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 CYT4BFBCCH device is used in this code example
- Board:
  - TRAVEO™ T2G KIT_T2G-B-H_EVK board is used for testing
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
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

Window for selecting peripheral and channel number.

Window for setting the operating parameters of selected peripherals.
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
- 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
- See "CAN FD" application for operation
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
CAN FD configuration (contd.)

› Configure Clock (System):
  4) Select **CLK_HF2**
  5) Select **CLL_PATH2** as “Source Clock”
  6) Set “Divider” to “1”
  7) Ensure that the frequency is set to 200 MHz

![Diagram showing the configuration process]
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

![Image of configuration process]

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”
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%”
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%”
Confirm configuration result

- You can check the configuration result in the “Code Preview” tab of the Device Configurator.
CAN FD configuration (contd.)

› Close Device configurator:
  - Click the “Save” button after completing all settings, then close the “Device configurator”
    1) Click “Save” button
    2) Close “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:

```
cycfg_peripherals.h

cycfg_peripherals.c
```
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

1) Double click the main.c file

Open the main.c edit window
Implementation (contd.)

- Add include file

```c
#include <stdio.h>
#include "cy_pdl.h"
#include "cycfg.h"
#include "cybsp.h"
#include "cy_retarget_io.h"
```

Add include file in the main.c
Implementation (contd.)

› Add CAN FD initialization

```c
int main(void)
{
    cy_rslt_t result;
    cy_en_canfd_status_t status;
    /* Initialize the device and board per
     * result = cybsp_init();
    if (result != CY_RSLT_SUCCESS)
    { CY_ASSERT(0); }
    */

    status = Cy_CANFD_Init(CANFD_HW, CAN_HW_CHANNEL, &CANFD_config &canfd_context);
    if (status != CY_CANFD_SUCCESS)
    { CY_ASSERT(0); }

    /* Initialize CANFD Channel */

    status = Cy_CANFD_UpdateAndTransmitMsgBuffer(CANFD_HW, CAN_HW_CHANNEL,
                                               &CANFD_config &canfd_context);
}
```

Use this structure to configure CAN FD in the cycfg_peripherals.c file

Add CAN FD initialization function

Add CAN FD transmit function
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
UART configuration (contd.)

› **Configure Clock (System):**

4) Select **CLK_HF2**
5) Select **CLL_PATH2** as “Source Clock”
6) Set “Divider” to “1”
7) Ensure that the frequency is set to 196 MHz

![Diagram showing the selection of clocks and configuration steps]
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.

![Diagram showing steps for configuring clock settings](image-url)
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
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

5) Set “Value” of “Connections” parameters
   - Clock divider: 8-bit Divider 0
   - Used ports:
     - Tx : SCB3_TX (P13.1)
     - Rx : SCB3_RX (P13.0)
6) Check the Actual Baud Rate and update it for your device
UART configuration (contd.)

» Confirm configuration result
  - You can check the configuration result in the “Code Preview” tab of the Device Configurator
UART configuration (contd.)

› **Close Device Configurator:**
  
  - Click the “Save” button after completing all settings, and then close the “Device configurator”

  1) Click “Save” button
  2) Close “Device configurator”

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

  ![Diagram of Device Configurator window with highlighted options]

  ![Screenshot of Errors/Tasks dialog box with highlighted button]

Click
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:
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

```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

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

Add UART initialization function

Add UART enable function

Add UART control function, if necessary
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
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 case, it changes to “SPI_training”

5) Click Create and start application creation
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

1) Click

2) Select

3) Set to 192.000

4) 192 MHz
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
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
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
SPI configuration (contd.)

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

6) Set “Value” of “Data Configuration” parameters
   - Tx/Rx data width = 8 bits
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
   - Set SS0 polarity to Active Low
   - Set Clock to 8 bit Divider 1 clk
   - Set used port
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.
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

1) Click “Save” button
2) Close “Device configurator”
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:
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

1) Double click the main.c file

Open the main.c edit window
Implementation (contd.)

› Add SPI initialization and enable

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

Add SPI initialization function

Add SPI enable function

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>ECN</th>
<th>Submission Date</th>
<th>Description of Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>7849954</td>
<td>12/19/2012</td>
<td>Initial release</td>
</tr>
</tbody>
</table>
Important notice and warnings

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

IMPORTANT NOTICE

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

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.