

Customer training workshop: Device configurator Communication

TRAVEO™ T2G CYT4BF series Microcontroller Training
V1.0.0 2022-12



Please read the [Important notice and warnings](#) at the end of this document

Scope of work

- › This document helps application developers understand how to use the Device Configurator for Communication as part of creating a ModusToolbox™ (MTB) application
 - The Device Configurator for Communication is part of a collection of tools included with the MTB software. It provides a GUI to configure the communication. This document describes use cases for CAN FD, UART, and SPI.

- › ModusToolbox™ tools package version: 3.0.0
- › Device Configurator version: 4.0
- › Device:
 - TRAVEO™ T2G CYT4BFBCH device is used in this code example
- › Board:
 - TRAVEO™ T2G KIT_T2G-B-H_EVK board is used for testing

Introduction

› The CAN FD controller has the following features:

- Flexible data-rate (FD) (ISO 11898-1: 2015)
 - Up to 64 data bytes per message
 - Maximum 8 Mbps supported
- Time-Triggered (TT) communication on CAN (ISO 11898-4: 2004)
 - TTCAN protocol level 1 and level 2 completely in hardware
- AUTOSAR support
- Acceptance filtering
- Two configurable receive FIFOs
- Up to 64 dedicated receive buffers
- Up to 32 dedicated transmit buffers
- Configurable transmit FIFO
- Configurable transmit queue
- Configurable transmit event FIFO
- Programmable loop-back test mode
- Power-down support
- Shared message RAM

Introduction (contd.)

› **The CAN FD controller has the following features:**

- ECC protection for message RAM
- Global fault structure to handle ECC errors
- Receive FIFO top pointer logic
 - Enables DMA access on FIFO
- DMA for debug message and received FIFOs
- Shared time stamp counter

Introduction (contd.)

› **The SCB controller has the following features:**

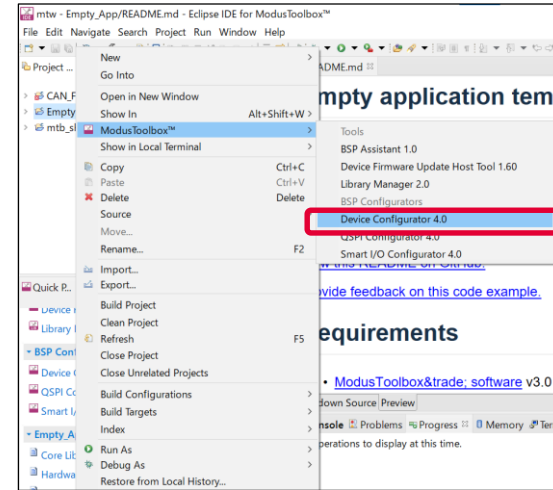
- Standard SPI master and slave functionality with Motorola, Texas Instruments, and National Semiconductor protocols
- Standard UART functionality with SmartCard reader, local interconnect network (LIN), and IrDA protocols
 - Standard LIN slave functionality with LIN v1.3 and LIN v2.1/2.2 specification compliance
The SCB has only standard LIN slave functionality.
- Standard I2C master and slave functionality
- EZ mode for SPI and I2C slaves; allows operation without CPU intervention
- CMD_RESP mode for SPI and I2C slaves; allows operation without CPU intervention and is available only on
- DeepSleep-capable SCB
- Low-power (DeepSleep) mode of operation for SPI and I2C slaves (using external clocking), available only on
- DeepSleep-capable SCB
- DeepSleep wakeup on I2C slave address match or SPI slave selection; available only on DeepSleep-capable SCB
- Trigger outputs for connection to DMA
- Multiple interrupt sources to indicate status of FIFOs and transfers
- Local loop-back control

Launch Device Configurator

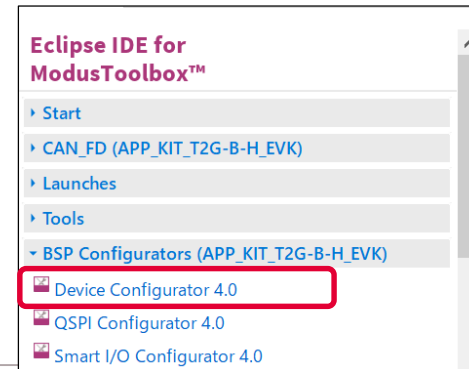
> From Eclipse IDE

Launch the Device configurator by either of the following methods:

a) Right-click on the project in “Project Explorer” and select **ModusToolbox™ > Device Configurator <version>**



b) Click the “Device Configurator” link in the Quick Panel



Device Configurator view for communication config

> From Eclipse IDE

- Open the "Peripherals" tab in the Device Configurator

The screenshot displays the Device Configurator interface with two callout boxes:

- Window for selecting peripheral and channel number:** Points to the "Peripherals" tab and the tree view where "Controller Area Network FD (CAN FD) 1" is expanded, and "Channel 1" is selected.
- Window for setting the operating parameters of selected peripherals:** Points to the "Channel 1 - Parameters" panel, which contains a table of configuration options.

Name	Value
Configuration Help	Open CAN FD Documentation
TxCallback Function	
RxCallback Function	
ErrorCallback Function	
CAN FD Mode	<input type="checkbox"/>

At the bottom, the Notice List shows:

- 0 Errors
- 0 Warnings
- 1 Task
- 1 Info

Fix Description: The 'Clock Signal' parameter must not be empty. Location: CYT4BFBCHE: Channel 1 [Clock Signal].

The WCO is enabled. Chip startup will be slower because clock configuration cannot continue until the WCO is ready. See the

Ready

Quick start

› **To use the Device Configurator for communication setting**

- Launch the Device Configurator.
- Use the various pull-down menus to configure signals.
- Save the file to generate source code.
- Device Configurator generates code into a "GeneratedSource" directory in your Eclipse IDE application, or in the same location you saved the *.modus file for non-IDE applications. That directory contains the necessary source (.c) and header (.h) files for the generated firmware, which uses the relevant driver APIs to configure the hardware.
- Use the generated structures as input parameters for communication functions in your application.

Use case for CAN FD



Use case

› Overview of configuration parameters for CAN FD:

- Mode : CAN FD
- CAN instance : CAN0_CH1
- Clock frequency : 40 MHz (Clock divider: Peri Clock Group 1 16-bit Divider 0)
- Used ports:
 - RX port = P0.3 (CYBSP_CAN_RX)
 - TX port = P0.2 (CYBSP_CAN_TX)
- Bitrate setting:

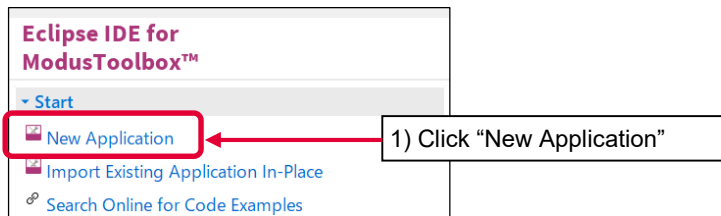
- Nominal bitrate	= 500 kbps
- Sampling point	= 75%
- Prescaler	= 10
- Nominal time segment 1	= 5
- Time segment 2	= 2
- Synchronization jump width	= 2
- See “CAN FD” application for operation
- Fast Bitrate Setting:

- Data Bitrate	= 1000 kbps
- Sampling Point	= 75 %
- Prescaler	= 5
- Data Time segment 1	= 5
- Data Time segment 2	= 2
- Data Synchronization Jump Width	= 2

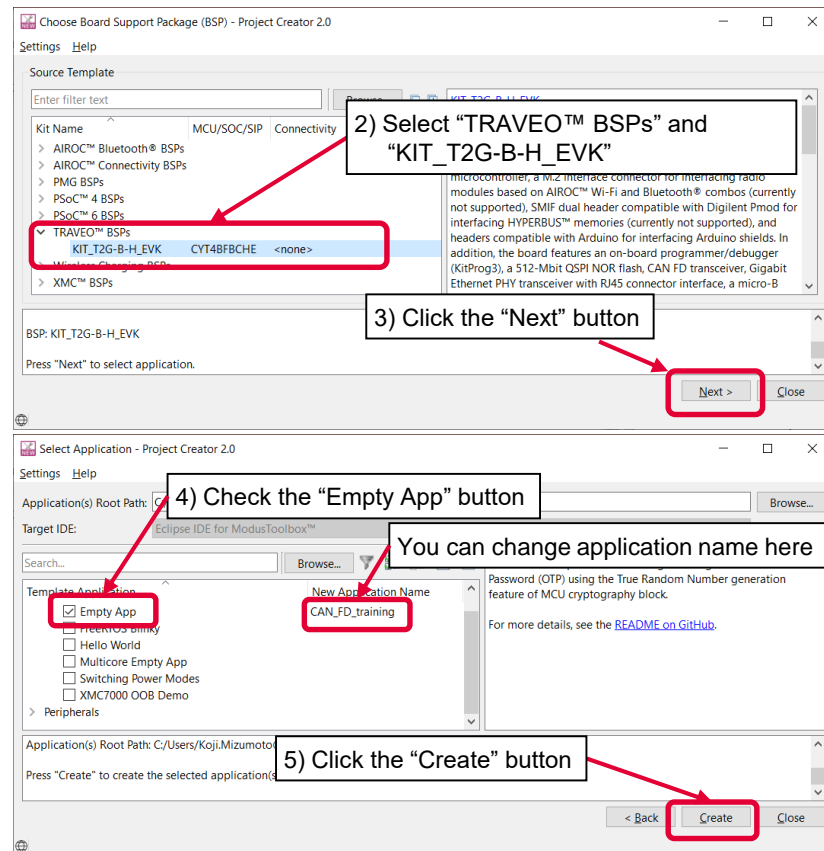
CAN FD configuration

> Create project

- 1) Click “New Application” in Quick Panel and open **Choose Board Support Package (BSP)** window



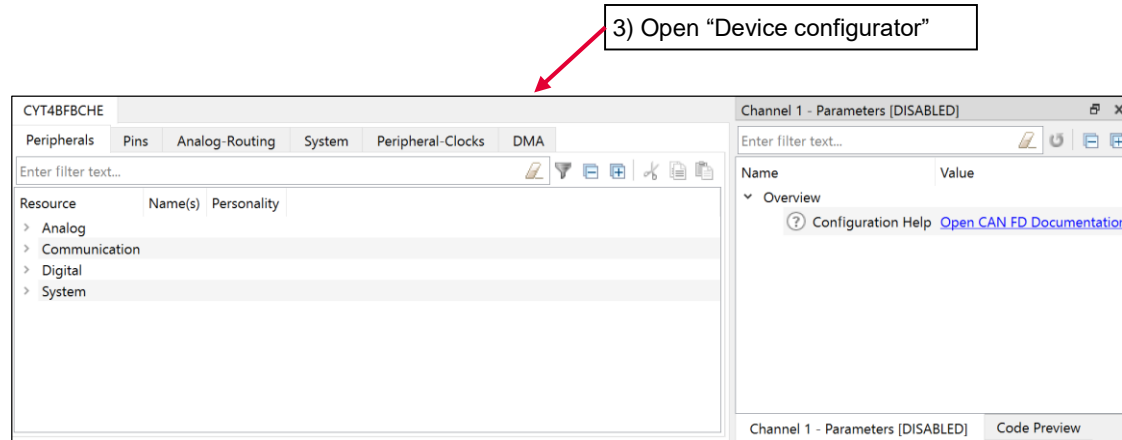
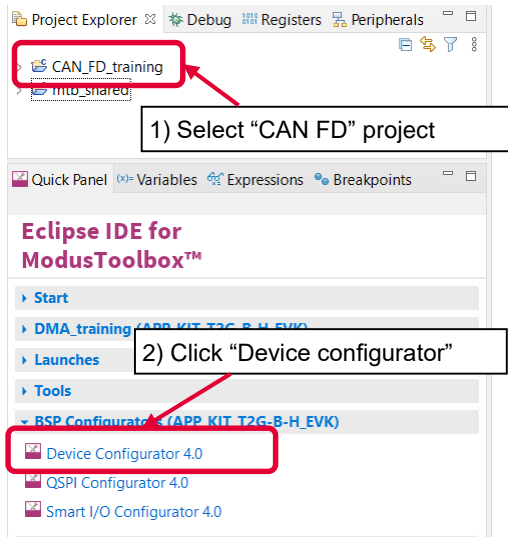
- 2) Select **TRAVEO™ BSPs** and **KIT_T2G-B-H_EVK**
- 3) Click **Next** and open the Application window
- 4) In this use case, it changes to “CAN_FD_training”
- 5) Click **Create** and start application creation



CAN FD configuration (contd.)

› Launch “Device configurator”:

- 1) Select the “CAN_FD_training” project.
- 2) Click “Device configurator” in Quick Panel
- 3) Open the “Device configurator” window



CAN FD configuration (contd.)

› Configure Clock (System):

- 1) Click the **System** tab
- 2) Select **PLL400M1**
- 3) Set “Desired Frequency” to “200.000”
- 4) Ensure that the frequency is set to 200 MHz

1) Click

2) Select

3) Set to 200.000

4) 200 MHz

Name	Value
Overview	
Configuration Help	Open PLL Documentation
General	
Source Frequency	8 MHz ± 1%
Low Frequency Mode	false
Configuration	Automatic
Desired Frequency (MHz)	200.000
Optimization	Min Power
Feedback (16-200)	50
Reference (1-16)	1

CAN FD configuration (contd.)

› Configure Clock (System):

- 4) Select **CLK_HF2**
- 5) Select **CLK_PATH2** as “Source Clock”
- 6) Set “Divider” to “1”
- 7) Ensure that the frequency is set to 200 MHz

The screenshot shows the 'Peripheral-Clocks' configuration window for device CYT4BFBCHE. The 'Peripheral-Clocks' tab is active, displaying a tree view of clock sources. A red box highlights the selection of **CLK_HF2** (200 MHz ± 1%) in the tree, with an arrow pointing to the instruction '4) Select'. Another red box highlights the selection of **CLK_PATH2** (340 MHz ± 1%) as the source clock for CLK_HF2, with an arrow pointing to the instruction '5) Select CLK_PATH2'. The 'CLK_HF2 - Parameters' dialog box is open, showing the following configuration:

Name	Value
Source Clock	CLK_PATH2
Source Frequency	200 MHz ± 1%
Divider	1
Frequency	200 MHz ± 1%

Red boxes and arrows in the dialog box highlight the 'Source Clock' field (pointing to '5) Select CLK_PATH2'), the 'Divider' field (pointing to '6) Set to 1'), and the 'Frequency' field (pointing to '7) 200 MHz').

CAN FD configuration (contd.)

› Configure Clock (Peripheral Clocks):

- 1) Click the **Peripheral-Clocks** tab for peripheral clock divider configuration
- 2) Select **16 bit Divider 0** in Peri Clock Group 1
- 3) Set “Divider” to “5”
- 4) You can see 40 MHz clock (200 MHz/5) as output frequency
- 5) Select **Channel 1 clock_can (CAN_FD)** as “Peripherals” connection

1) Click “Peripheral-Clocks” tab

2) Select 16 bit Divider 0 for CAN FD

3) Divider set to 5

4) 200 MHz/5 = 40 MHz

5) Select Channel 1 clock_can as peripherals

CAN FD configuration (contd.)

> Configure CAN FD (Clock and GPIO):

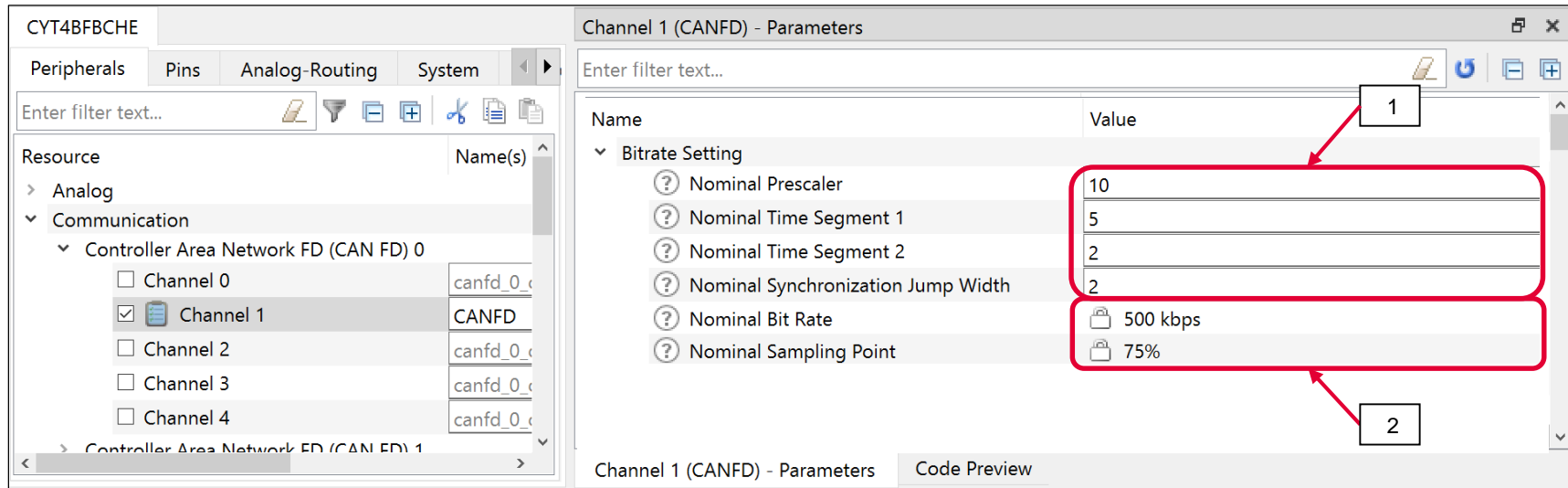
- 1) Make the following settings in the **Peripherals** tab
- 2) When you configure the peripheral clock connection in “Peripheral-Clocks”, CAN FD0 Channel1 is already selected.
- 3) Enter CANFD as the name
- 4) Set the “CAN FD Mode”
- 5) When you configure the peripheral clock connection, **16 bit Divider 0 clk** is already selected as Clock Signal
- 6) Select P0_3 (CAN_RX) and P0_2 (CAN_TX) to “CAN Rx Pin” and the “CAN Tx Pin”

The screenshot shows the configuration interface for CAN FD. The left sidebar shows the 'Resource' tree with 'Communication' expanded to 'Controller Area Network FD (CAN FD) 0'. 'Channel 1' is selected and checked. The name 'CANFD' is entered in the 'Name(s)' field. The right panel shows the 'Channel 1 (CANFD) - Parameters' configuration. The 'Mode' section has 'CAN FD Mode' checked. Under 'Connections', 'Clock Signal' is set to '16 bit Divider 0 clk [USED]', 'CAN Rx Pin' is set to 'P0[3] digital_in (CYBSP_CAN_RX) [USED]', and 'CAN Tx Pin' is set to 'P0[2] digital_out (CYBSP_CAN_TX) [USED]'. Red boxes and arrows highlight these specific settings, corresponding to the numbered steps in the text above.

CAN FD configuration (contd.)

› Configure CAN FD (Bitrate Setting):

- 1) Set the value of each Bitrate Setting
- 2) Ensure that “Nominal Bit Rate” is “500 kbps” and “Nominal Sampling Point” is “75%”



The screenshot shows the configuration interface for a CAN FD controller. The left pane shows a tree view with 'Channel 1' selected under 'Controller Area Network FD (CAN FD) 0'. The right pane displays the 'Channel 1 (CANFD) - Parameters' dialog, with the 'Bitrate Setting' section expanded. A red box highlights the 'Nominal Time Segment 1' (5), 'Nominal Time Segment 2' (2), 'Nominal Synchronization Jump Width' (2), 'Nominal Bit Rate' (500 kbps), and 'Nominal Sampling Point' (75%) settings. Red arrows labeled '1' and '2' point to the 'Nominal Time Segment 1' and 'Nominal Bit Rate' values, respectively.

Name	Value
Bitrate Setting	
Nominal Prescaler	10
Nominal Time Segment 1	5
Nominal Time Segment 2	2
Nominal Synchronization Jump Width	2
Nominal Bit Rate	500 kbps
Nominal Sampling Point	75%

CAN FD configuration (contd.)

› Configure CAN FD (Fast Bitrate Setting):

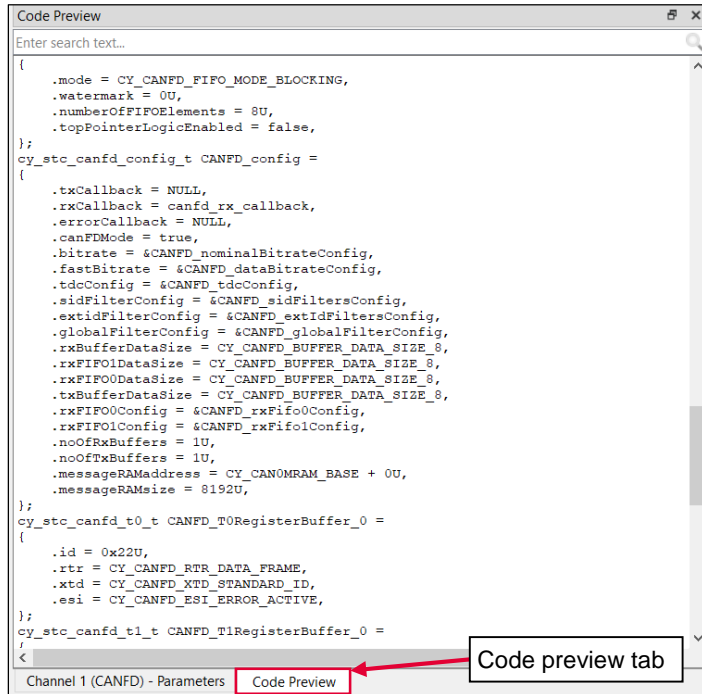
- 1) Set the value of each Fast Bitrate Setting
- 2) Ensure that “Data Bit Rate” is “1000 kbps” and “Data Sampling Point” is “75%”

Name	Value
Fast Bitrate Setting	
Data Prescaler	5
Data Time Segment 1	5
Data Time Segment 2	2
Data Synchronization Jump Width	2
Data Bit Rate	1000 kbps
Data Sampling Point	75%

CAN FD configuration (contd.)

> Confirm configuration result

- You can check the configuration result in the “Code Preview” tab of the Device Configurator



```

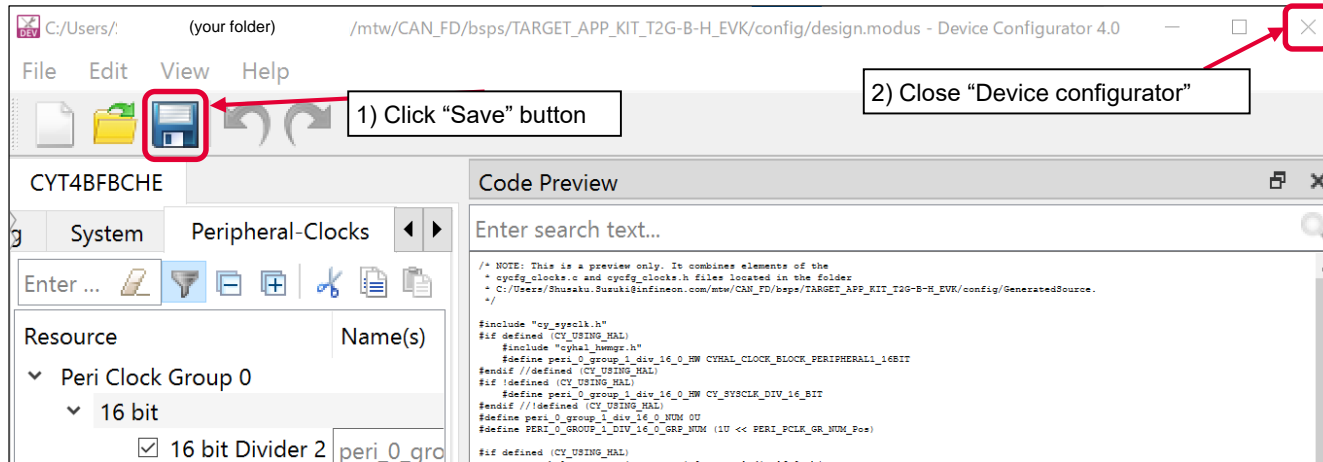
Code Preview
Enter search text...

{
    .mode = CY_CANFD_FIFO_MODE_BLOCKING,
    .watermark = 0U,
    .numberOfFIFOElements = 8U,
    .topPointerLogicEnabled = false,
};
cy_stc_canfd_config_t CANFD_config =
{
    .txCallback = NULL,
    .rxCallback = canfd_rx_callback,
    .errorCallback = NULL,
    .canFDMode = true,
    .bitrate = &CANFD_nominalBitrateConfig,
    .fastBitrate = &CANFD_dataBitrateConfig,
    .tdcConfig = &CANFD_tdcConfig,
    .sidFilterConfig = &CANFD_sidFiltersConfig,
    .extidFilterConfig = &CANFD_extIdFiltersConfig,
    .globalFilterConfig = &CANFD_globalFilterConfig,
    .rxBufferDataSize = CY_CANFD_BUFFER_DATA_SIZE_8,
    .rxFIFODataSize = CY_CANFD_BUFFER_DATA_SIZE_8,
    .txBufferDataSize = CY_CANFD_BUFFER_DATA_SIZE_8,
    .rxFIFO0Config = &CANFD_rxFifo0Config,
    .rxFIFO1Config = &CANFD_rxFifo1Config,
    .noOfRxBuffers = 1U,
    .noOfTxBuffers = 1U,
    .messageRAMAddress = CY_CAN0MRAM_BASE + 0U,
    .messageRAMsize = 8192U,
};
cy_stc_canfd_t0_t CANFD_T0RegisterBuffer_0 =
{
    .id = 0x22U,
    .rtr = CY_CANFD_RTR_DATA_FRAME,
    .xtd = CY_CANFD_XTD_STANDARD_ID,
    .esi = CY_CANFD_ESI_ERROR_ACTIVE,
};
cy_stc_canfd_t1_t CANFD_T1RegisterBuffer_0 =
{
    /
<
Channel 1 (CANFD) - Parameters Code Preview
    
```

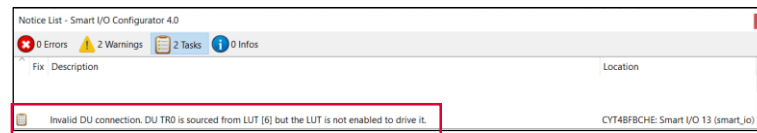
CAN FD configuration (contd.)

> Close Device configurator:

- Click the “Save” button after completing all settings, then close the “Device configurator”



- If an **Errors/Tasks** message appears, it should be resolved according to the instructions



CAN FD configuration (contd.)

> Configuration file:

- Close "Device configurator", it generates code into a "GeneratedSource" directory in your Eclipse IDE application, or in the same location you saved the *.modus file for non-IDE applications.
- This example has the following code:

The screenshot shows the Eclipse IDE interface. On the left, the 'GeneratedSource' directory is expanded, showing a list of files. Two files, 'cycfg_peripherals.c' and 'cycfg_peripherals.h', are highlighted with red dashed boxes. Red dashed arrows point from these boxes to the corresponding code blocks in the main editor window.

GeneratedSource Directory:

Name	Date modified
CAT1C_SMIFFLM	2022/11/25 13:24
cycfg.c	2022/11/24 17:45
cycfg.h	2022/11/22 16:54
cycfg.timestamp	2022/11/25 13:24
cycfg_clocks.c	2022/11/25 13:24
cycfg_clocks.h	2022/11/25 13:24
cycfg_notices.h	2022/11/24 17:45
cycfg_peripherals.c	2022/11/25 13:24
cycfg_peripherals.h	2022/11/25 13:24
cycfg_pins.c	2022/11/25 13:24
cycfg_pins.h	2022/11/25 13:24
cycfg_qspi_memslot.c	2022/11/22 16:54
cycfg_qspi_memslot.h	2022/11/22 16:54
cycfg_routing.h	2022/11/25 13:24
cycfg_system.c	2022/11/25 13:24
cycfg_system.h	2022/11/22 16:54
FlashCAT1C_SMIFout	2022/11/25 13:24
qspi_config.cfg	2022/11/22 16:54

cycfg_peripherals.c (lines 121-155):

```

121  ...mode = CY_CANFD_FIFO_MODE_BLOCKING,
122  ...watermark = 0U,
123  ...numberOfFIFOElements = 8U,
124  ...topPointerLogicEnabled = false,
125  };
126
127  cy_stc_canfd_config_t CANFD_config =
128  {
129  ...txCallback = NULL,
130  ...rxCallback = canfd_rx_callback,
131  ...errorCallback = NULL,
132  ...canFDMode = true,
133  ...bitrate = &CANFD_nominalBitrateConfig,
134  ...fastBitrate = &CANFD_dataBitrateConfig,
135  ...tdcConfig = &CANFD_tdcConfig,
136  ...sidFilterConfig = &CANFD_sidFiltersConfig,
137  ...extidFilterConfig = &CANFD_extIdFiltersConfig,
138  ...globalFilterConfig = &CANFD_globalFilterConfig,
139  ...rxBufferDataSize = CY_CANFD_BUFFER_DATA_SIZE_8,
140  ...rxFIFODataSize = CY_CANFD_BUFFER_DATA_SIZE_8,
141  ...txFIFODataSize = CY_CANFD_BUFFER_DATA_SIZE_8,
142  ...txBufferDataSize = CY_CANFD_BUFFER_DATA_SIZE_8,
143  ...rxFIFOConfig = &CANFD_rxFifoConfig,
144  ...txFIFOConfig = &CANFD_txFifoConfig,
145  ...noOfRxBuffers = 1U,
146  ...noOfTxBuffers = 1U,
147  ...messageRAMaddress = CY_CANOMRAM_BASE + 0U,
148  ...messageRAMsize = 8192U,
149  };
150  cy_stc_canfd_t0_t CANFD_T0RegisterBuffer_0 =
151  {
152  ...id = 0x22U,
153  ...rtr = CY_CANFD_RTR_DATA_FRAME,
154  ...xtd = CY_CANFD_XTD_STANDARD_ID,
155  ...esi = CY_CANFD_ESI_ERROR_ACTIVE,

```

cycfg_peripherals.h (lines 37-65):

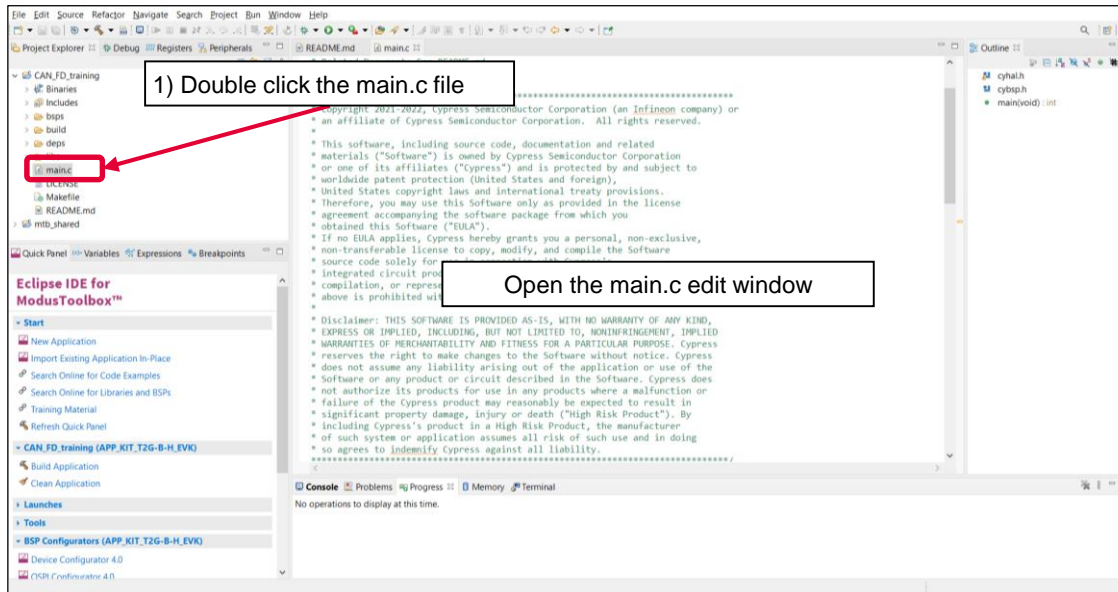
```

37  #endif //defined (CY_USING_HAL)
38
39  #if defined( _cplusplus)
40  extern "C" {
41  #endif
42
43  #define CANFD_ENABLED_1U
44  #define CANFD_HW_CANFD0
45  #define CANFD_CHANNEL_CANFD0_CH1
46  #define CANFD_STD_ID_FILTER_ID_0_0
47  #define CANFD_EXT_ID_FILTER_ID_0_0
48  #define CANFD_DATA_0_0
49  #define CANFD_DATA_1_1
50  #define CANFD_DATA_2_2
51  #define CANFD_DATA_3_3
52  #define CANFD_DATA_4_4
53  #define CANFD_DATA_5_5
54  #define CANFD_DATA_6_6
55  #define CANFD_DATA_7_7
56  #define CANFD_DATA_8_8
57  #define CANFD_DATA_9_9
58  #define CANFD_DATA_10_10
59  #define CANFD_DATA_11_11
60  #define CANFD_DATA_12_12
61  #define CANFD_DATA_13_13
62  #define CANFD_DATA_14_14
63  #define CANFD_DATA_15_15
64  #define CANFD_IRQ_0_canfd_0_interrupts0_1_IRQn
65  #define CANFD_IRQ_1_canfd_0_interrupts1_1_IRQn

```

Implementation

- › This section describes how to implement the configured CAN FD. This example will implement CAN FD configuration in the CAN_FD_training project.
 - Open main.c in the CAN_FD_training project



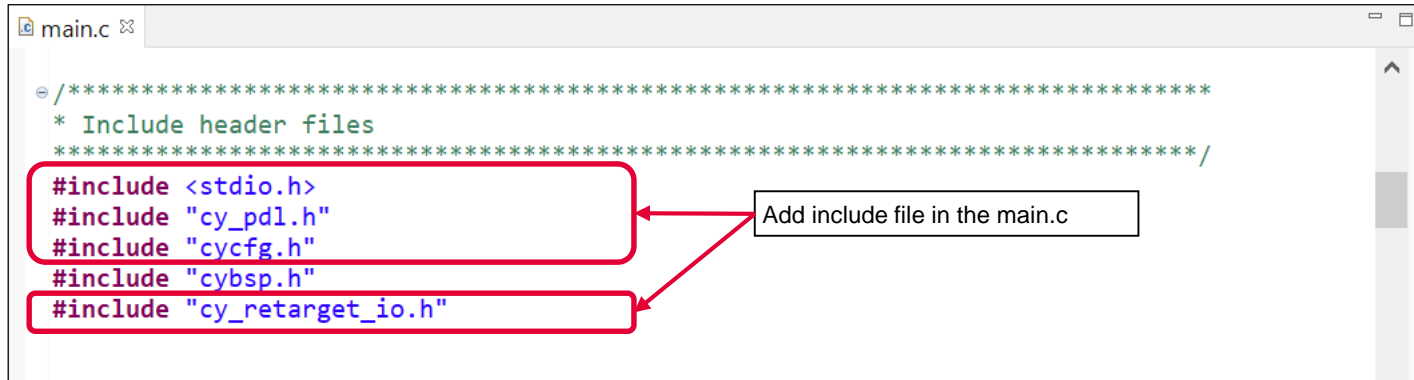
Implementation (contd.)

- › Add include file

```

main.c
/*****
 * Include header files
 *****/
#include <stdio.h>
#include "cy_pdl.h"
#include "cycfg.h"
#include "cybsp.h"
#include "cy_retarget_io.h"

```



The image shows a code editor window titled 'main.c'. The code contains a block of include statements enclosed in a multi-line comment. A callout box with the text 'Add include file in the main.c' has two red arrows pointing to the lines: '#include "cybsp.h"' and '#include "cy_retarget_io.h"'. Both of these lines are also enclosed in red rounded rectangular boxes.

Implementation (contd.)

› Add CAN FD initialization

The image shows two code editors side-by-side. The left editor, titled 'main.c', contains the following code:

```
int main(void)
{
    cy_rslt_t result;
    cy_en_canfd_status_t status;

    /* Initialize the device and board peripherals */
    result = cybsp_init();
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Initialize CANFD channel */
    status = Cy_CANFD_Init(CANFD_HW, CAN_HW_CHANNEL, &CANFD_config,
                          &canfd_context);
    if (status != CY_CANFD_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Sending CANFD frame to other node */
    status = Cy_CANFD_UpdateAndTransmitMsgBuffer(CANFD_HW,
        CAN_HW_CHANNEL,
        &CANFD_txBuffer_0,
        CAN_BUFFER_INDEX,
        &canfd_context);
}

```

Annotations on the left editor:

- A box labeled "Use this structure to configure CAN FD in the cycfg_peripherals.c file" points to the `&CANFD_config` argument in the `Cy_CANFD_Init` call.
- A box labeled "Add CAN FD initialization function" points to the `Cy_CANFD_Init` call.
- A box labeled "Add CAN FD transmit function" points to the `Cy_CANFD_UpdateAndTransmitMsgBuffer` call.

The right editor, titled 'cycfg_peripherals.c', shows the configuration structure:

```
cy_stc_canfd_config_t CANFD_config =
{
    .txCallback = NULL,
    .rxCallback = canfd_rx_callback,
    .errorCallback = NULL,
    .canFDMode = true,
    .bitrate = &CANFD_nominalBitrateConfig,
    .fastBitrate = &CANFD_dataBitrateConfig,
    .tdcConfig = &CANFD_tdcConfig,
    .sidFilterConfig = &CANFD_sidFiltersConfig,
    .extidFilterConfig = &CANFD_extIdFiltersConfig,
    .globalFilterConfig = &CANFD_globalFilterConfig,
    .rxBufferDataSize = CY_CANFD_BUFFER_DATA_SIZE_8,
    .rxFIFO1DataSize = CY_CANFD_BUFFER_DATA_SIZE_8,
    .rxFIFO0DataSize = CY_CANFD_BUFFER_DATA_SIZE_8,
    .txBufferDataSize = CY_CANFD_BUFFER_DATA_SIZE_8,
    .rxFIFO0Config = &CANFD_rxFifo0Config,
    .rxFIFO1Config = &CANFD_rxFifo1Config,
    .noOfRxBuffers = 1U,
    .noOfTxBuffers = 1U,
    .messageRAMaddress = CY_CAN0MRAM_BASE + 0U,
    .messageRAMsize = 8192U,
};

```

An annotation "Use this structure to configure CAN FD in the cycfg_peripherals.c file" points to the `CANFD_config` variable definition.

Implementation (contd.)

CAN FD initialization:

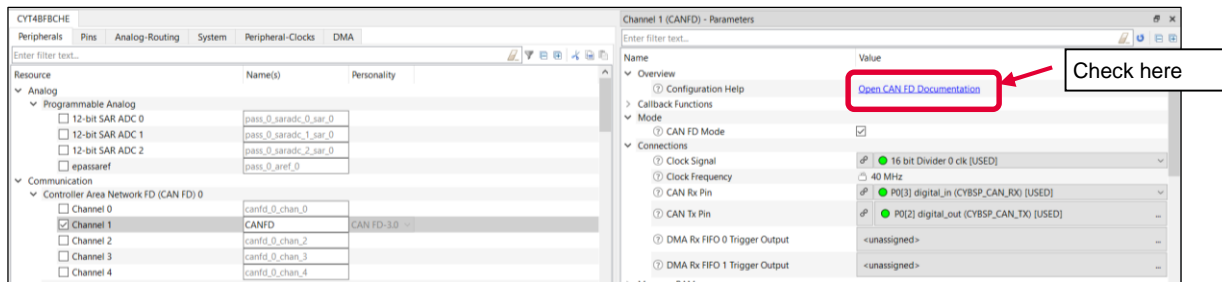
- › Call the [**Cy_CANFD_Init\(\)**](#) function to configure CAN FD
 - Initializes the CAN FD

CAN FD message transmit:

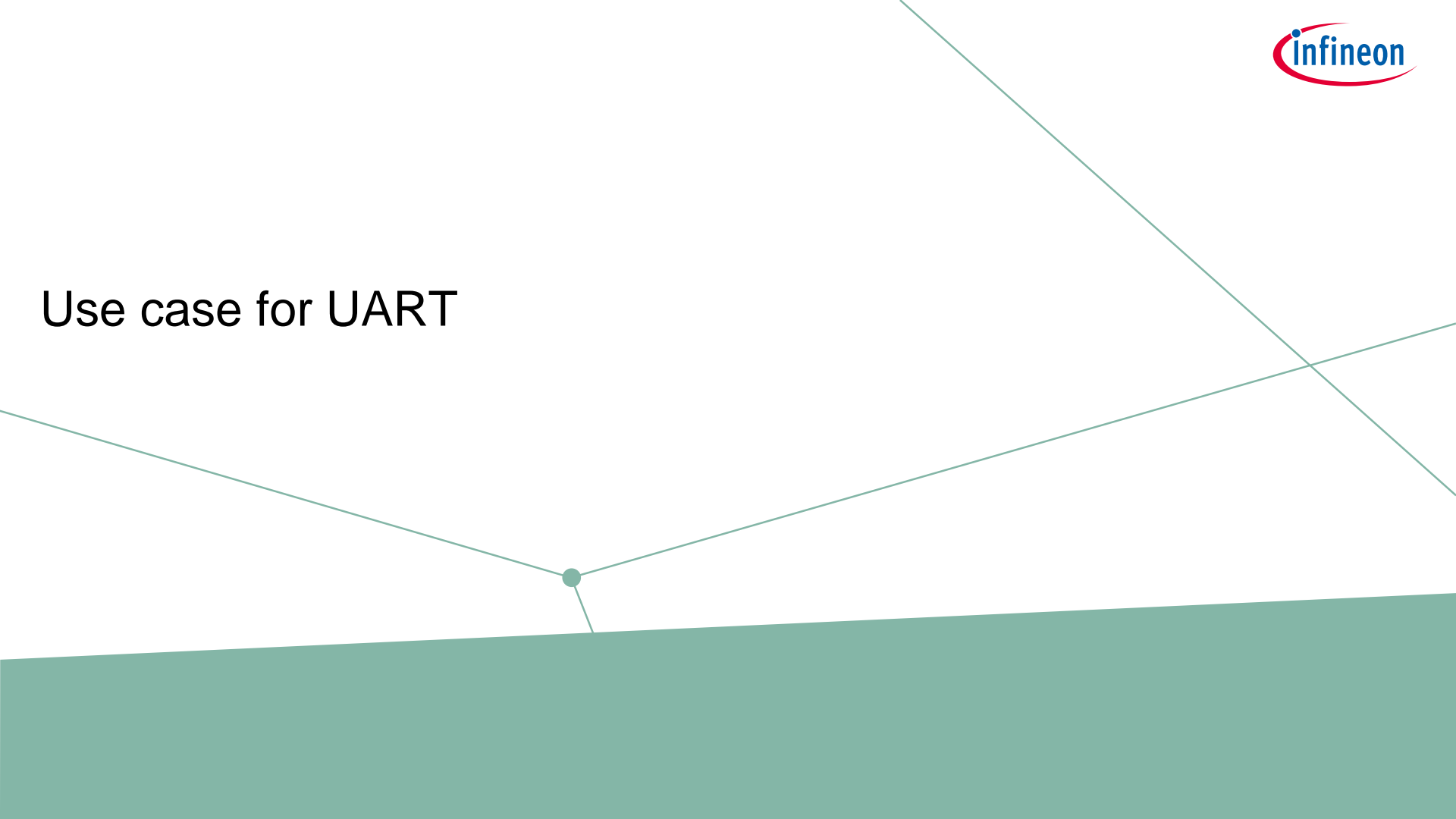
- › Call the [**Cy_CANFD_UpdateAndTransmitMsgBuffer\(\)**](#) function for CAN FD
 - Updates the Tx buffer element parameters in Message RAM, copies data to Message RAM, and then transmits the message.

Other functions:

- › Check the following for more information



Use case for UART



Use case

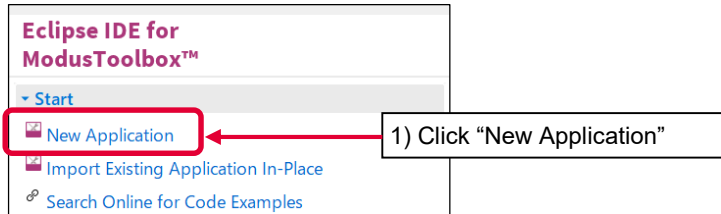
› Overview of configuration parameters for UART:

- Mode : Standard UART
- SCB instance : SCB3
- Clock frequency : 920.2 kHz (Clock divider: Peri Clock Group 1 8-bit Divider 0)
- Used ports:
 - Tx : SCB3_TX (P13.1)
 - Rx : SCB3_RX (P13.0)
- Baud rate : 115,200 bps
- Data width : 8 bits
- Parity : None
- Stop bits : 1
- Flow control : None
- See “SCB UART Transmit and Receive using DMA” application for operation

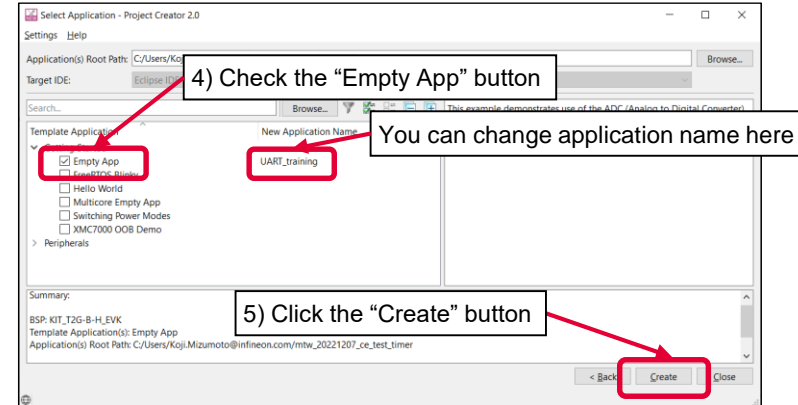
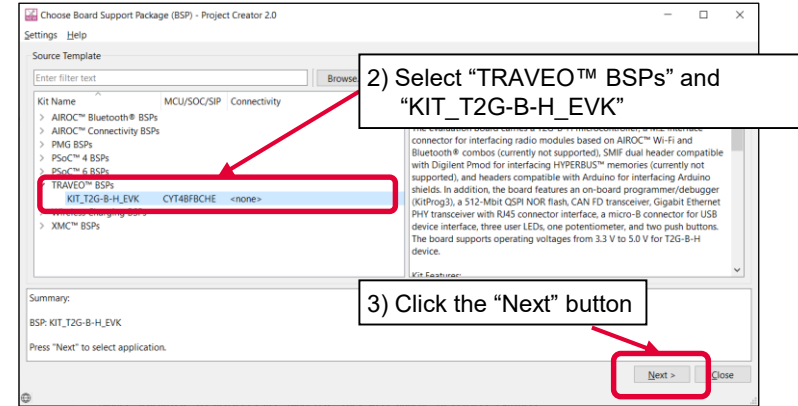
UART configuration

> Create project

- 1) Click **New Application** in Quick Panel and open the **Choose Board Support Package (BSP)** window



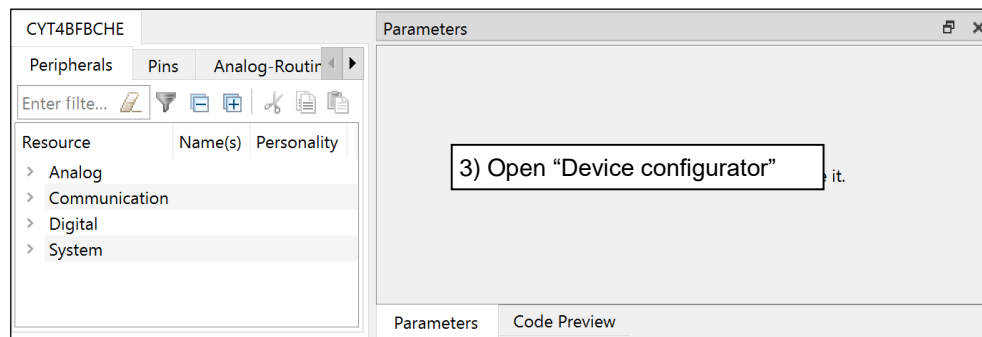
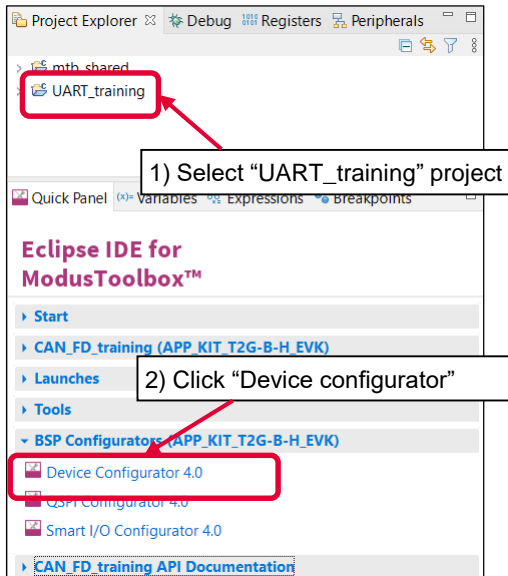
- 2) Select **TRAVEO™ BSPs** and **KIT_T2G-B-H_EVK**
- 3) Click **Next** button and open the Application window
- 4) In this use case, it changes to **“UART_training”**
- 5) Click **Create** and start application creation



UART configuration (contd.)

› Launch the “Device configurator”:

- 1) Select the **UART_training** project.
- 2) Click “Device configurator” in the Quick Panel
- 3) Open the “Device configurator” window



UART configuration (contd.)

› Configure Clock (System):

- 1) Click the **System** tab
- 2) Select **PLL400M1**
- 3) Set “Desired Frequency” to “196.000”
- 4) Ensure that the frequency is set to 196 MHz

1) Click

2) Select

3) Set to 196.000

4) 196 MHz

Name	Value
Configuration Help	Open PLL Document
Source Frequency	8 MHz ± 1%
Low Frequency Mode	false
Configuration	Automatic
Desired Frequency (MHz)	196.000
Optimization	Min Power
Feedback (16-200)	73
Reference (1-16)	1
Output (2-16)	3
Fraction divider (0-16777215)	8388608
Fraction Dither	false
Fraction Enable	true
Actual Frequency	196 MHz ± 1%

UART configuration (contd.)

> Configure Clock (System):

- 4) Select **CLK_HF2**
- 5) Select **CLK_PATH2** as “Source Clock”
- 6) Set “Divider” to “1”
- 7) Ensure that the frequency is set to 196 MHz

The screenshot displays the clock configuration interface for the CYT4BFBCHC device. On the left, the 'Resource' list shows various clock resources, with **CLK_HF2** selected and highlighted by a red box and an arrow labeled '4) Select'. The central diagram shows the clock tree with PLL400M0, PLL400M1, PLL0, and PLL1, and their respective paths (CLK_PATH1 to CLK_PATH4). **CLK_HF2** is highlighted with a red box and an arrow labeled '7) 196 MHz'. On the right, the 'CLK_HF2 - Parameters' window shows the configuration for the selected clock. The 'Source Clock' is set to **CLK_PATH2** (highlighted with a red box and arrow labeled '5) Select CLK_PATH2'), the 'Source Frequency' is 196 MHz ± 1%, the 'Divider' is set to **1** (highlighted with a red box and arrow labeled '6) Set to 1'), and the 'Frequency' is 196 MHz ± 1%.

UART configuration (contd.)

› Configure Clock (Peripheral Clocks):

- 1) Click the **Peripheral-Clocks** tab for the peripheral clock divider configuration
- 2) Select **8 bit Divider 0** in Peri Clock Group 1
- 3) Set “Divider” to “213”
- 4) You can see 920.02 kHz clock (196 MHz/213) as output frequency
- 5) Select **Serial communication Block (SCB) 3 clock** as “Peripherals” connection

1) Click “Peripheral-Clock” tab

2) Select 8 bit Divider0 for UART

3) Set Divider to 213

4) 196 MHz/213 = 920.2 kHz

5) Select Serial communication Block (SCB) 3 clock as peripherals

Select signal(s) - Device Configurator 4.0

Select any signal(s) to connect to 'Peripherals'.

Enter filter text...

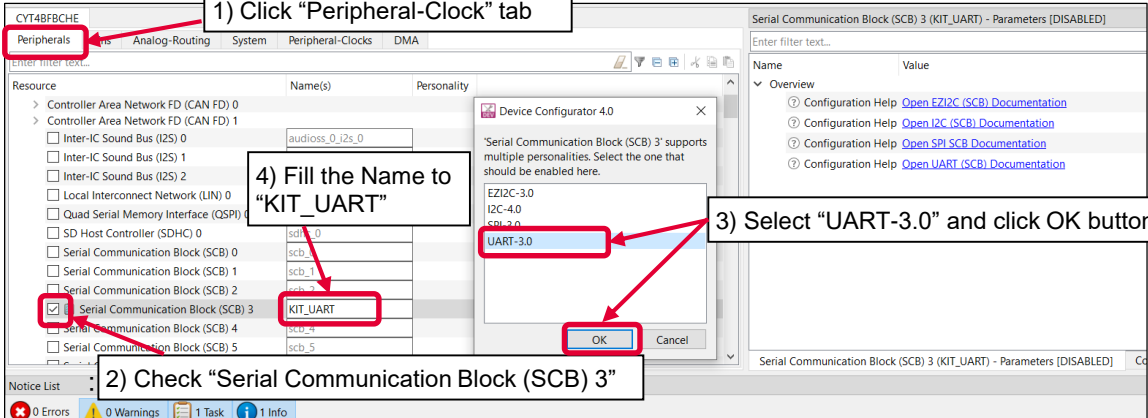
- Serial Communication Block (SCB) 0 clock
- Serial Communication Block (SCB) 1 clock
- Serial Communication Block (SCB) 2 clock
- Serial Communication Block (SCB) 3 clock [USED]
- Serial Communication Block (SCB) 4 clock
- Serial Communication Block (SCB) 5 clock
- Serial Communication Block (SCB) 6 clock
- Serial Communication Block (SCB) 7 clock

OK Cancel

UART configuration (contd.)

› Configure UART:

- 1) Check **Serial Communication Block (SCB) 3** in the **Peripherals** tab
- 2) Select **Serial Communication Block (SCB) 3** and fill in **KIT_UART** as the name
- 3) Select **UART-3.0** and click **OK**



1) Click "Peripheral-Clock" tab

2) Check "Serial Communication Block (SCB) 3"

3) Select "UART-3.0" and click OK button

4) Fill the Name to "KIT_UART"

Serial Communication Block (SCB) 3 (KIT_UART) - Parameters [DISABLED]

Name	Value
Overview	
Configuration Help	Open EZI2C (SCB) Documentation
Configuration Help	Open I2C (SCB) Documentation
Configuration Help	Open SPI SCB Documentation
Configuration Help	Open UART (SCB) Documentation

Serial Communication Block (SCB) 3 (KIT_UART) - Parameters [DISABLED]

UART configuration (contd.)

4) Set "Value" of "General" parameters

- Baud rate : 115,200 bps
- Data width : 8 bits
- Parity : None
- Stop bits : 1
- Flow control: None

Serial Communication Block (SCB) 3 (KIT_UART) - Parameters

Enter filter text...

Name	Value
Com Mode	Standard
Baud Rate (bps)	115200
Oversample	8
Bit Order	LSB First
Data Width	8 bits
Parity	None
Stop Bits	1 bit
Enable Digital Filter	

Serial Communication Block (SCB) 3 (KIT_UART) - Parameters Code Preview

5) Set "Value" of "Connections" parameters

- Clock divider: 8-bit Divider 0
- Used ports:
 - Tx : SCB3_TX (P13.1)
 - Rx : SCB3_RX (P13.0)

Serial Communication Block (SCB) 3 (KIT_UART) - Parameters

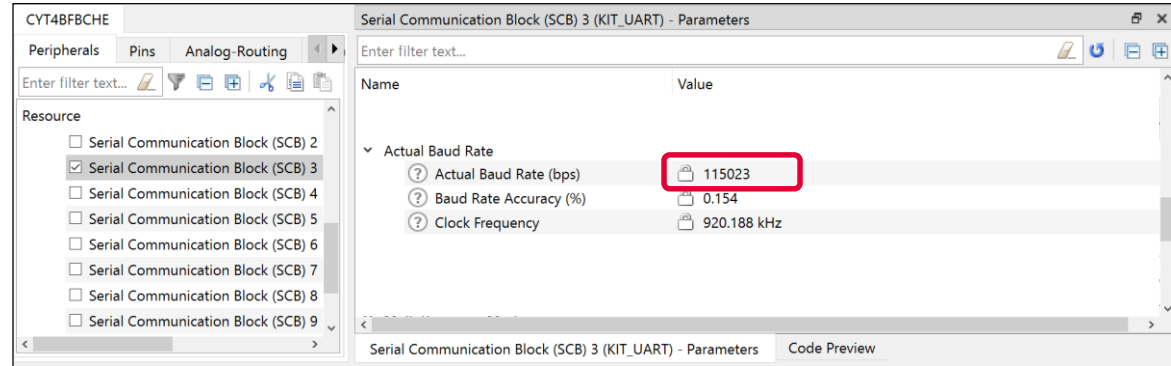
Enter filter text...

Name	Value
Clock	8 bit Divider 0 clk [USED]
RX	P13[0] digital_inout (CYBSP_DEBUG_UART_RX, CYBSP_D0) [USED]
TX	P13[1] digital_inout (CYBSP_DEBUG_UART_TX, CYBSP_D1) [USED]
RX Trigger Output	<unassigned>
TX Trigger Output	<unassigned>

Serial Communication Block (SCB) 3 (KIT_UART) - Parameters Code Preview

UART configuration (contd.)

- 6) Check the Actual Baud Rate and update it for your device



The screenshot displays the configuration window for 'Serial Communication Block (SCB) 3 (KIT_UART) - Parameters'. On the left, a 'Resource' list shows 'Serial Communication Block (SCB) 3' selected. The main area shows a table of parameters:

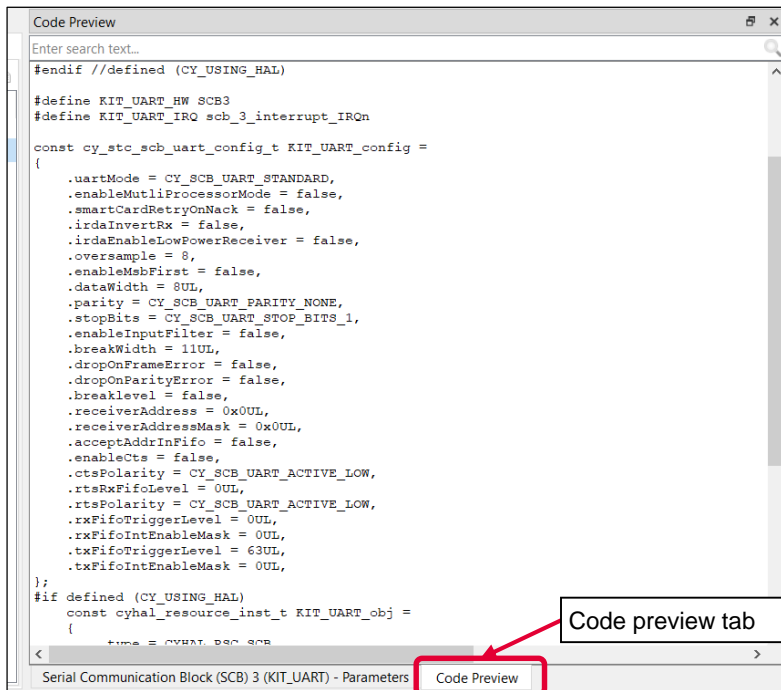
Name	Value
Actual Baud Rate	
Actual Baud Rate (bps)	115023
Baud Rate Accuracy (%)	0.154
Clock Frequency	920.188 kHz

The 'Actual Baud Rate' parameter and its value '115023' are highlighted with a red rectangular box.

UART configuration (contd.)

> Confirm configuration result

- You can check the configuration result in the “Code Preview” tab of the Device Configurator



The screenshot shows a window titled "Code Preview" with a search bar at the top. Below the search bar, there is C code defining UART parameters. A red box highlights the "Code Preview" tab in the bottom navigation bar, with a red arrow pointing to it from a label "Code preview tab".

```

Code Preview
Enter search text...

#endif //defined (CY_USING_HAL)

#define KIT_UART_HW_SCB3
#define KIT_UART_IRQ scb_3_interrupt_IRQn

const cy_stc_scb_uart_config_t KIT_UART_config =
{
    .uartMode = CY_SCB_UART_STANDARD,
    .enableMutliProcessorMode = false,
    .smartCardRetryOnNack = false,
    .irdaInvertRx = false,
    .irdaEnableLowPowerReceiver = false,
    .oversample = 8,
    .enableMsbFirst = false,
    .dataWidth = 8UL,
    .parity = CY_SCB_UART_PARITY_NONE,
    .stopBits = CY_SCB_UART_STOP_BITS_1,
    .enableInputFilter = false,
    .breakWidth = 11UL,
    .dropOnFrameError = false,
    .dropOnParityError = false,
    .breakLevel = false,
    .receiverAddress = 0x0UL,
    .receiverAddressMask = 0x0UL,
    .acceptAddrInFifo = false,
    .enableCts = false,
    .ctsPolarity = CY_SCB_UART_ACTIVE_LOW,
    .rtsRxFifoLevel = 0UL,
    .rtsPolarity = CY_SCB_UART_ACTIVE_LOW,
    .rxFifoTriggerLevel = 0UL,
    .rxFifoIntEnableMask = 0UL,
    .txFifoTriggerLevel = 63UL,
    .txFifoIntEnableMask = 0UL,
};

#if defined (CY_USING_HAL)
const cyhal_resource_inst_t KIT_UART_obj =
{
    .type = CYHAL_RSC_SCB

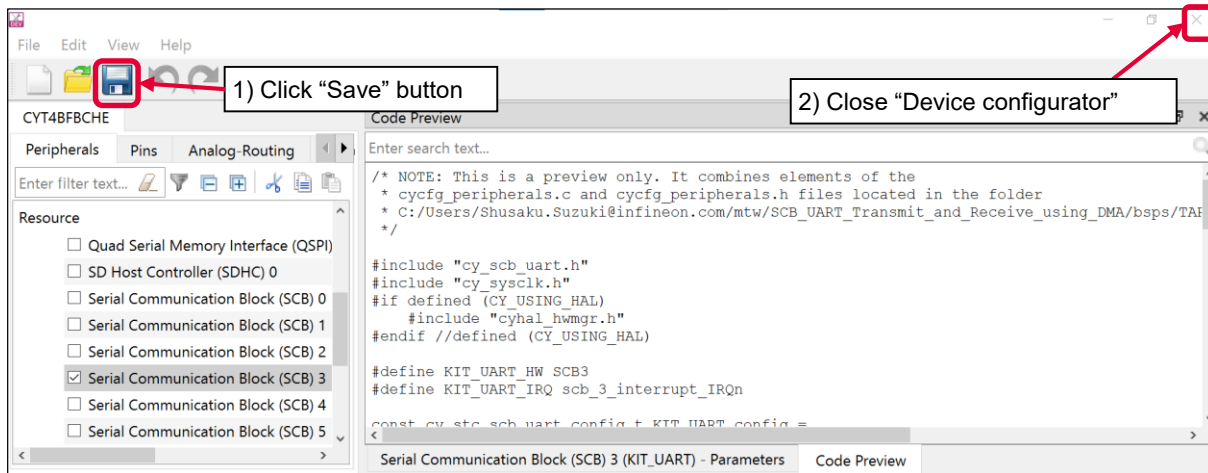
```

Serial Communication Block (SCB) 3 (KIT_UART) - Parameters **Code Preview**

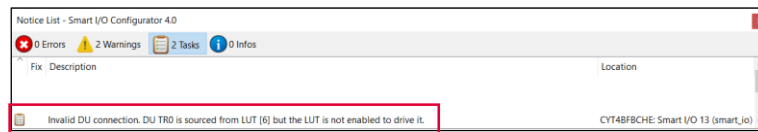
UART configuration (contd.)

> Close Device Configurator:

- Click the “Save” button after completing all settings, and then close the “Device configurator”



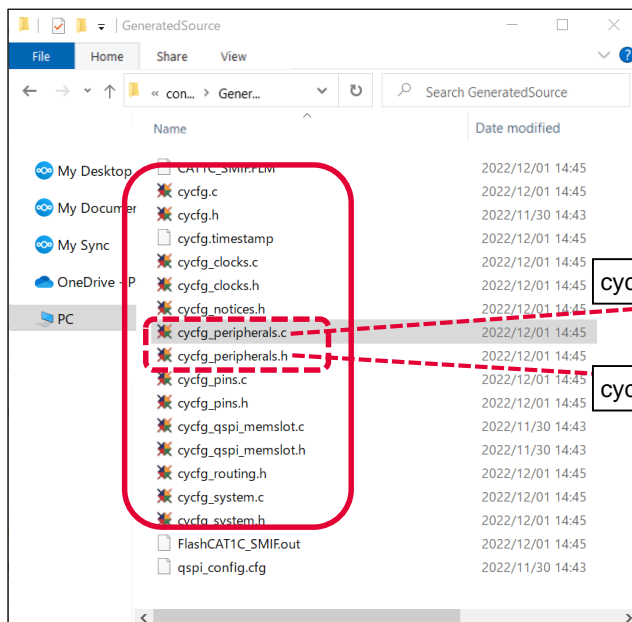
- If an **Errors/Tasks** message appears, it should be resolved according to the instructions



UART configuration (contd.)

> Configuration file:

- Close the “Device configurator”. It generates code into a "GeneratedSource" directory in your Eclipse IDE application, or in the same location you saved the *.modus file for non-IDE applications.
- This example has the following code:



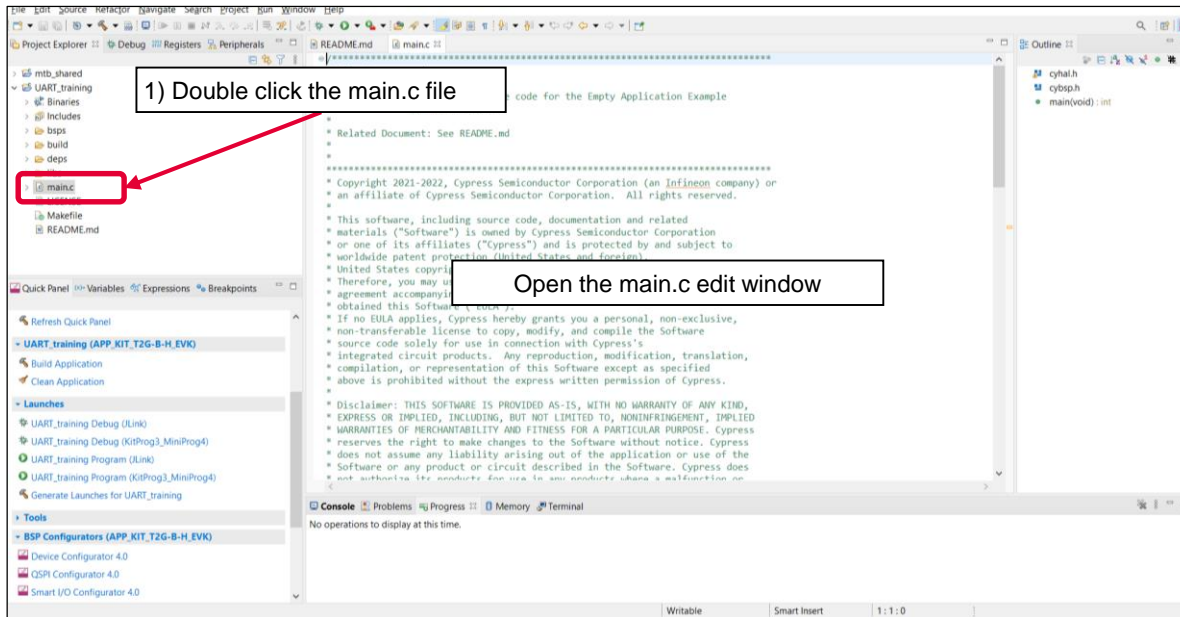
cycfg_peripherals.c

cycfg_peripherals.h

```
#include "cycfg_peripherals.h"
const cy_stc_scb_uart_config_t scb3_config =
{
    .uartMode = CY_SCB_UART_STANDARD,
    .enableMultiProcessorMode = false,
    .smartCardRetryOnNack = false,
    .irdsInvertRx = false,
    .irdsEnableLowPowerReceiver = false,
    .oversample = 8,
    .enableMsbFirst = false,
    .dataWidth = 8UL,
    .parity = CY_SCB_UART_PARITY_NONE,
    .stopBits = CY_SCB_UART_STOP_BITS_1,
    .enableInoutFilter = false,
    .breakWidth = 11UL,
    .dropOnFrameError = false,
    .dropOnParityError = false,
    .breakLevel = false,
    .receiverAddress = 0x0UL,
    .receiverAddressMask = 0x0UL,
    .acceptAddrInInfo = false,
    .enableCts = false,
    .ctsPolarity = CY_SCB_UART_RTSXFIfoLevel = 0UL,
    .rtsPolarity = CY_SCB_UART_RTSXFIfoLevel = 0UL,
    .rxFIfoTrigLevel = 630,
    .rxFIfoIntEnableMask = 0UL,
    .txFIfoTrigLevel = 630,
    .txFIfoIntEnableMask = 0UL
};
#if defined(CY_USING_HAL)
extern "C" {
    const cyhal_resource_inst_t
    {
        .type = CYHAL_RSC_SCB3,
        .clock_num = 30,
        .hw_scb3 = 0,
        .hw_scb3_irq = 0,
    };
}
#endif //defined (CY_USING_HAL)
extern const cy_stc_scb_uart_config_t scb3_config;
#if defined(CY_USING_HAL)
extern const cyhal_resource_inst_t scb3_obj;
#endif //defined (CY_USING_HAL)
void init_cycfg_peripherals(void);
void reserve_cycfg_peripherals(void);
void init_cycfg_peripherals(void);
void reserve_cycfg_peripherals(void);
#if defined(CY_USING_HAL)
    cyhal_linear_resource(scb3);
#endif //defined (CY_USING_HAL)
#endif /* CYCFG_PERIPHERALS_H */
```

Implementation

- › This section describes how to implement the configured UART. This example will implement UART configuration in the UART_training project.
 - Open main.c in the UART_training project



Implementation (contd.)

- › Add include file

```
*main.c
*****/

#include "cyhal.h"
#include "cybsp.h"
#include "cy_retarget_io.h"
```

Add include file in the main.c

Implementation (contd.)

> Add UART initialization and enable function

The image shows a code editor with two files: `main.c` and `cycfg_peripherals.c`. Annotations with arrows point to specific code lines:

- main.c:**
 - `init_status = Cy_SCB_UART_Init(KIT_UART_HW, &KIT_UART_config, &KIT_UART_context);` - Callout: "There is structure to configure UART in the cycfg_peripherals.c file" (points to `KIT_UART_config` in the other file).
 - `Cy_SCB_UART_Enable(KIT_UART_HW);` - Callout: "Add UART enable function".
 - `Cy_SCB_UART_PutString(KIT_UART_HW, "\x1b[2J\x1b[H");` - Callout: "Add UART control function, if necessary".
- cycfg_peripherals.c:**
 - `#include "cycfg_peripherals.h"` - Callout: "There is structure to configure UART in the cycfg_peripherals.c file" (points to `KIT_UART_config` in the other file).
 - `static cy_stc_scb_uart_config_t KIT_UART_config =` - Callout: "Add UART initialization function".
 - Configuration fields for `KIT_UART_config` such as `.uartMode = CY_SCB_UART_STANDARD`, `.breakWidth = 11UL`, etc.

Other visible UI elements include a "Peripherals" pane on the right showing "Serial Communication Block (SCB) 3 (KIT UART)" selected, and a "Code Preview" pane.

Implementation (contd.)

UART initialization:

- › Call the [**Cy_SCB_UART_Init\(\)**](#) function to configure UART
 - Initializes the SCB for UART operation

UART enable:

- › Call the [**Cy_SCB_UART_Enable\(\)**](#) function to enable UART
 - Enables the SCB for UART operation

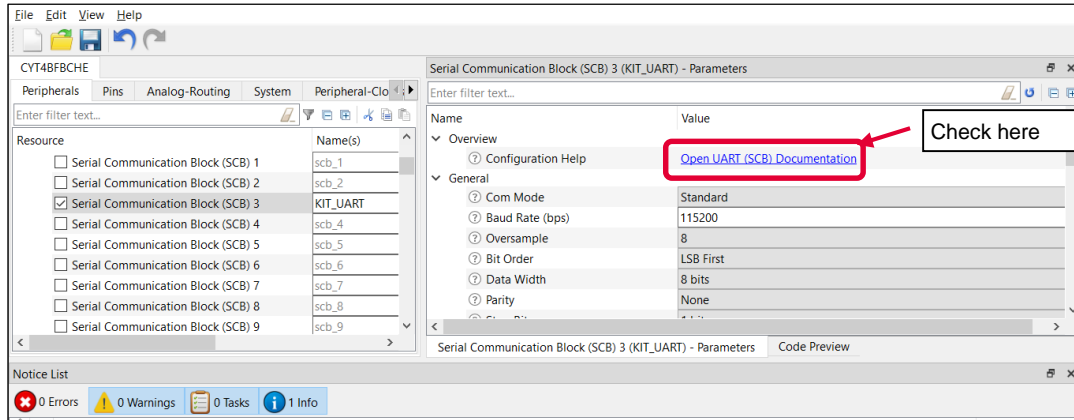
UART FIFO Control:

- › Call the [**Cy_SCB_UART_PutString\(\)**](#) function for UART TX FIFO
 - Places a NULL terminated string in the UART TX FIFO.
- › Call the [**Cy_SCB_UART_GetRxFifoStatus\(\)**](#) function for UART RX FIFO
 - Returns the current status of the UART RX FIFO.
- › Call the [**Cy_SCB_UART_ClearRxFifoStatus\(\)**](#) function for UART RX FIFO
 - Clears the selected statuses of the UART RX FIFO.

Implementation (contd.)

Other functions:

- › Check the following for more information



The screenshot shows the 'Serial Communication Block (SCB) 3 (KIT_UART) - Parameters' window. The 'General' section is expanded, showing various configuration parameters. A red box highlights the link 'Open UART (SCB) Documentation' under the 'Configuration Help' section. An arrow points from this link to a callout box labeled 'Check here'.

Name	Value
Com Mode	Standard
Baud Rate (bps)	115200
Oversample	8
Bit Order	LSB First
Data Width	8 bits
Parity	None

Use case for SPI



Use case

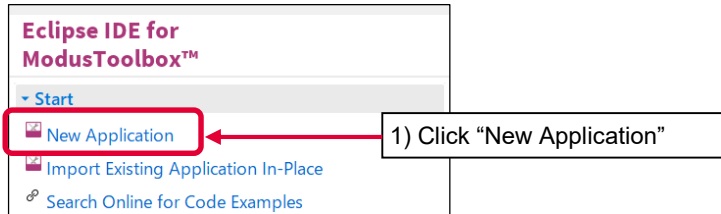
› Overview of configuration parameters for SPI:

- SCB mode = Motorola SPI Master mode
- SCB channels = 2
- Clock frequency: 16 MHz (Clock divider: Peri Clock Group 1 8-bit Divider 1)
- Bit rate = 1 Mbps
- Tx/Rx data width = 8 bits
- Used ports
 - SCLK : SCB2_CLK (P14.2)
 - MOSI : SCB2_MOSI (P14.1)
 - MISO : SCB2_MISO (P14.0)
 - SELECT : SCB2_SEL0 (P14.3), Active Low
- CPHA = 0, CPHL = 0
 - MOSI data is driven on a falling edge of SCLK
 - MISO data is captured on a falling edge of SCLK
- Trigger
 - Tx FIFO less than 63
- See “SCB_SPI_Master_DMA” application for operation

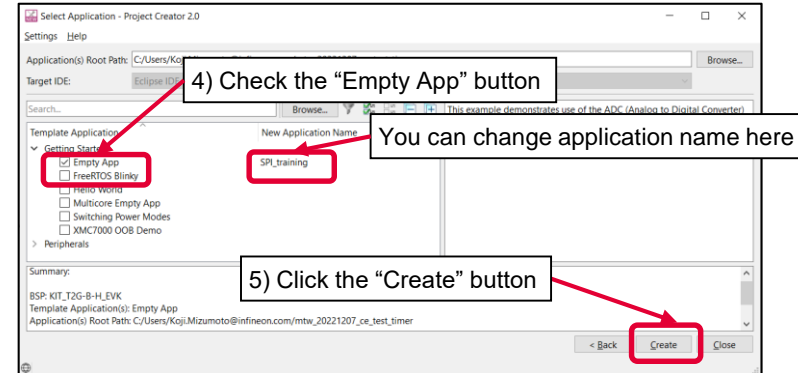
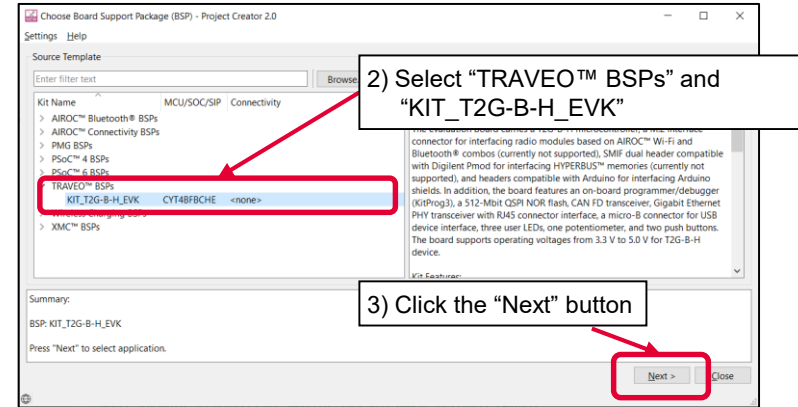
SPI configuration

> Create project

- 1) Click **New Application** in Quick Panel and open the **Choose Board Support Package (BSP)** window



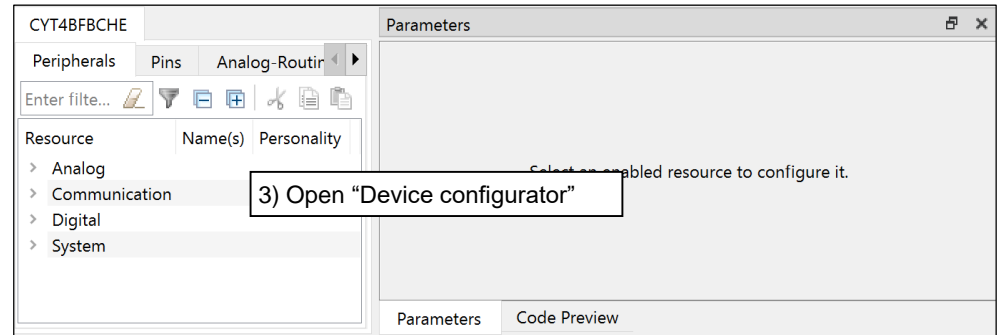
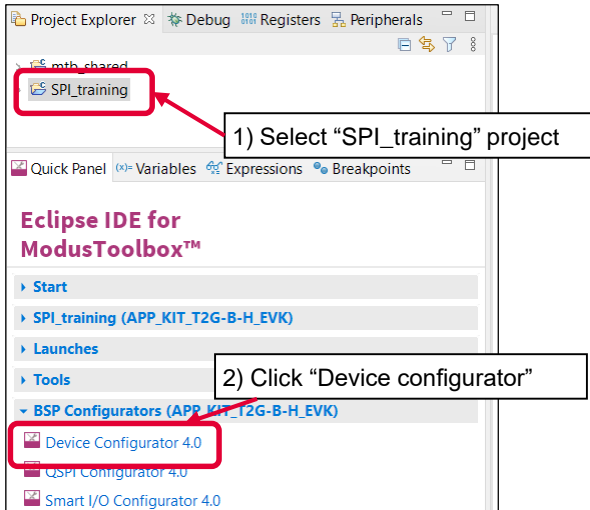
- 2) Select **TRAVEO™ BSPs** and **KIT_T2G-B-H_EVK**
- 3) Click **Next** and open the Application window
- 4) In this use cas?e, it changes to "SPI_training"
- 5) Click **Create** and start application creation



SPI configuration (contd.)

> Launch **Device configurator**:

- 1) Select the “SPI_training” project.
- 2) Click “Device configurator” in Quick Panel
- 3) Open the “Device configurator” window



SPI configuration (contd.)

› Configure Clock (System):

- 1) Click **System** tab
- 2) Select “PLL400M1”
- 3) Set “Desired Frequency” to “192.000”
- 4) Ensure that the frequency is set to 192 MHz

The screenshot shows the Infineon development tool interface. On the left, the 'System' tab is selected in the top navigation bar. Below it, a resource list table is visible:

Resource	Name(s)	Person
<input checked="" type="checkbox"/> PLL400M0	srss_0_clock_0_pll400m_0	PLL400
<input checked="" type="checkbox"/> PLL400M1	srss_0_clock_0_pll400m_1	PLL400
High Frequency		
<input checked="" type="checkbox"/> CLK_FAST0	istclk_0	CLK_FA
<input checked="" type="checkbox"/> CLK_FAST1	istclk_1	CLK_FA
<input checked="" type="checkbox"/> CLK_HF0	srss_0_clock_0_hfclk_0	CLK_HI
<input checked="" type="checkbox"/> CLK_HF1	srss_0_clock_0_hfclk_1	CLK_HI
<input checked="" type="checkbox"/> CLK_HF2	srss_0_clock_0_hfclk_2	CLK_HI
<input checked="" type="checkbox"/> CLK_HF3	srss_0_clock_0_hfclk_3	CLK_HI
<input checked="" type="checkbox"/> CLK_HF4	srss_0_clock_0_hfclk_4	CLK_HI
<input checked="" type="checkbox"/> CLK_HF5	srss_0_clock_0_hfclk_5	CLK_HI
<input checked="" type="checkbox"/> CLK_HF6	srss_0_clock_0_hfclk_6	CLK_HI
<input checked="" type="checkbox"/> CLK_HF7	srss_0_clock_0_hfclk_7	CLK_HI
<input type="checkbox"/> CLK_MEM	srss_0_clock_0_memclk_0	CLK_HI
<input checked="" type="checkbox"/> CLK_PERI	srss_0_clock_0_periclk_0	CLK_HI
<input type="checkbox"/> CLK_SLOW	srss_0_clock_0_slowclk_0	CLK_HI

In the center, a clock tree diagram shows PLL400M1 highlighted with a red box and labeled '4) 192 MHz'. Other PLLs shown include PLL0 (144 MHz ± 1%) and PLL1 (100 MHz ± 1%).

On the right, the 'PLL400M1 - Parameters' dialog is open. The 'Desired Frequency (MHz)' field is highlighted with a red box and labeled '3) Set to 192.000'. The 'Configuration' dropdown is set to 'Automatic'.

SPI configuration (contd.)

› Configure **Clock (System)**:

- 4) Select “CLK_HF2”
- 5) Select the “CLK_PATH2” as “Source Clock”
- 6) Set “Divider” to “1”
- 7) Ensure that the frequency is set to 192 MHz

The screenshot displays the Infineon configuration tool interface for the CYT48FB8CHE device. It is divided into three main sections:

- Left Pane (Resource List):** A table listing various clock resources. The 'CLK_HF2' resource is selected, indicated by a red box and an arrow pointing to it with the annotation "4) Select".
- Center Pane (Clock Tree Diagram):** A hierarchical diagram of the clock system. It shows PLLs (PLL400M0, PLL400M1, PLL0, PLL1) feeding into CLK_PATHs (CLK_PATH1 to CLK_PATH5). The 'CLK_HF2' resource is highlighted with a red box and an arrow pointing to it with the annotation "7) 192 MHz".
- Right Pane (CLK_HF2 - Parameters):** A configuration dialog for the selected 'CLK_HF2' resource. It shows the following settings:
 - Source Clock:** Set to 'CLK_PATH2', highlighted with a red box and an arrow pointing to it with the annotation "5) Select CLK_PATH2".
 - Source Frequency:** Set to '192 MHz ± 1%'.
 - Divider:** Set to '1', highlighted with a red box and an arrow pointing to it with the annotation "6) Set to 1".
 - Frequency:** Set to '192 MHz ± 1%'.

SPI configuration (contd.)

› Configure **Clock (Peripheral Clocks)**:

- 1) Click “Peripheral-clock” tab for peripheral clock divider configuration
- 2) Select “8 bit Divider 1” in Peri Clock Group 1
- 3) Set “Divider” to “12”
- 4) You can see 16 MHz clock (192 MHz/12) as output frequency
- 5) Select “Serial communication Block (SCB) 2 clock” as “Peripherals” connection

1) Click “Peripheral-Clock” tab

2) Select 8 bit Divider 1 for SPI

3) Set Divider to 12

4) 192 MHz/12 = 16 MHz

5) Select Serial communication Block (SCB) 2 clock as peripherals

SPI configuration (contd.)

› Configure **SPI**:

- 1) Check **Serial Communication Block (SCB) 2** in the Peripheral tab
- 2) Select “SPI-3.0”
- 3) Click **OK**
- 4) Fill the mSPI to name

The screenshot shows the Infineon Device Configurator 4.0 interface. The 'Peripherals' tab is selected, displaying a list of resources. The 'Serial Communication Block (SCB) 2' is checked, and its name is set to 'mSPI'. The 'Device Configurator 4.0' dialog is open, showing 'SPI-3.0' selected under the 'Serial Communication Block (SCB) 2' configuration. The 'OK' button is highlighted.

Resource	Name(s)
<input type="checkbox"/> Inter-IC Sound Bus (I2S) 1	audioss_1_i2s_0
<input type="checkbox"/> Inter-IC Sound Bus (I2S) 2	audioss_2_i2s_0
<input type="checkbox"/> Local Interconnect Network (LIN) 0	lin_0
<input type="checkbox"/> Serial Communication Block (SCB) 0	scb_0
<input type="checkbox"/> Serial Communication Block (SCB) 1	scb_1
<input checked="" type="checkbox"/> Serial Communication Block (SCB) 2	mSPI
<input type="checkbox"/> Serial Communication Block (SCB) 3	scb_3
<input type="checkbox"/> Serial Communication Block (SCB) 4	scb_4
<input type="checkbox"/> Serial Communication Block (SCB) 5	scb_5
<input type="checkbox"/> Serial Communication Block (SCB) 6	scb_6
<input type="checkbox"/> Serial Communication Block (SCB) 7	scb_7

Annotations in the image:

- 1) Check “Serial Communication Block (SCB) 2” (points to the checked checkbox)
- 2) Select “SPI-3.0” (points to the selected option in the dialog)
- 3) Click (points to the OK button)
- 4) Fill the Name to “mSPI” (points to the name field)

SPI configuration (contd.)

- 5) Set "Value" of "General" parameters
 - SCB Mode = Motorola SPI Master mode
 - CPHA = 0, CPOL = 0

Serial Communication Block (SCB) 2 (mSPI) - Parameters

Enter filter text...

Name: mSPI

Personality: SPI-3.0

General

- Mode: Master
- Sub Mode: Motorola
- CPHA = 0, CPOL = 0
- SCLK Mode: CPHA = 0, CPOL = 0

- 6) Set "Value" of "Data Configuration" parameters
 - Tx/Rx data width = 8 bits

Serial Communication Block (SCB) 2 (mSPI) - Parameters

Enter filter text...

Name: mSPI

Personality: SPI-3.0

Data Configuration

- RX Data Width: 8
- TX Data Width: 8

SPI configuration (contd.)

7) Set "Value" of "Connection" parameters

- Clock frequency: 16 MHz (Clock divider: Peri Clock Group 1 8-bit Divider 1)
- Used ports
 - SCLK: SCB2_CLK (P14.2)
 - MOSI: SCB2_MOSI (P14.1)
 - MISO: SCB2_MISO (P14.0)
 - SELECT: SCB2_SEL0 (P14.3), Active Low

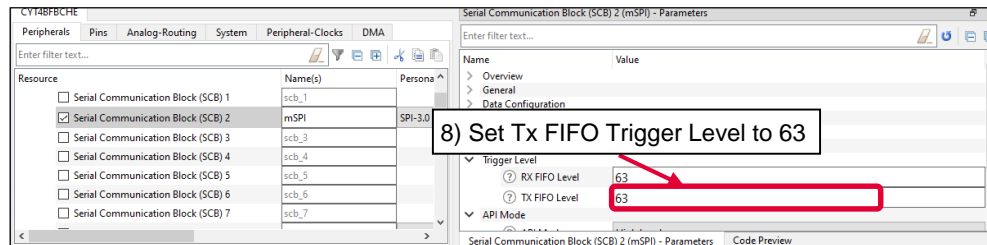
The screenshot shows the configuration for the Serial Communication Block (SCB) 2 (mSPI). The Parameters window is open, showing the following settings:

Name	Value
Slave Select	
Hold Delay	1.5 Clock Cycles
Inter-dataframe Delay	1.5 Clock Cycles
SS0 Polarity	Active Low
SS1 Polarity	Active Low
Clock	8 bit Divider 1 clk [USED]
MISO	P14[2] digital_inout (CYBSP_A2) [USED]
SS0	P14[3] digital_inout (CYBSP_A3) [USED]
SS1	<unassigned>
SS2	<unassigned>
RX Trigger Output	<unassigned>
TX Trigger Output	<unassigned>
Data Rate	
Actual Data Rate (kbps)	1000.000
Clock Frequency	16 MHz

SPI configuration (contd.)

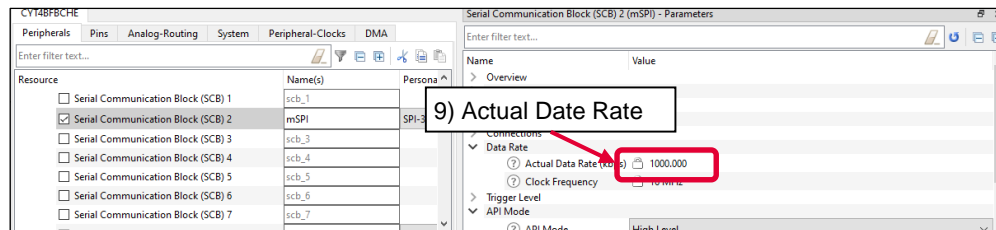
8) Set "Value" of "Trigger level" parameters

- Trigger
 - Tx FIFO less than 63



9) Check "Actual Data Rate (kbps)"

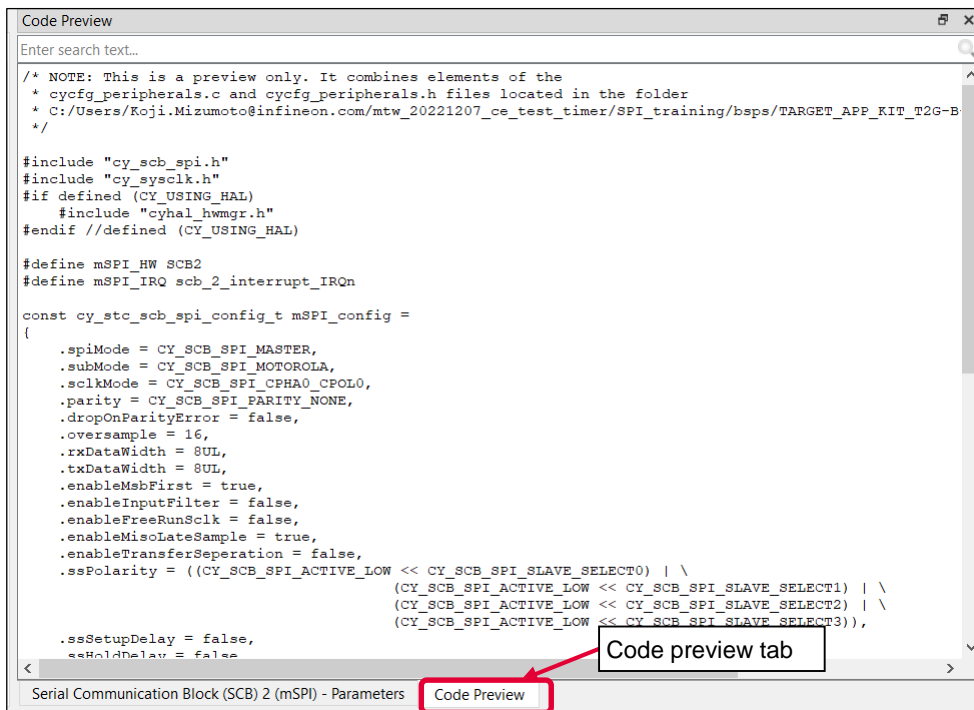
- Bit rate: 1 Mbps



SPI configuration (contd.)

> Confirm configuration result

- You can check the configuration result in the “Code Preview” tab of Device configurator



```

Code Preview
Enter search text...

/* NOTE: This is a preview only. It combines elements of the
 * cycfg_peripherals.c and cycfg_peripherals.h files located in the folder
 * C:/Users/Koji.Mizumoto@infineon.com/mtw_20221207_ce_test_timer/SPI_training/bmps/TARGET_APP_RIT_T2G-B
 */

#include "cy_scb_spi.h"
#include "cy_sysclk.h"
#if defined (CY_USING_HAL)
#include "cyhal_hwmgr.h"
#endif //defined (CY_USING_HAL)

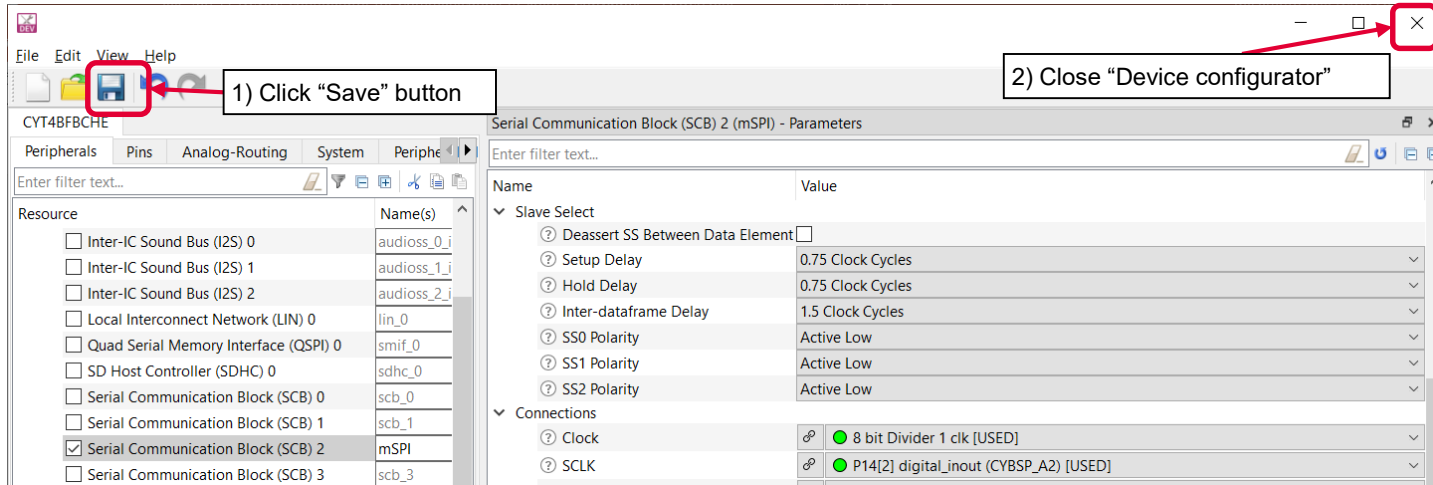
#define mSPI_HW_SCB2
#define mSPI_IRQ scb_2_interrupt_IRQn

const cy_stc_scb_spi_config_t mSPI_config =
{
    .spiMode = CY_SCB_SPI_MASTER,
    .subMode = CY_SCB_SPI_MOTOROLA,
    .clkMode = CY_SCB_SPI_CPHA0_CPOL0,
    .parity = CY_SCB_SPI_PARITY_NONE,
    .dropOnParityError = false,
    .oversample = 16,
    .rxDataWidth = SUL,
    .txDataWidth = SUL,
    .enableMsbFirst = true,
    .enableInputFilter = false,
    .enableFreeRunSclk = false,
    .enableMisolateSample = true,
    .enableTransferSeperation = false,
    .ssPolarity = ((CY_SCB_SPI_ACTIVE_LOW << CY_SCB_SPI_SLAVE_SELECT0) | \
                  (CY_SCB_SPI_ACTIVE_LOW << CY_SCB_SPI_SLAVE_SELECT1) | \
                  (CY_SCB_SPI_ACTIVE_LOW << CY_SCB_SPI_SLAVE_SELECT2) | \
                  (CY_SCB_SPI_ACTIVE_LOW << CY_SCB_SPI_SLAVE_SELECT3)),
    .ssSetupDelay = false,
    .ssHoldDelay = false
}
Serial Communication Block (SCB) 2 (mSPI) - Parameters Code Preview
  
```

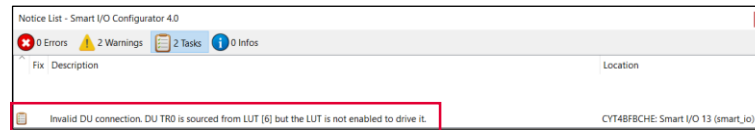
SPI configuration (contd.)

› Close Device Configurator:

- Click the “Save” button after completing all settings, then close “Device configurator”



- If an Errors/Tasks message appears, it should be resolved according to the instructions



SPI configuration (contd.)

> Configuration file:

- Close "Device configurator"; it generates code into a "GeneratedSource" directory in your Eclipse IDE application, or in the same location where you saved the *.modus file for non-IDE applications.
- This example has the following code:

The image shows two parts of the Eclipse IDE. On the left, a file explorer window displays the 'GeneratedSource' directory. A red dashed box highlights the 'cycfg_peripherals.c' file. On the right, the code editor shows the contents of this file. A red dashed arrow points from the highlighted file in the explorer to the code editor. Another red dashed arrow points from the code editor to a smaller inset window showing the license for the code.

```

30
31 const cy_stc_scb_spi_config_t mSPI_config =
32 {
33     .spiMode = CY_SCB_SPI_MASTER,
34     .subMode = CY_SCB_SPI_MOTOROLA,
35     .sclkMode = CY_SCB_SPI_CPHA0_CPOLO,
36     .parity = CY_SCB_SPI_PARITY_NONE,
37     .dropOnParityError = false,
38     .overSample = 16,
39     .rxDataWidth = 8UL,
40     .txDataWidth = 8UL,
41     .enableMsbFirst = true,
42     .enableInputFilter = false,
43     .enableFreeRunSclk = false,
44     .enableIsolateSample = true,
45     .enableTransferSeparation = false,
46     .ssPolarity = ((CY_SCB_SPI_ACTIVE_LOW << CY_SCB_SPI_SLAVE_SELECT0) | \
47     (CY_SCB_SPI_ACTIVE_LOW << CY_SCB_SPI_SLAVE_SELECT1) | \
48     (CY_SCB_SPI_ACTIVE_LOW << CY_SCB_SPI_SLAVE_SELECT2) | \
49     (CY_SCB_SPI_ACTIVE_LOW << CY_SCB_SPI_SLAVE_SELECT3)),
50     .ssSetupDelay = false,
51     .ssHoldDelay = false,
52     .ssInterFrameDelay = false,
53     .enableWakeFromSleep = false,
54     .rxFifoTriggerLevel = 63UL,
55     .txFifoIntEnableMask = 0UL,
56     .txFifoTriggerLevel = 63UL,
57     .txFifoIntEnableMask = 0UL,
58     .masterSlaveIntEnableMask = 0UL,
59 };
60
61 #if defined (CY_USING_HAL)
62 const cyhal_resource_inst_t mSPI_obj =
63 {
64     .type = CYHAL_RSC_SCB,
65     .block_num = 20,
66     .channel_num = 0,
67     .};
68 #endif //defined (CY_USING_HAL)
    
```

The license text in the inset window includes:

```

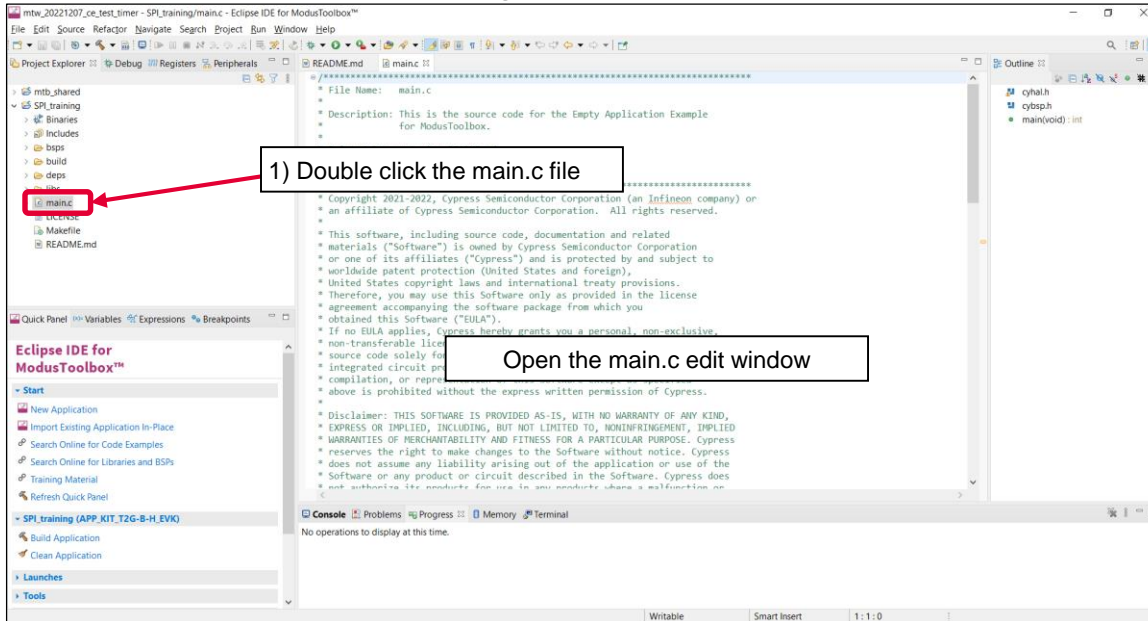
19 *
20 * http://www.infineon.com/legal/TOOLS_LICENSE_2-0
21 *
22 * Unless required by applicable law or agreed to in writing,
23 * distributed under the license is distributed on an
24 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
25 * express or implied. See the License for the specific language governing
26 * permissions and limitations under the License.
    
```

Implementation (contd.)

> Implementation:

This section describes how to implement the configured SPI. This example will implement SPI configuration in the SPI_training project.

– Open main.c in SPI_training project



Implementation (contd.)

- › Add SPI initialization and enable

```

int main(void)
{
    cy_rslt_t result;

    #if defined(CY_DEVICE_SECURE)
        cyhal_wdt_t wdt_obj;

        /* Clear watchdog timer so that it doesn't trigger a reset */
        result = cyhal_wdt_init(&wdt_obj, cyhal_wdt_get_max_timeout_ms());
        CY_ASSERT(CY_RSLT_SUCCESS == result);
        cyhal_wdt_free(&wdt_obj);
    #endif

    /* Initialize the device and board peripherals
    result = cybsp_init();

    /* Board init failed. Stop program execution
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Configure SPI block */
    cy_SCB_SPI_Init(mSPI_HW, &mSPI_config, NULL);

    /* Enable SPI master block */
    Cy_SCB_SPI_Enable(mSPI_HW);

    /* Enable global interrupts
    _enable_irq();

    for (;;)
    {
    }
    }
    
```

There is structure to configure SPI in the cycfg_peripherals.c file

Add SPI initialization function

/* Configure SPI block */
 cy_SCB_SPI_Init(mSPI_HW, &mSPI_config, NULL);

/* Enable SPI master block */
 Cy_SCB_SPI_Enable(mSPI_HW);

Add SPI enable function

You can use the "mSPI_HW" (SCB#2) to specify the hardware

```

const cy_stc_scb_spi_config_t mSPI_config =
{
    .spiMode = CY_SCB_SPI_MASTER,
    .scbNode = CY_SCB_SPI_MOTROLA,
    .scbNode = CY_SCB_SPI_CPMAB_CP0L0,
    .parity = CY_SCB_SPI_PARITY_NONE,
    .dropOnArtyError = false,
    .oversample = 16,
    .rdDataWidth = 8UL,
    .tdDataWidth = 8UL,
    .enableMSBFirst = true,
    .enableInputFilter = false,
    .enableRedundantClock = false,
    .enableIsolateSample = true,
    .enableTransferSeparation = false,
    .spiPolarity = ((CY_SCB_SPI_ACTIVE_LOW << CY_SCB_SPI_SLAVE_SELECT0) | \
        (CY_SCB_SPI_ACTIVE_LOW << CY_SCB_SPI_SLAVE_SELECT1) | \
        (CY_SCB_SPI_ACTIVE_LOW << CY_SCB_SPI_SLAVE_SELECT2)) | \
        (CY_SCB_SPI_ACTIVE_LOW << CY_SCB_SPI_SLAVE_SELECT3));

    .ssSetupDelay = false,
    .ssHoldDelay = false,
    .ssInterFrameDelay = false,
    .enableWakeFromSleep = false,
    .rxfifoTriggerLevel = 63UL,
    .rxfifoIntEnableMask = 0UL,
    .tdfifoTriggerLevel = 63UL,
    .tdfifoIntEnableMask = 0UL,
    .masterSlaveIntEnableMask = 0UL,
};
    
```

```

/* Unless required by applicable law or agreed to in writing, software
 * distributed under the license is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the license for the specific language governing permissions and
 * limitations under the license.
 */

#if defined(CYCFG_PERIPHERALS_H)
#define CYCFG_PERIPHERALS_H

#include "cycfg_notices.h"
#include "cy_scb_spi.h"
#include "cy_sysclk.h"
#endif

#if defined(CY_USING_HAL)
#include "cyhal_hwmgm.h"
#endif

#if defined(CY_USING_IRQ)
#include "cyhal_irq.h"
#endif

#if defined(CY_PLUSPLUS)
extern "C" {
#endif

#define mSPI_HW SCB2
#define mSPI_ENABLED 1U
#define mSPI_HW_SCBB
#define mSPI_IRQ_scb_8_interrupt_IRQ0

extern const cy_stc_scb_spi_config_t mSPI_config;
    
```

Implementation (contd.)

SPI initialization:

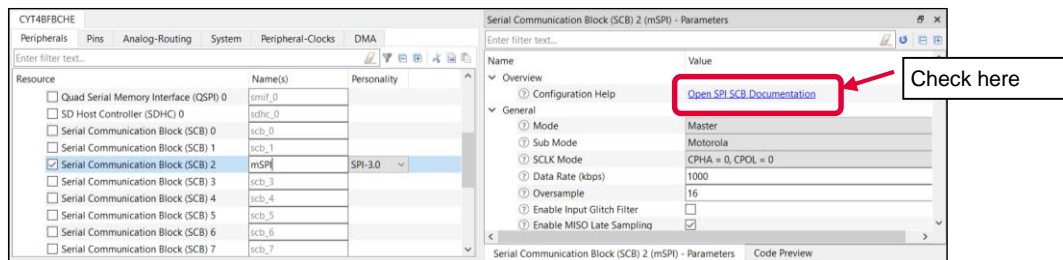
- › Call the [**Cy_SCB_SPI_Init\(\)**](#) function to configure SCB
 - Initializes the SCB for SPI operation
 - Configure SCB with parameters in the *mSPI_config* structure

SPI enable:

- › Call the [**Cy_SCB_SPI_Enable\(\)**](#) function to enable SCB
 - Enable the SCB for SPI
 - Initiate transmission by transferring data from DMA to TX FIFO

Other functions:

- › Check the following for more information



The screenshot shows the PSoC Designer interface for a CY14B8C4E device. The 'Peripheral-Clocks' tab is active, and the 'Serial Communication Block (SCB) 2 (mSPI) - Parameters' window is open. In the 'Parameters' window, the 'Open SPI SCB Documentation' link is highlighted with a red box. A callout box with an arrow points to this link and contains the text 'Check here'.

Name	Value
Mode	Master
Sub Mode	Motorola
SCLK Mode	CPHA = 0, CPOL = 0
Data Rate (kbps)	1000
Oversample	16
Enable Input Glitch Filter	<input type="checkbox"/>
Enable MISO Late Sampling	<input checked="" type="checkbox"/>

References

Datasheet

- › [CYT4BF datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family](#)

Architecture Technical reference manual

- › [TRAVEO™ T2G automotive body controller high family architecture technical reference manual](#)

Registers Technical reference manual

- › [TRAVEO™ T2G Automotive body controller high registers technical reference manual](#)

PDL/HAL

- › [PDL](#)

- › [HAL](#)

Training

- › [TRAVEO™ T2G Training](#)

Revision History

Revision	ECN	Submission Date	Description of Change
**	7849954	12/19/2012	Initial release

Important notice and warnings

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2022-12

**Published by
Infineon Technologies AG
81726 Munich, Germany**

**© 2022 Infineon Technologies
AG.
All Rights Reserved.**

**Do you have a question about
this document?**

Go to:
www.infineon.com/support

**Document reference
002-36744 Rev. ****

IMPORTANT NOTICE

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenhheitsgarantie”).

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (www.infineon.com).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.



Part of your life. Part of tomorrow.