Utilize After Rest Values – yes’
  - this option will enable code generation only for the modified pins (generating one configuration structure for each pin will result in a very large array residing in target memory)



# Hands-on – Blinking LED: Configuration

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

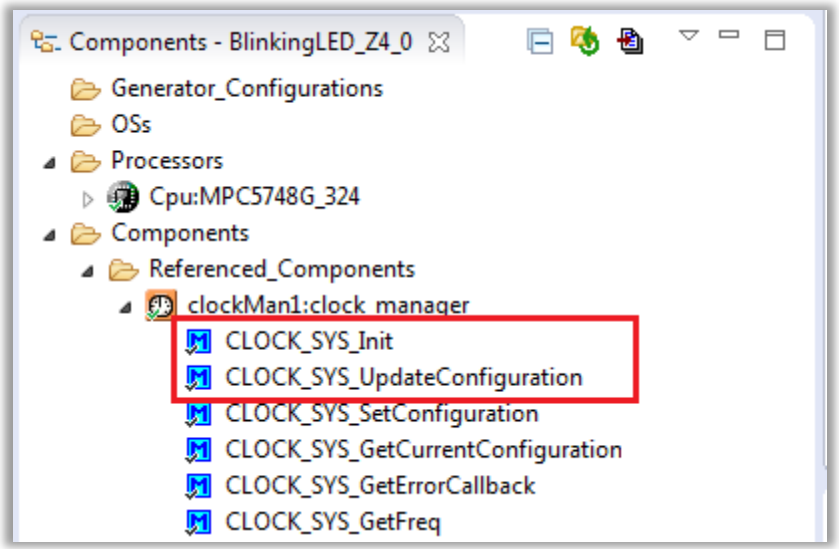


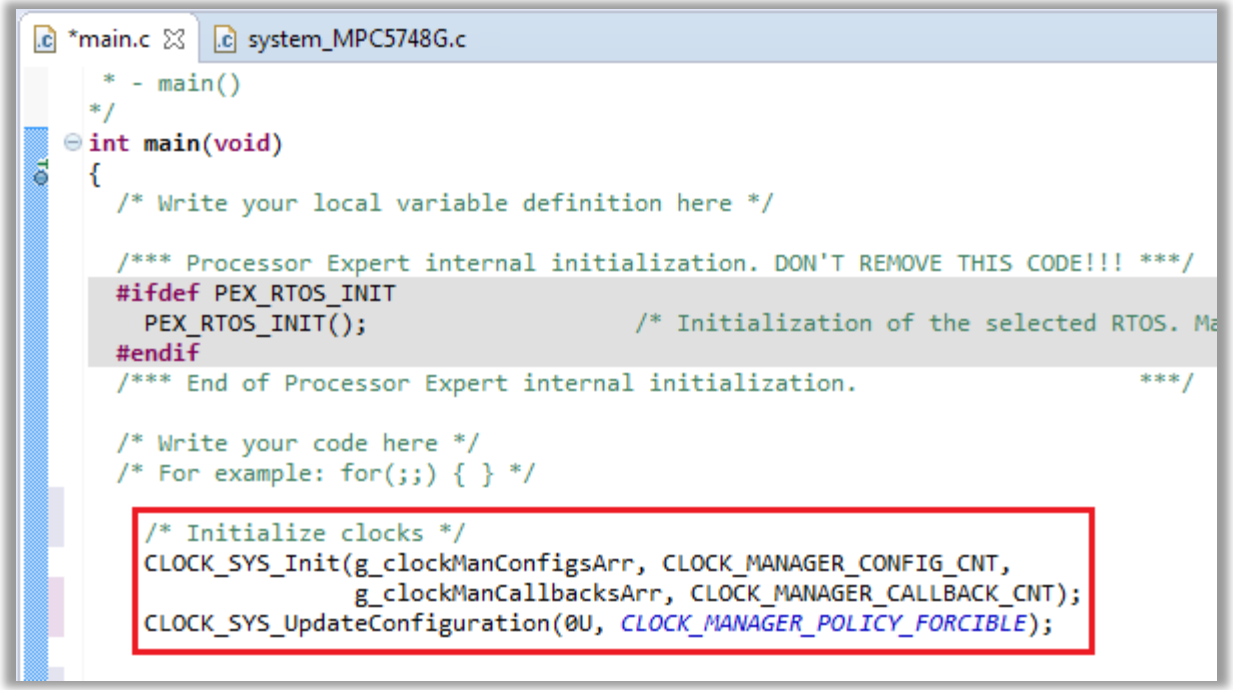



# Hands-on – Blinking LED: Application Code

- 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***

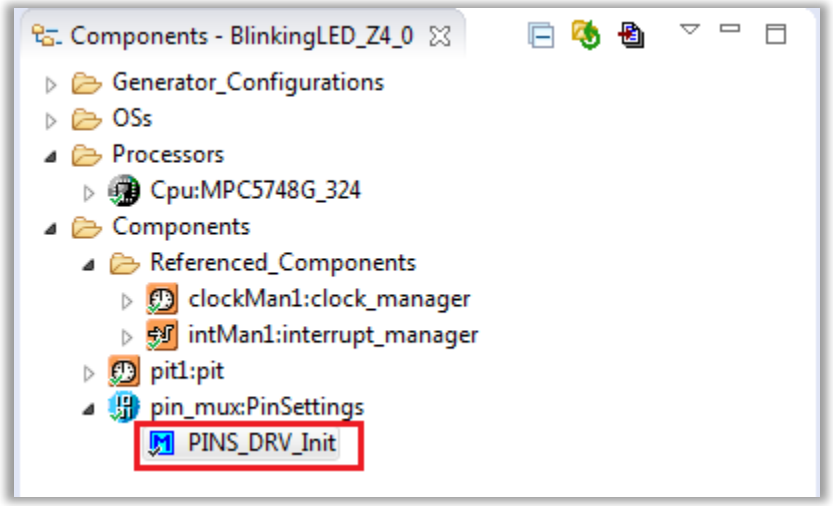


# Hands-on – Blinking LED: Application Code

2) Initialize pins

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

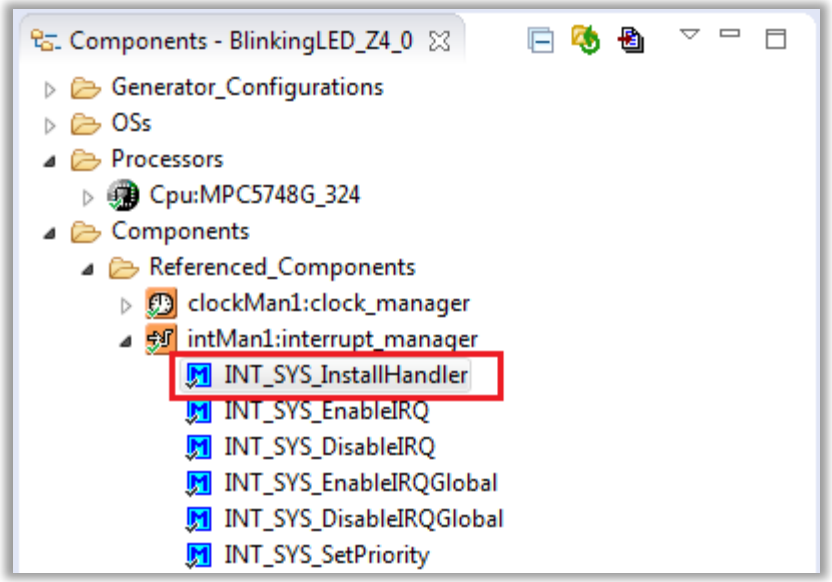



```
int main(void)
{
  /* Write your local variable definition here */

  /*** 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

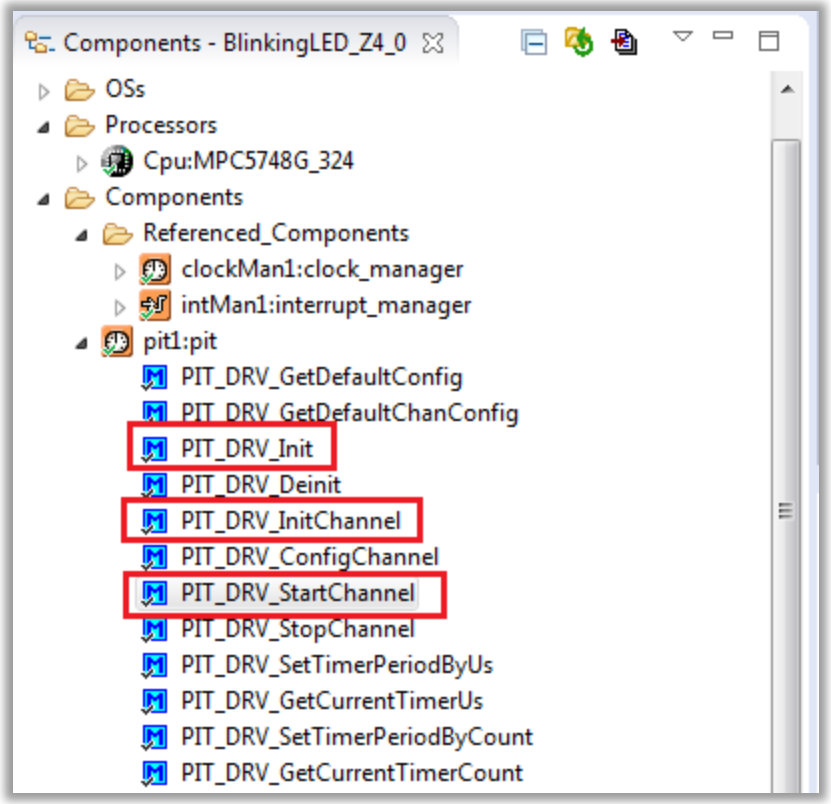
- create a function called ***pitCh0Handler***
- in the function body: clear the interrupt flag + toggle LED

```
/* 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

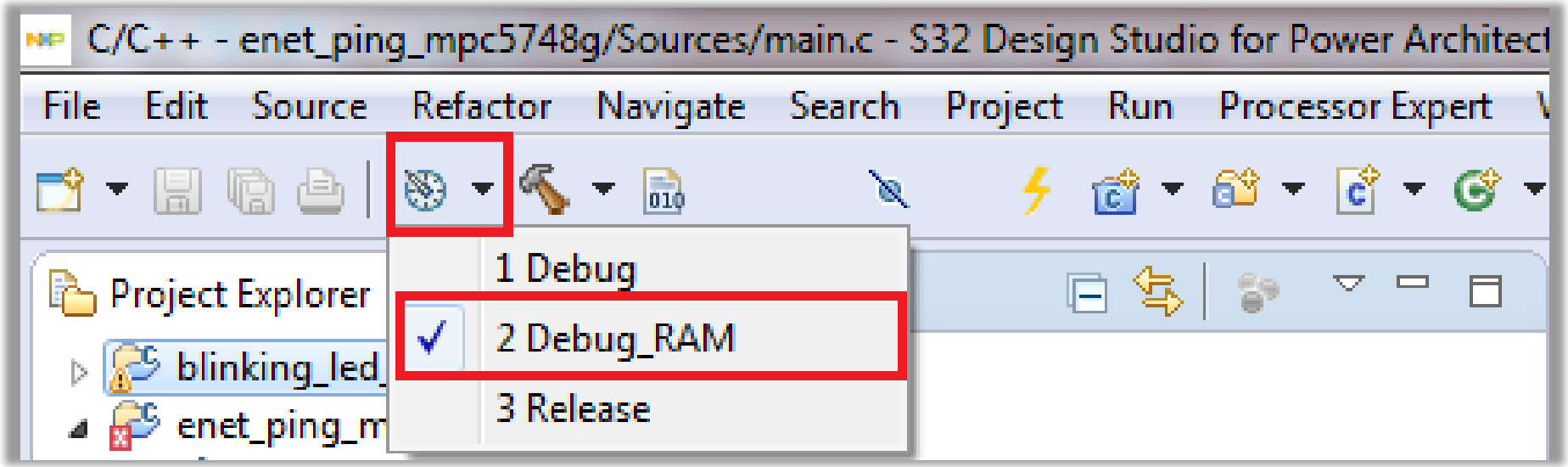
- expand ‘pit’ component -> drag and drop ***PIT_DRV_Init, PIT_DRV_InitChannel*** & ***PIT_DRV_StartChannel*** in ‘main()’
- for the last function, fill in the second parameter: ***0U*** - channel number




```
/* 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_ChnConfig0);
/* Start the counter for channel 0 */
PIT_DRV_StartChannel(INST_PIT1, 0U);
```

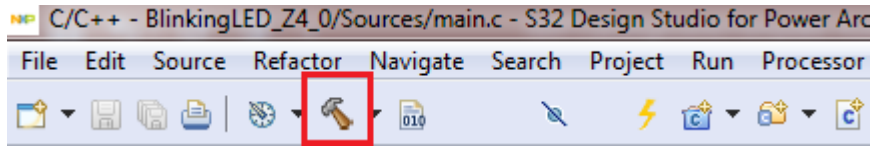
# Hands-on – Blinking LED: Build configuration

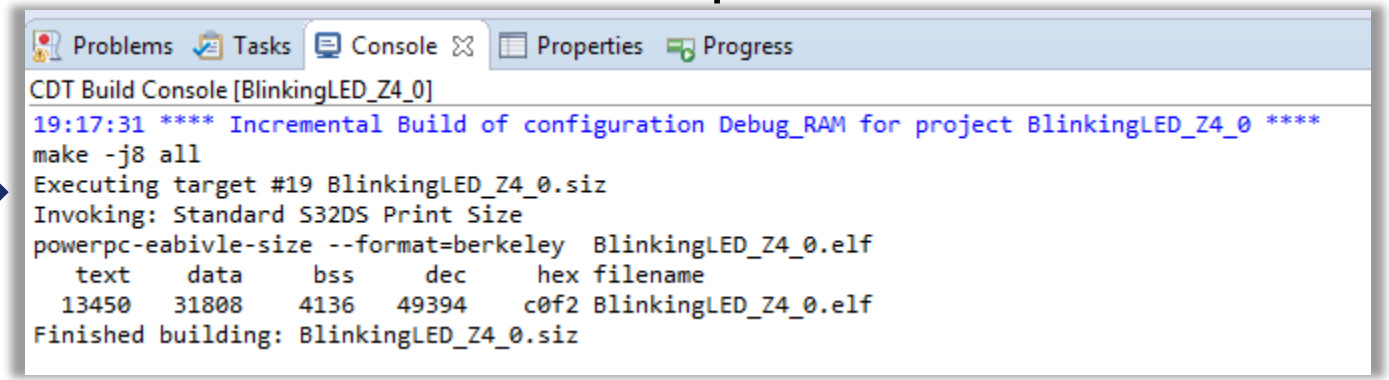
- Click the ‘manage configurations’ button and select Debug_RAM build configuration (more easy to debug)



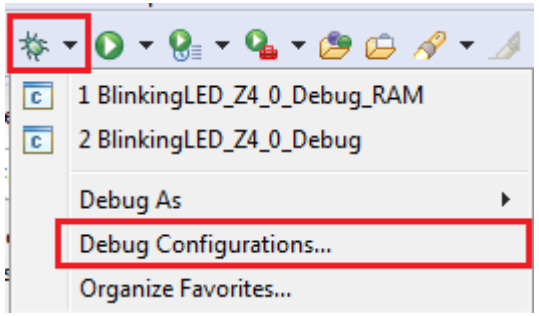
# Hands-on – Blinking LED: Build and Debug

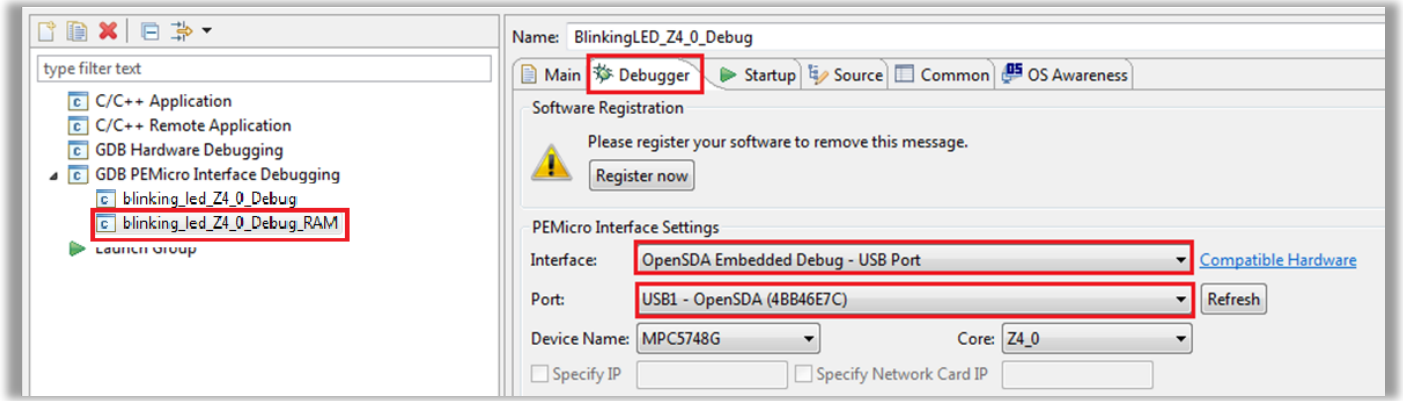
- Click the ‘build project’ button – make sure there are no compilation errors



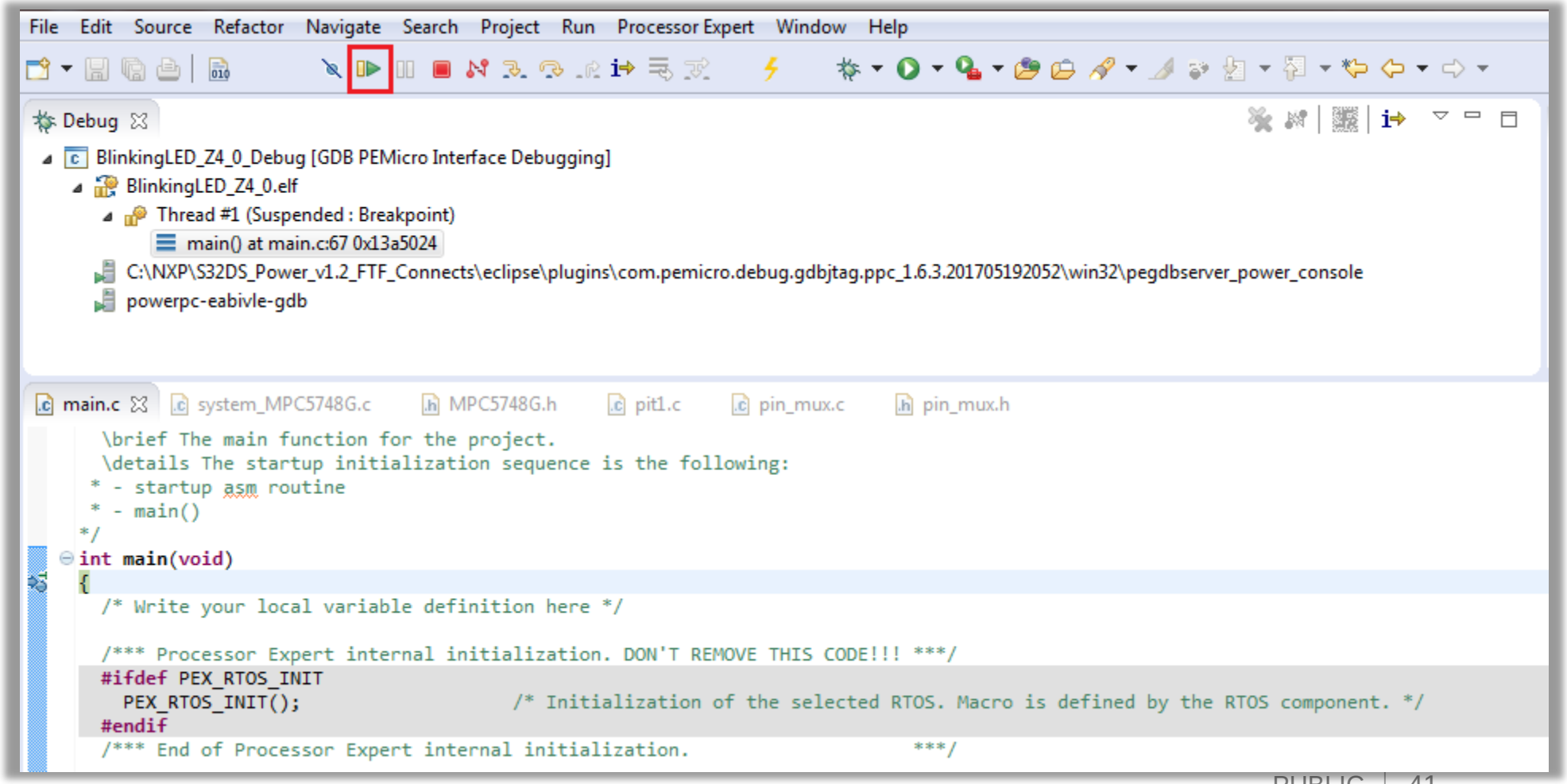

- Select the correct debug configuration and interface to debug the application



# 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

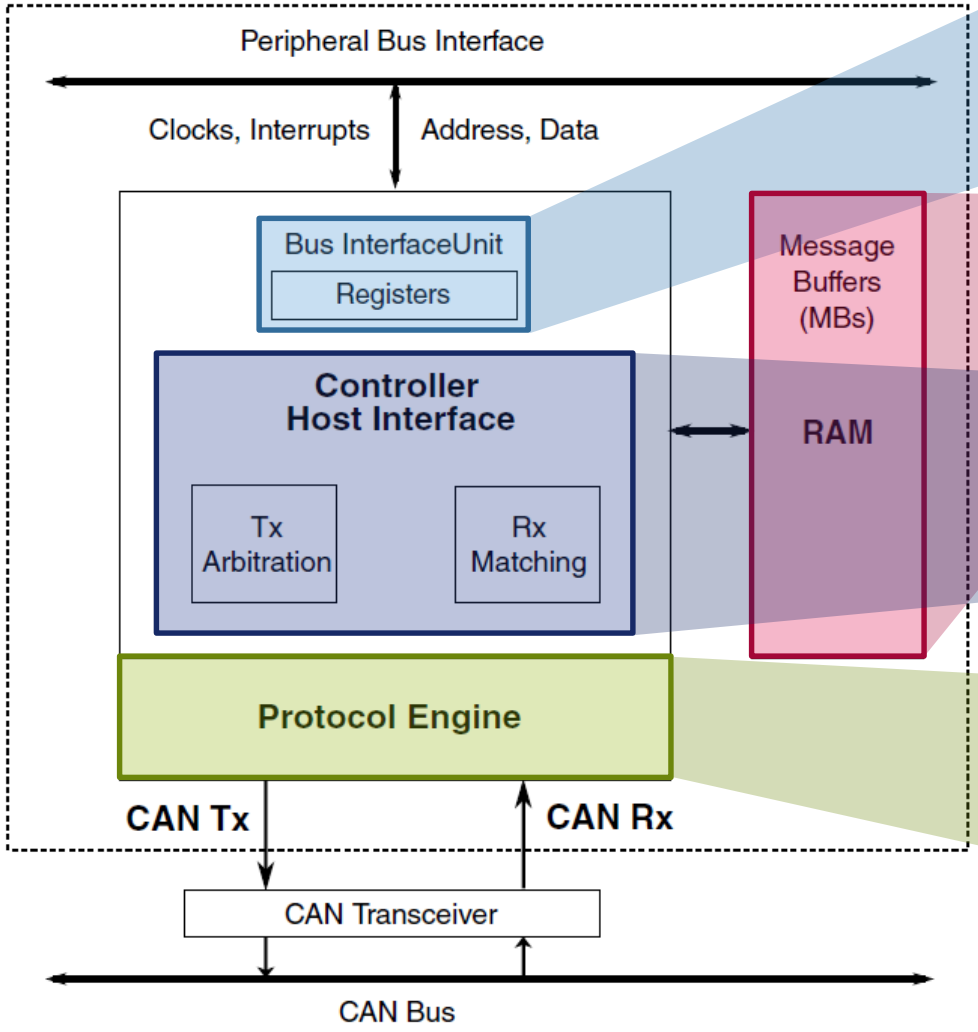
# Hands-on – ENET–CAN: ENET Theory

- 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




Access to and from the internal interface bus (clocks, address and data buses, interrupts, DMA and test signals)

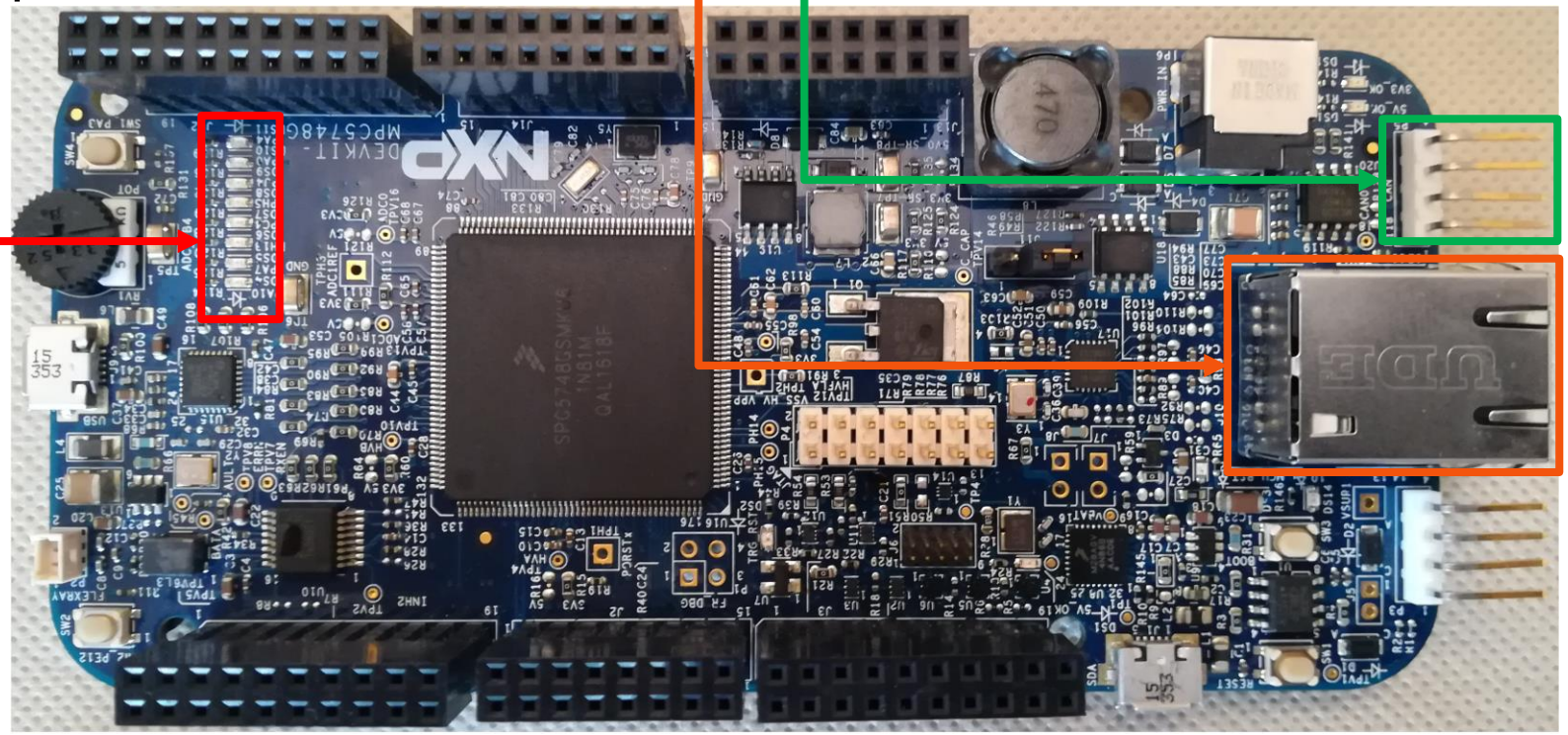
Embedded RAM dedicated to the FlexCAN

Message buffer selection for reception and transmission (arbitration and ID matching algorithms)

Serial communication on the CAN bus (RAM access requests for rx and tx frames, rx messages  validation, error handling)

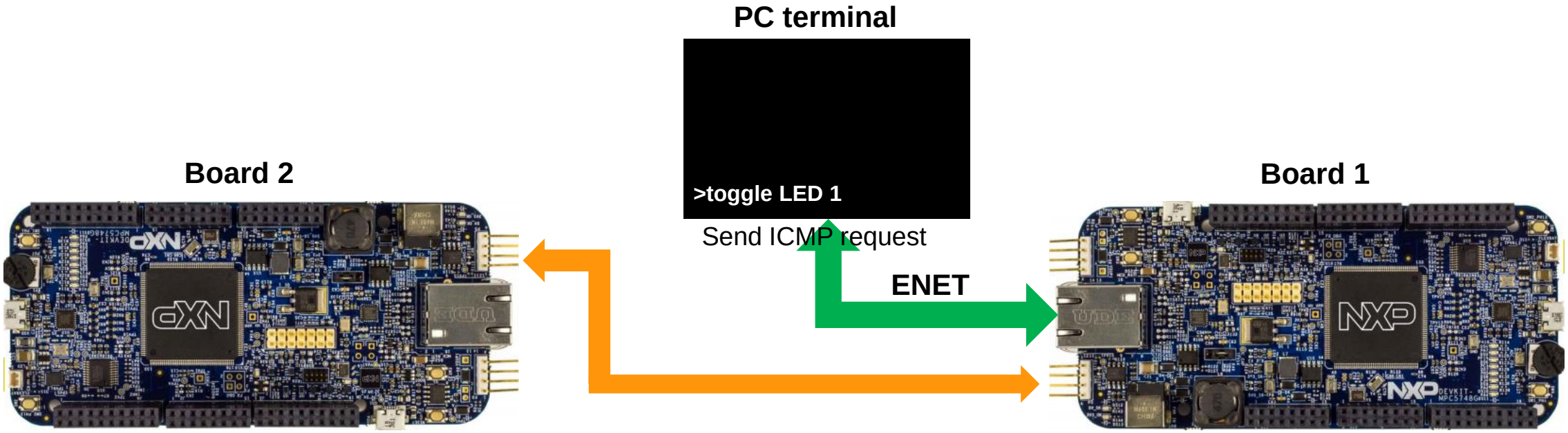
# Hands-on – ENET–CAN: Resources

- 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 receival

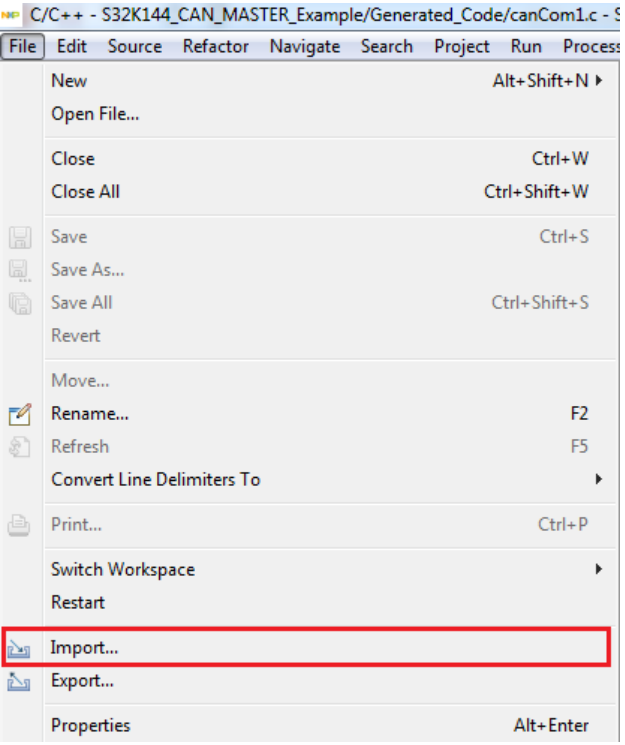


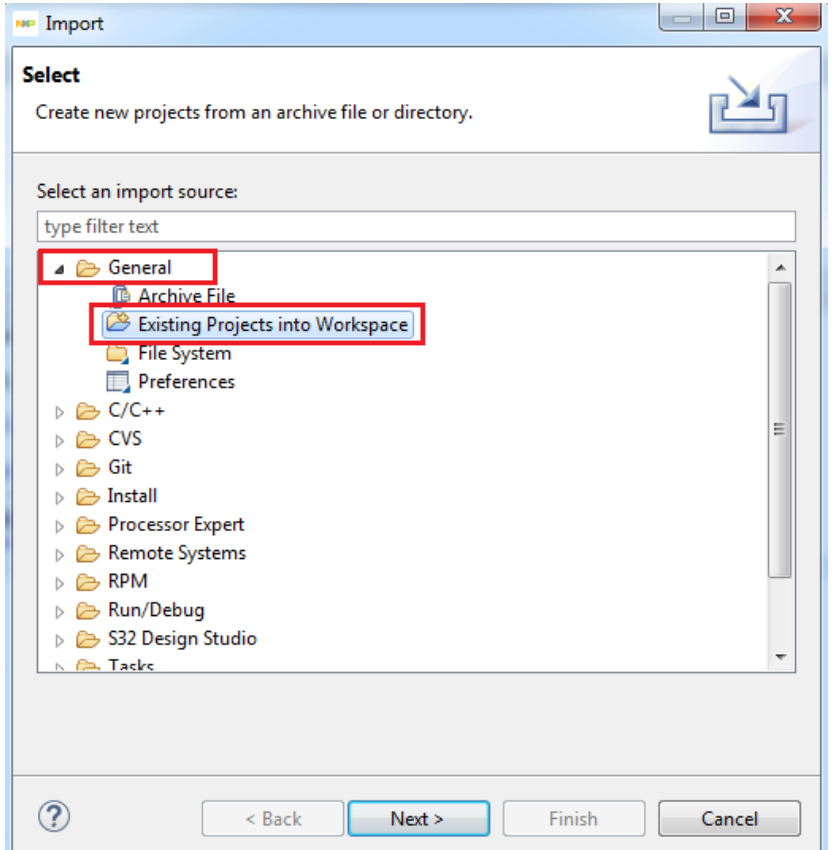
# Hands-on – ENET–CAN: Import Example Project

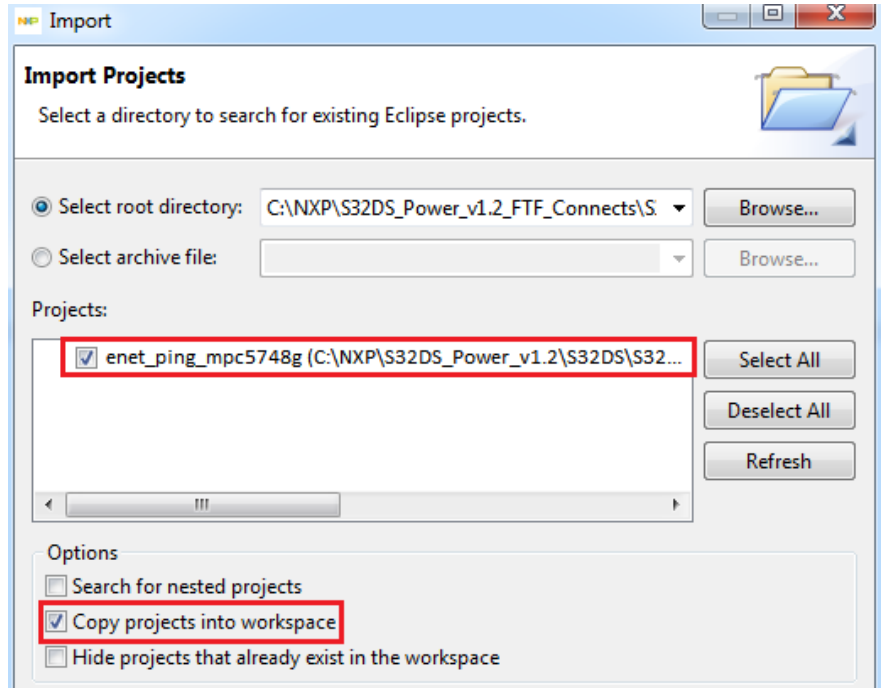
- Import ‘enet_ping’ example provided with the SDK:
  - File->Import->General->Existing Projects into Workspace->Select root directory
  - Select:

{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

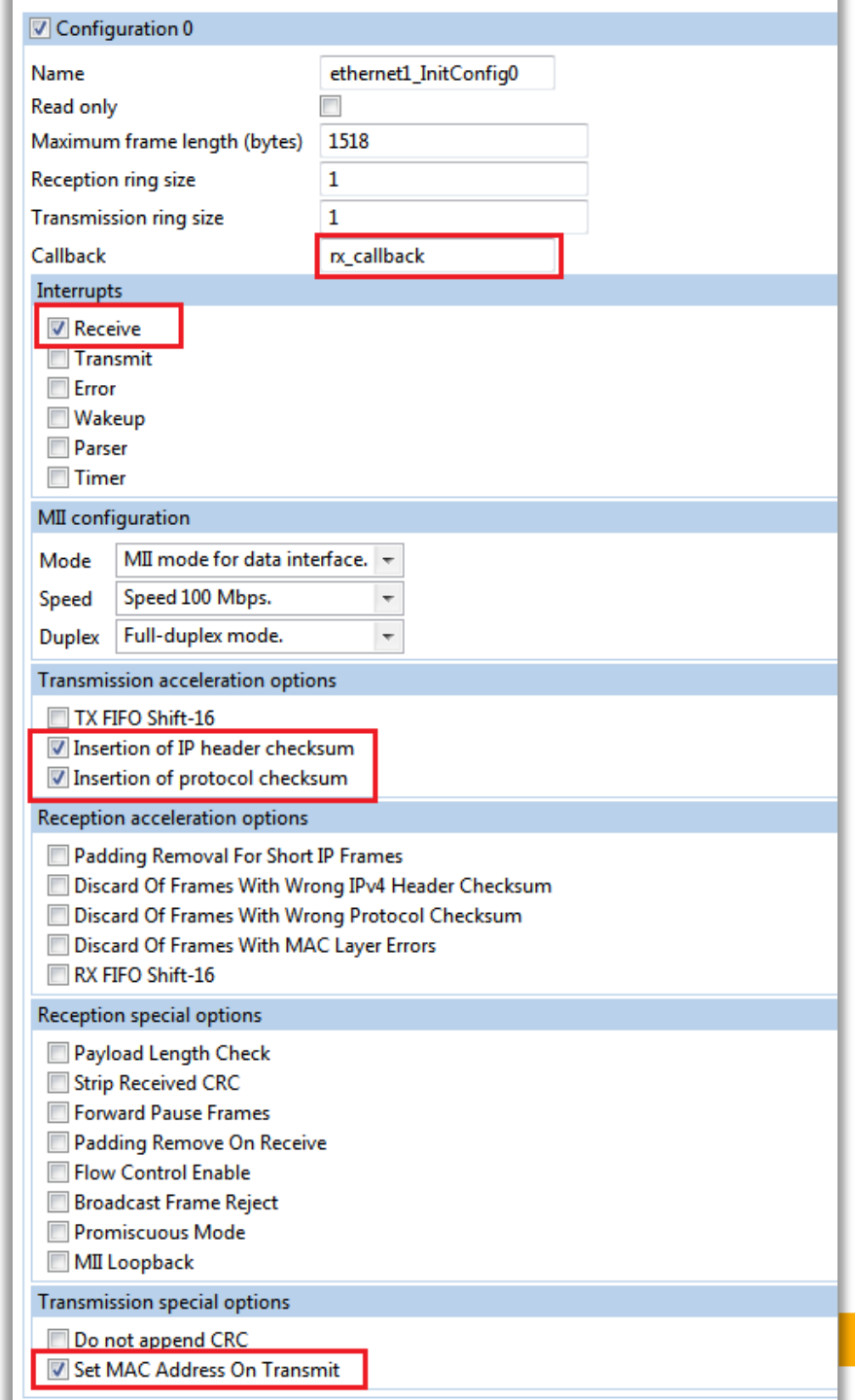


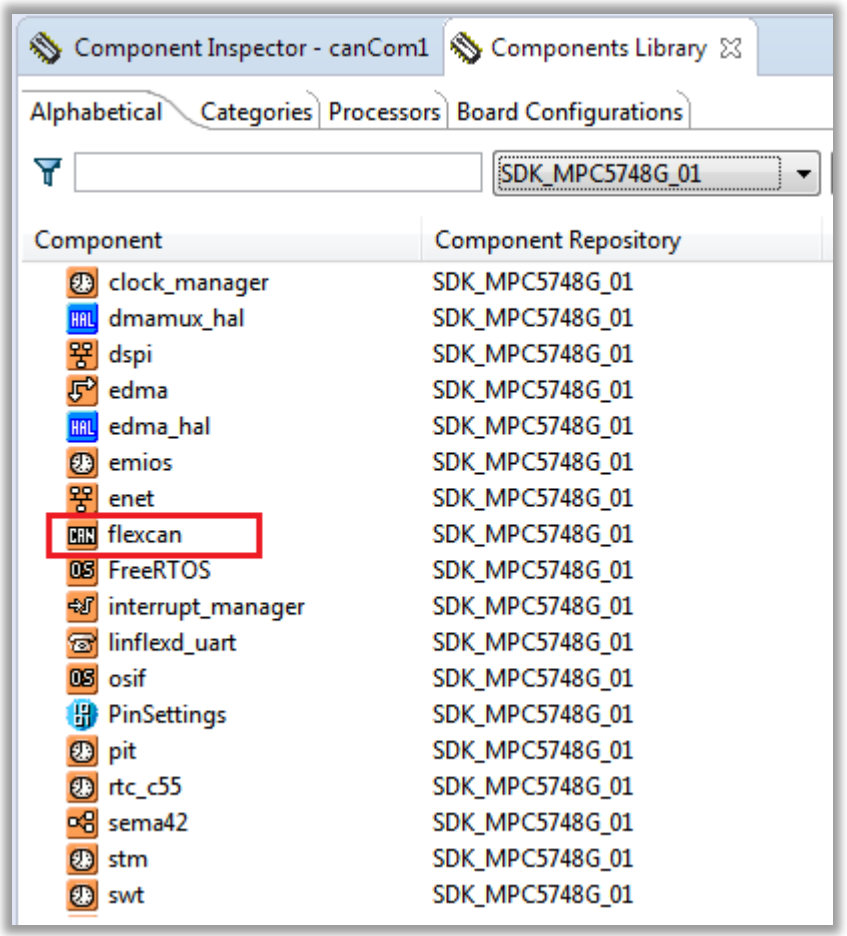
# Hands-on – ENET–CAN: Configuration

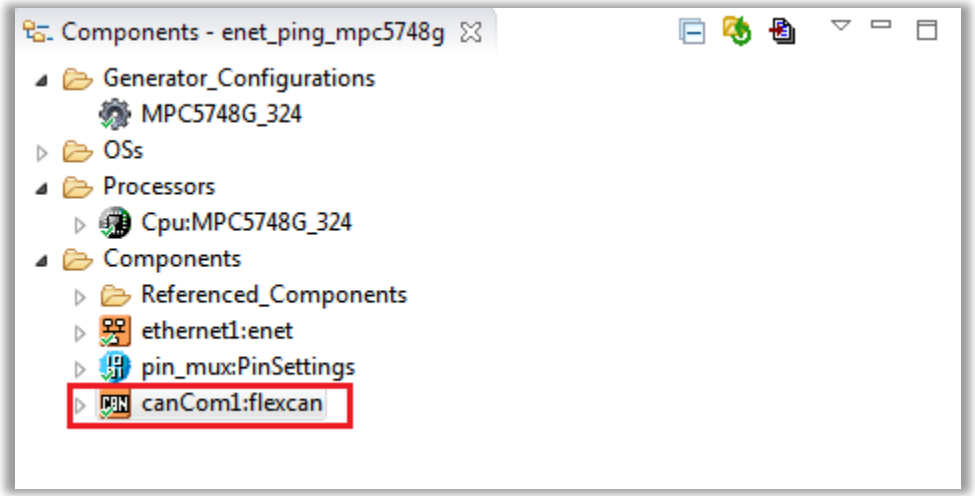
- ENET configuration
  - Receive callback function
  - Receive interrupt enabled
  - Insertion of IP + ICMP checksum
  - Set MAC address for frames



# Hands-on – ENET–CAN: Configuration

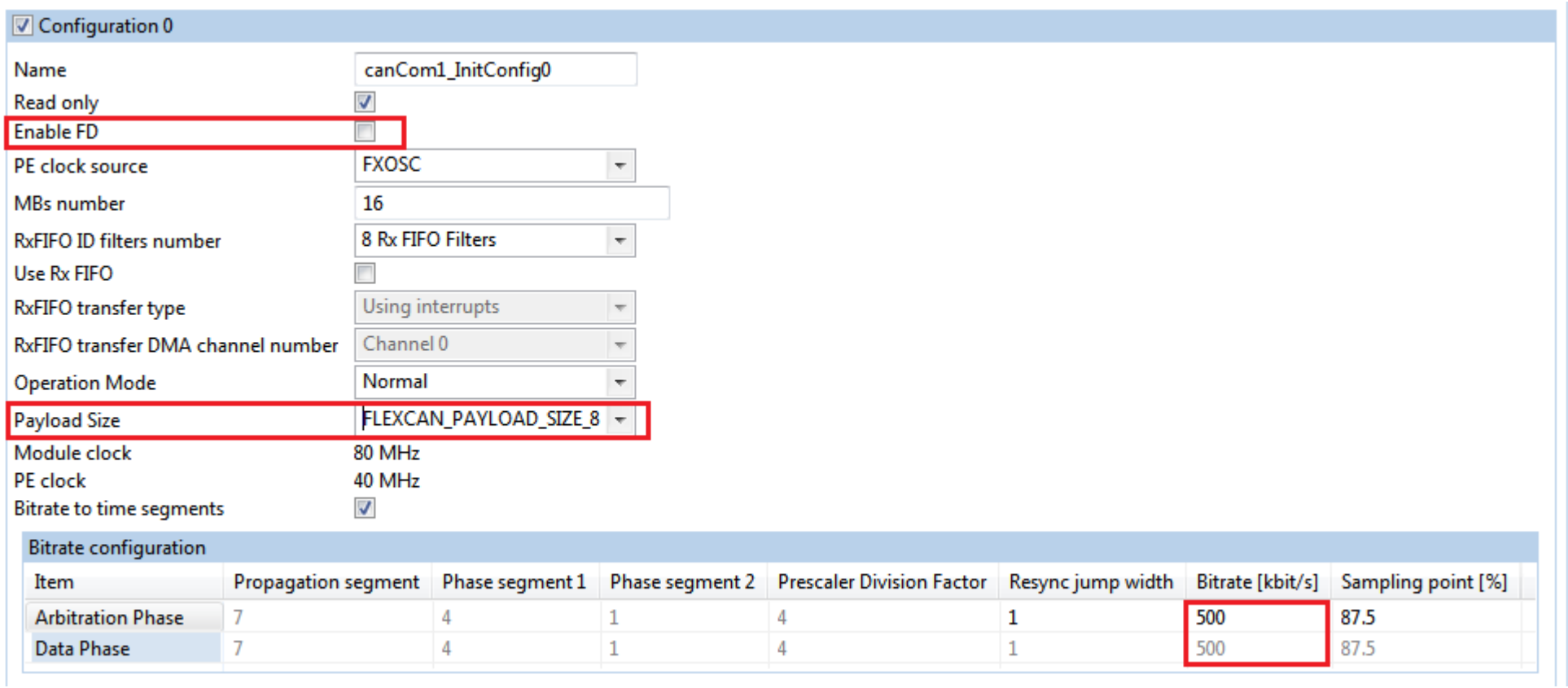
- From ‘Components Library’ view, double-click ‘flexcan’ component to add it the project



# Hands-on – ENET–CAN: Configuration

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



# 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

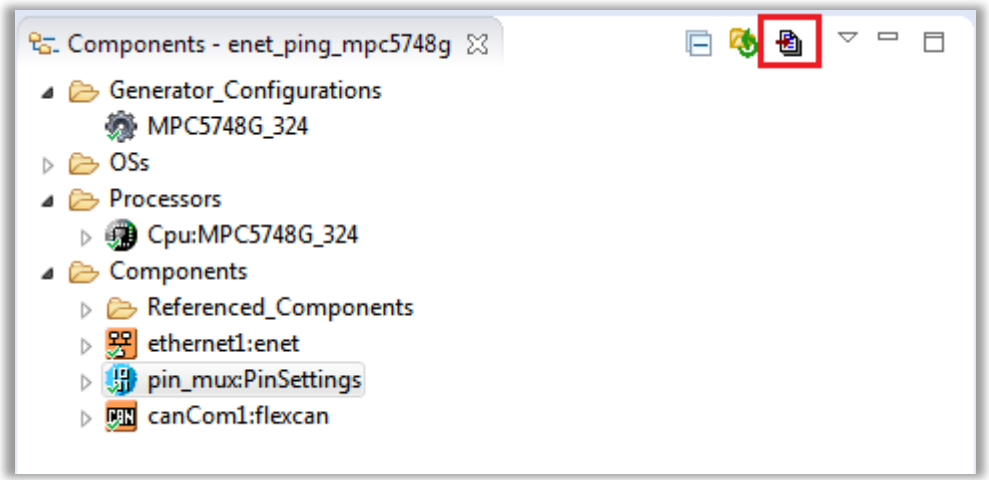
| Signals | Pin/Signal Selection | Direction |
|---|---|---|
| **CAN_0** | | |
| Receiver Input | PB[1] | Input |
| Transmitter Output | PB[0] | Output |
| **CAN_1** | | |

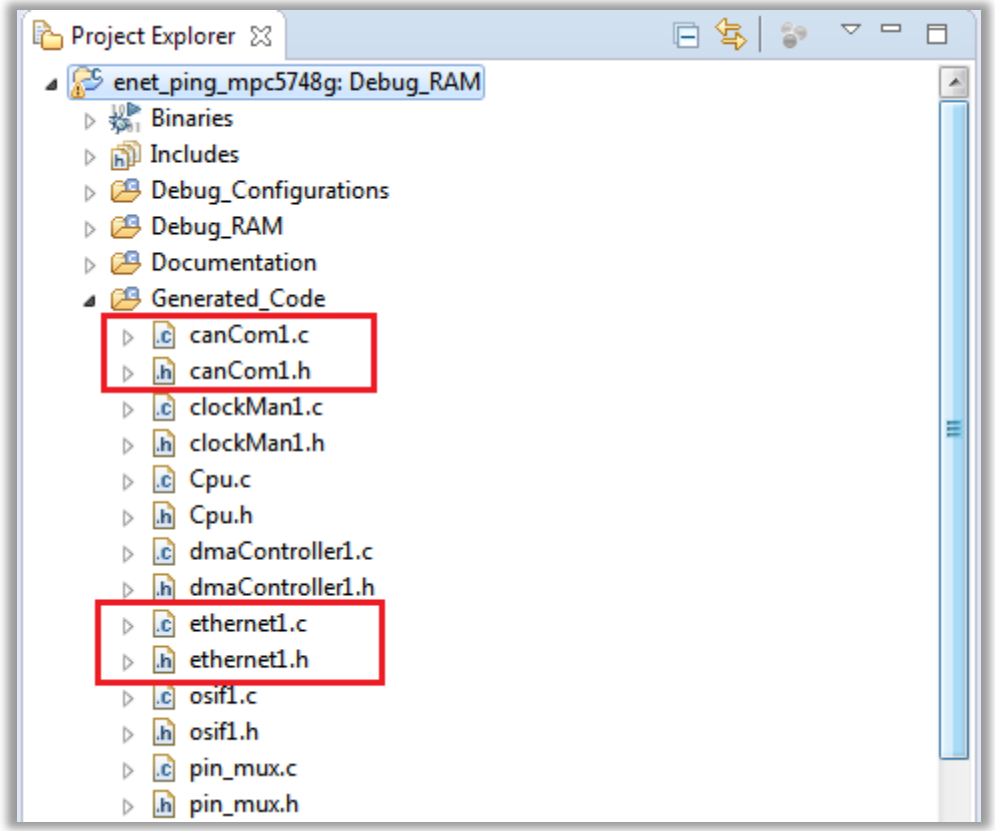
ADC | CAN | CMP | DCI | DSPI | ENET | FCCU | FlexRay | I2C | LINFlexD | MLB | Platform | PowerAndGround | SIUL2 | SPI

| 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

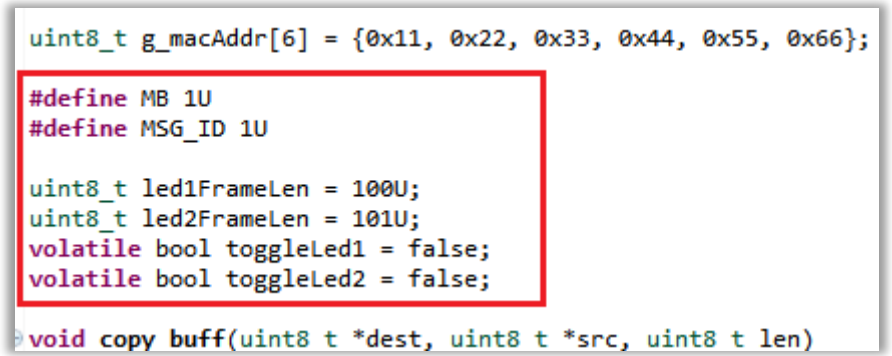
1) Below ENET driver init function, call the FlexCAN init driver function and define a ‘flexcan_data_info_t’ structure needed by the CAN driver to send data:

```
/* 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:

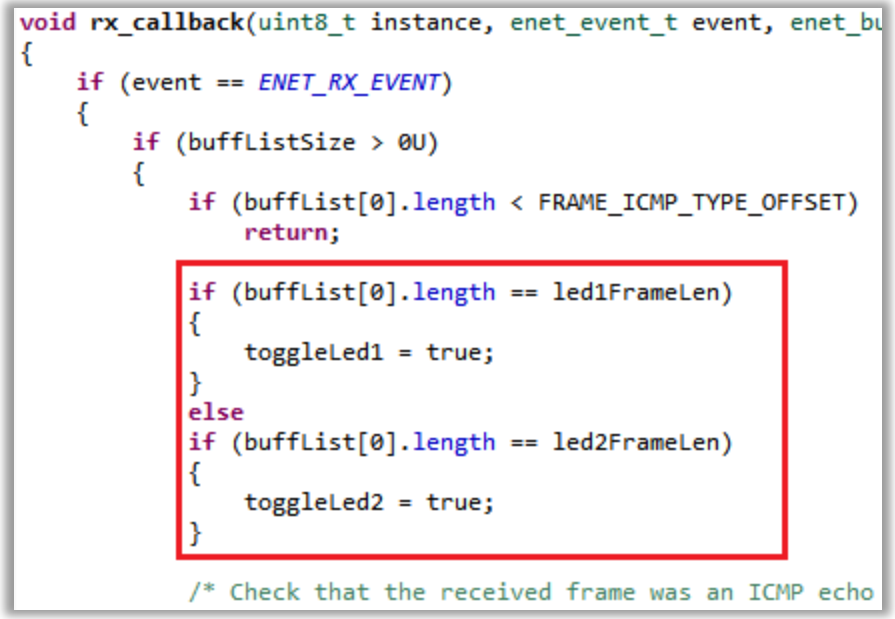
```
#define MB 1U      /* Index of the CAN message buffer */
#define MSG_ID 1U /* CAN message ID */

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 */
```



# 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 == led1FrameLen)
{
    /* frame length for LED1 toggle */
    toggleLed2 = true;
}
```



```
void rx_callback(uint8_t instance, enet_event_t event, enet_bu
{
    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 */
}
```

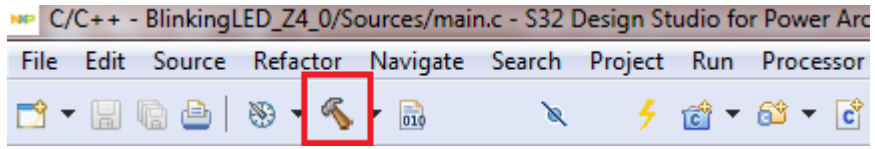
# Hands-on – ENET–CAN: Application Code

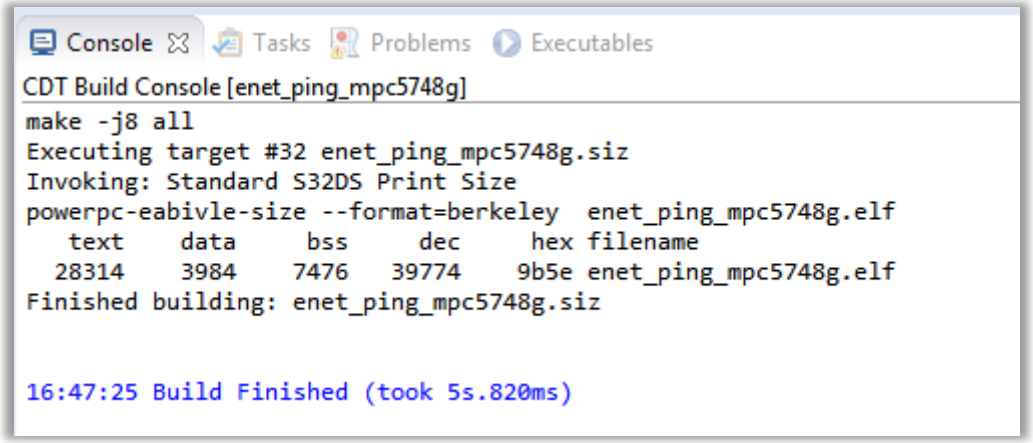
5) The new ‘main’ code (**only for the application receiving data via CAN**):

```
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->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 */
    }


}
```

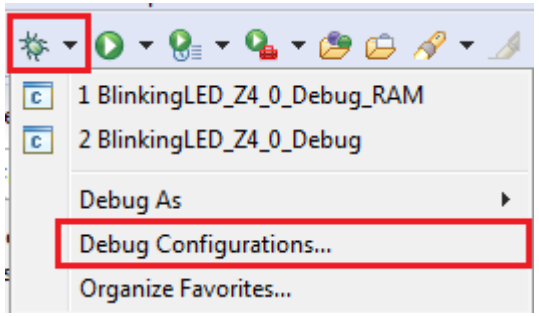
# Hands-on – ENET–CAN: Build and Debug

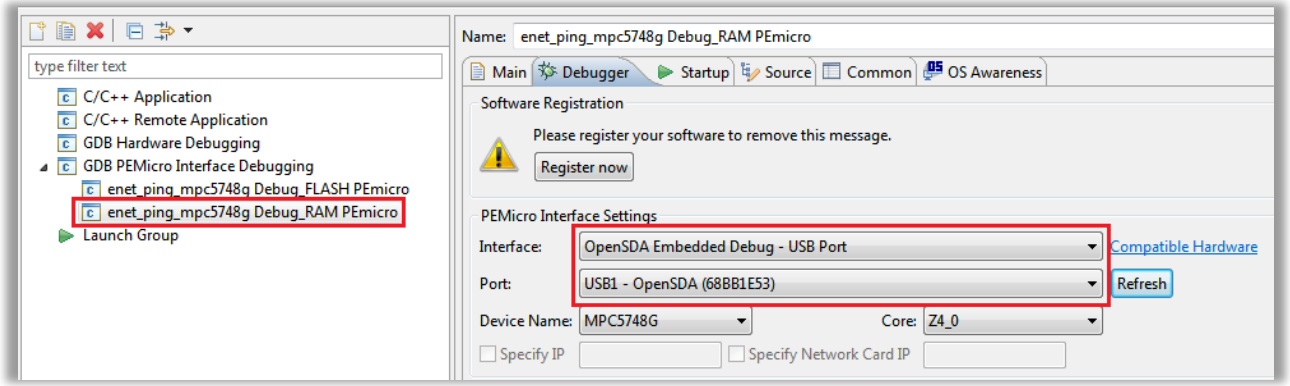
- Click the ‘build project’ button – make sure there are no compilation errors



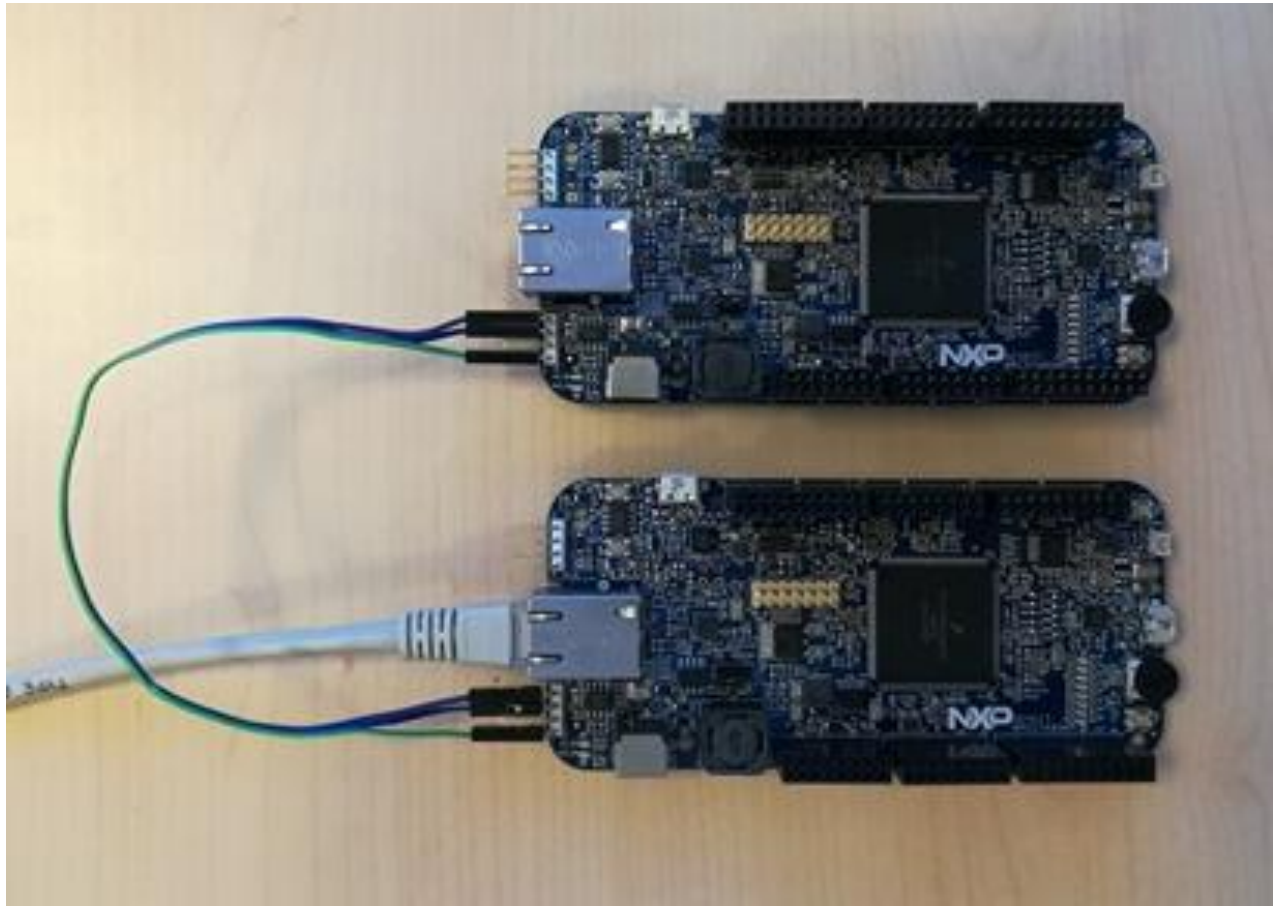

- Select the correct debug configuration and interface to debug the application



# 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

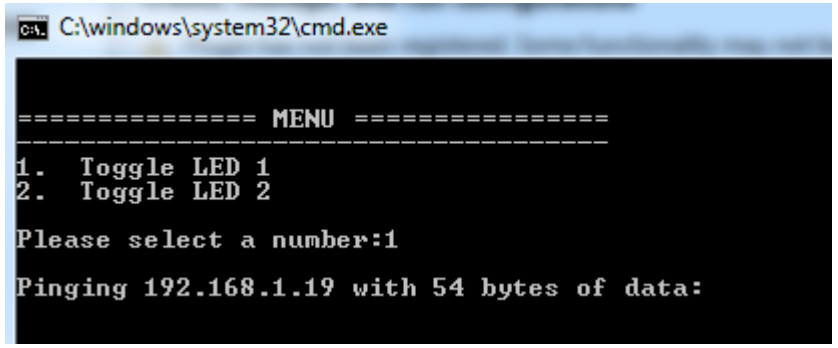
- In the ENET SDK example, the MAC address of the board is **11-22-33-44-55-66** and the IP is **192.168.1.19** (hardcoded values)
- Add this entry to your ARP table using the following command:

```
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




```
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