AN14711

Using CAN2CAN, CAN2ETH and ETH2CAN Features of FlexLLCE on S32Z/E

Rev. 1.0 — 4 June 2025 Application note

Document information

Information	Content
Keywords	FlexLLCE, Gateway, CAN2ETH, ETH2CAN, CAN2CAN, IEEE1722, Network Accelerator
Abstract	FlexLLCE is the gateway-oriented submodule of S32Z/E that enables off-loading of the host cores. This document describes how to execute the bundled CAN2ETH/ETH2CAN example project and CAN2CAN example project, whilst describing how to change the setup for building applications.

1 Introduction

This application note is complementary to the Integration Manual for S32Z/E FLEXLLCE_FW Driver and the User Manual for S32Z/E FLEXLLCE_FW Driver. (You can see those manuals under the installed FlexLLCE package folder S32ZE_FLEXLLCE_0.9.3_D2407/eclipse/plugins/ FlexLLCE_FW_TS_T31D53M9I3R0/doc) To understand the CAN2CAN, CAN2ETH and ETH2CAN features, executing the example project would be the good way. The purpose of this document is to introduce how to execute and modify the example projects.

- **CAN2CAN :**

The S32Z/E FlexLLCE FW has the routing feature between CAN channels. Once the configuration for the feature is done, the CAN2CAN feature does not need the host core's intervention at runtime.

From the example projects bundled in current FlexLLCE FW package ver0.9.3, this document focuses on the project "FlexLLCE_example_routing_S32Z2XX_R52"

This is the example project for CAN2CAN routing using internal loopback scenario in default. The default setup is not using external wire communication hence no exposure of the actual CAN frames. This doc introduces the steps to change the project to execute external wired loopback of CAN.

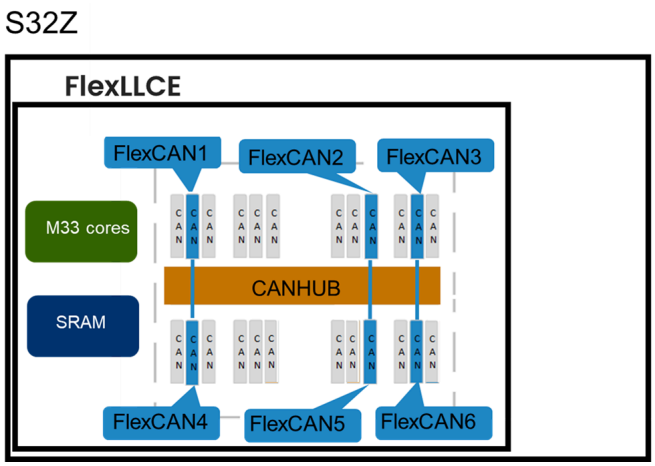

Figure 1. Default internal CAN bus config of the CAN2CAN example project

As the above figure, the default setting of CAN2CAN example is using internal bus in the CANHUB hence no external CAN pins are connected to the FlexCAN channels, which is unable to observe the CAN frames.

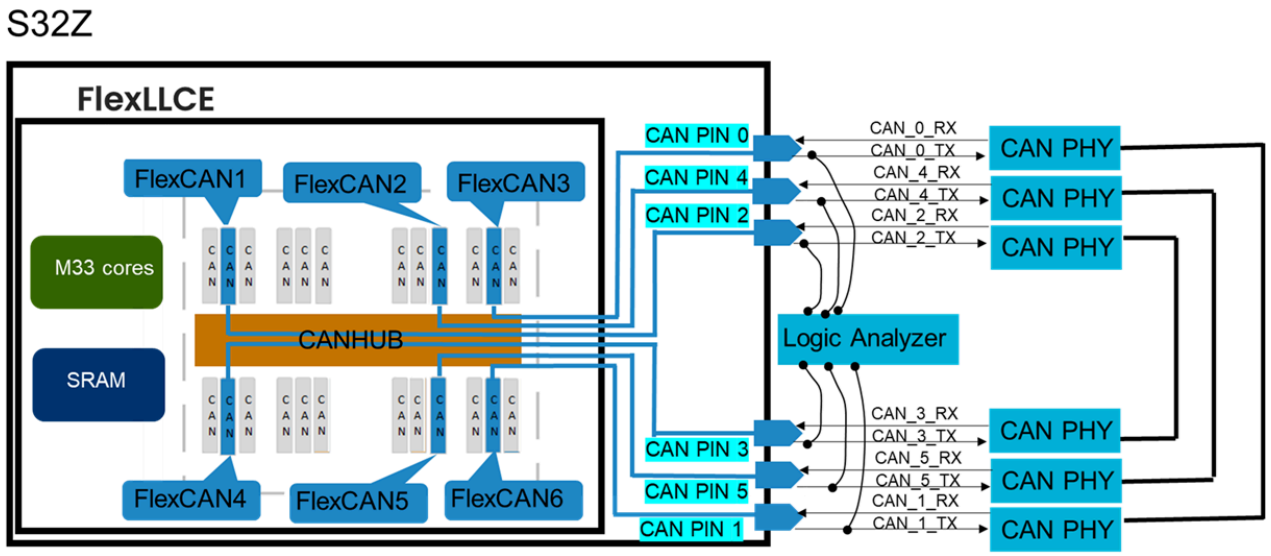

Figure 2. External loopback config to observe the internal CAN bus activity for CAN2CAN example

With the above external loopback configuration, you can observe the CAN frame behavior if you probe CAN_TX of each CAN channels with a logic analyzer.

This document introduces how you can build those config and what you will see the CAN2CAN behavior with those configs.

- **CAN2ETH / ETH2CAN:**

The S32Z/E FlexLLCE FW has the routing feature between CAN and Ethernet. Once the configuration for these features is done, these features do not need the host core's intervention at runtime.

From the example projects bundled in current FlexLLCE FW package ver0.9.3, this document focuses on the project "FlexLLCE_example_can2eth_ eth2can_S32Z2XX_R52"

This is the example project for CAN2ETH and ETH2CAN routing using internal loopback scenario in default. The default setup is not using external wire communication hence no exposure of the actual CAN/ETH frames. This doc introduces the steps to change those projects to execute external wired loopback of CAN, and also to connect Ethernet to the PC, which enables observation of the actual CAN/ETH frame behavior.

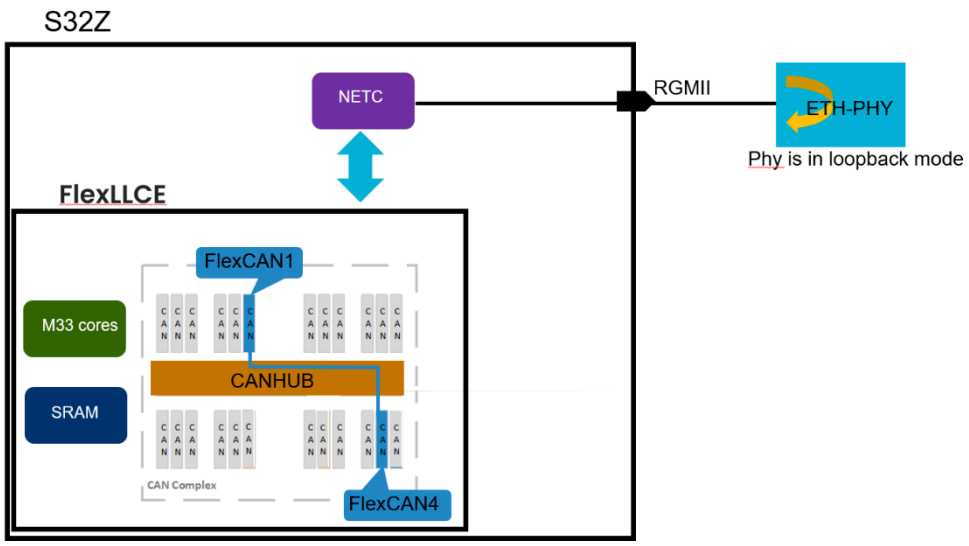

Figure 3. Default config of the CAN2ETH example project

As shown in the above figure, in the sample application, the internal bus is used in the CANHUB hence no external CAN pins are connected to the FlexCAN channels. And the Ethernet PHY on the EVB is configured to the loopback mode hence no frame exposure to the RJ45 Ethernet connector. This doc guides how to change this default config to the below configuration which is not using loopback/internal bus and able to observe the CAN/ETH frames.

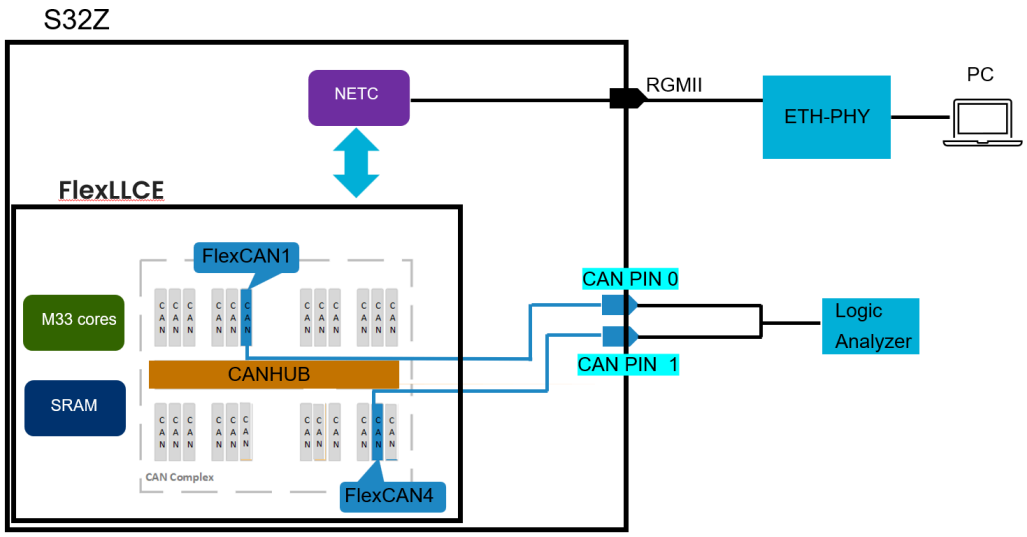

Figure 4. Modified CAN2ETH config

With the above configuration, you can observe the Ethernet frame on the packet capture tool on the PC, and the CAN frames on the Logic Analyzer.

2 How to setup the FlexLLCE FW package

2.1 Download the required NXP software packages

To run the example projects of LLCE FW package, you should download not only FlexLLCE FW package itself but also the NXP Real Time Driver software (aka RTD). Both FlexLLCE FW and RTD can be downloaded from NXP Software downloading site. As for FlexLLCE FW, download version 0.9.3 as below.

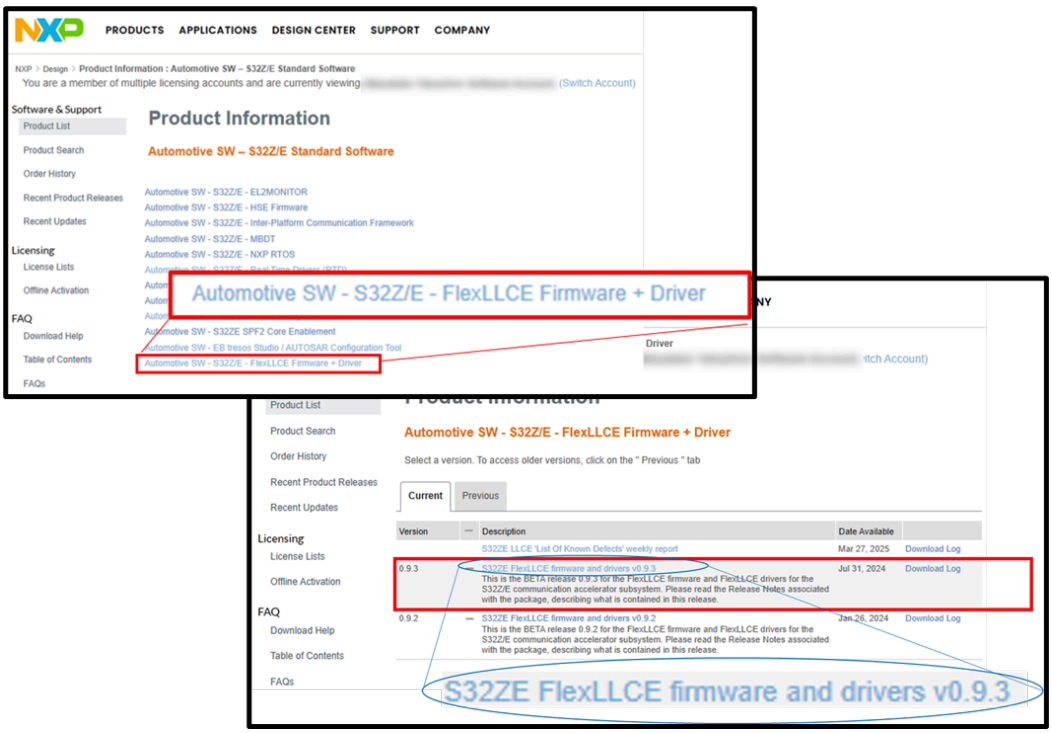

Figure 5. Download LLCE FW Ver0.9.3

As for the RTD, download ver 2.0.0. In this document, tested 2.0.0 HotFix01.

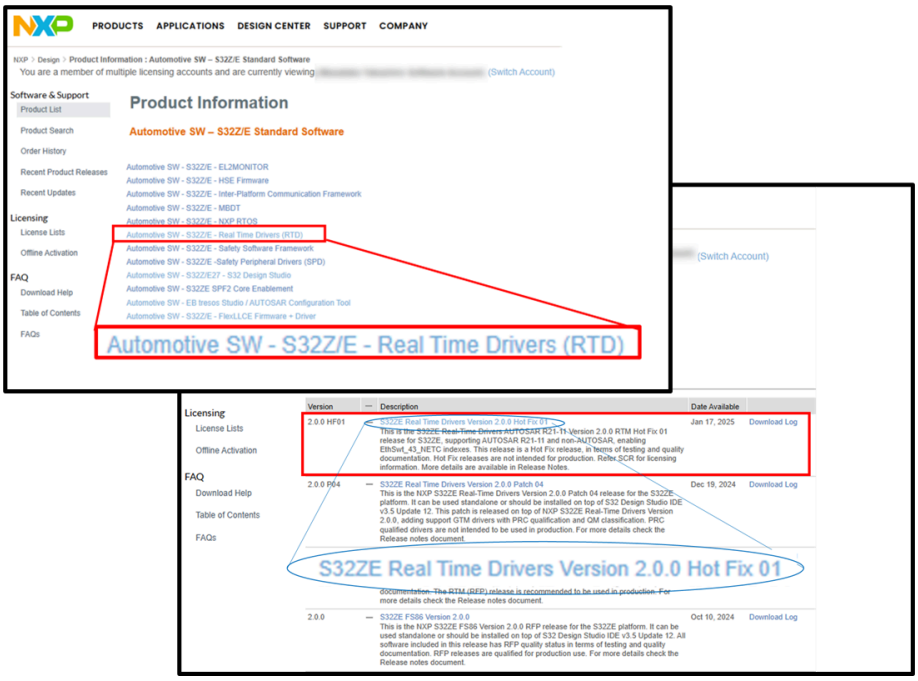

Figure 6. Download RTD ver2.0.0

And, if you don't have EB-Tresos Studio, download it also.

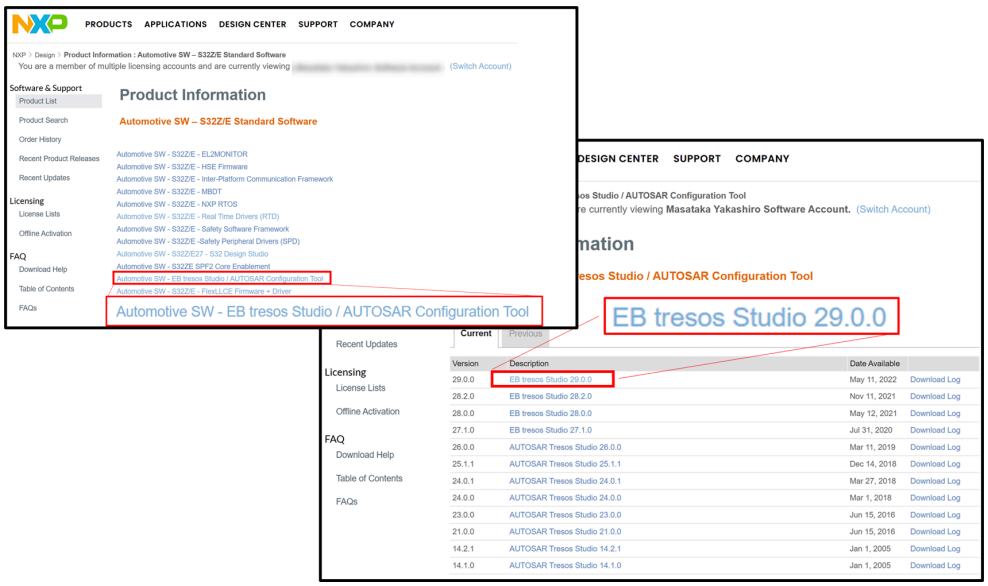

Figure 7. Download EB-Tresos Studio

2.2 Install the downloaded software

After EB-Tresos studio installation, execute the downloaded "S32ZE_FLEXLLCE_0.9.3_D2407.exe" to install FlexLLCE software package and "SW32ZE_RTD_R21-11_2.0.0_HF01_D2501.exe" to install RTD respectively.

Under the FlexLLCE software package installed folder, you will see the CAN2CAN example project folder and the CAN2ETH/ETH2CAN example project folder as below.

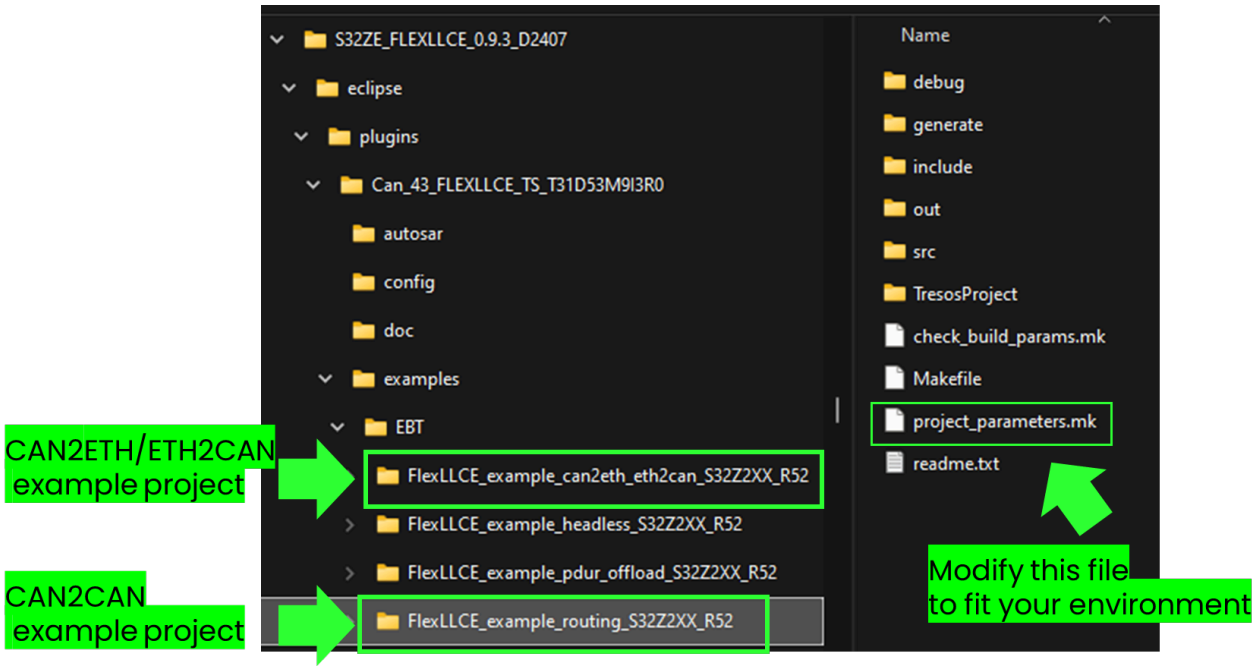

Figure 8. Example project folder

In both CAN2CAN and CAN2ETH/ETH2CAN projects, the file “project_parameter.mk” contains the folder preferences. You should edit it to fit your environment. The minimum required modifications are 2 points as in the following figure. One is the gcc v10.2 location. If you are already installed the NXP’s IDE environment S32DS 3.6, the location would be “C:/NXP/S32DS.3.6.0/S32DS/build_tools/gcc_v10.2/gcc-10.2-arm32-eabi”. The other one is the RTD plugins location. Make sure this would be eclipse/plugins under your RTD installed folder.

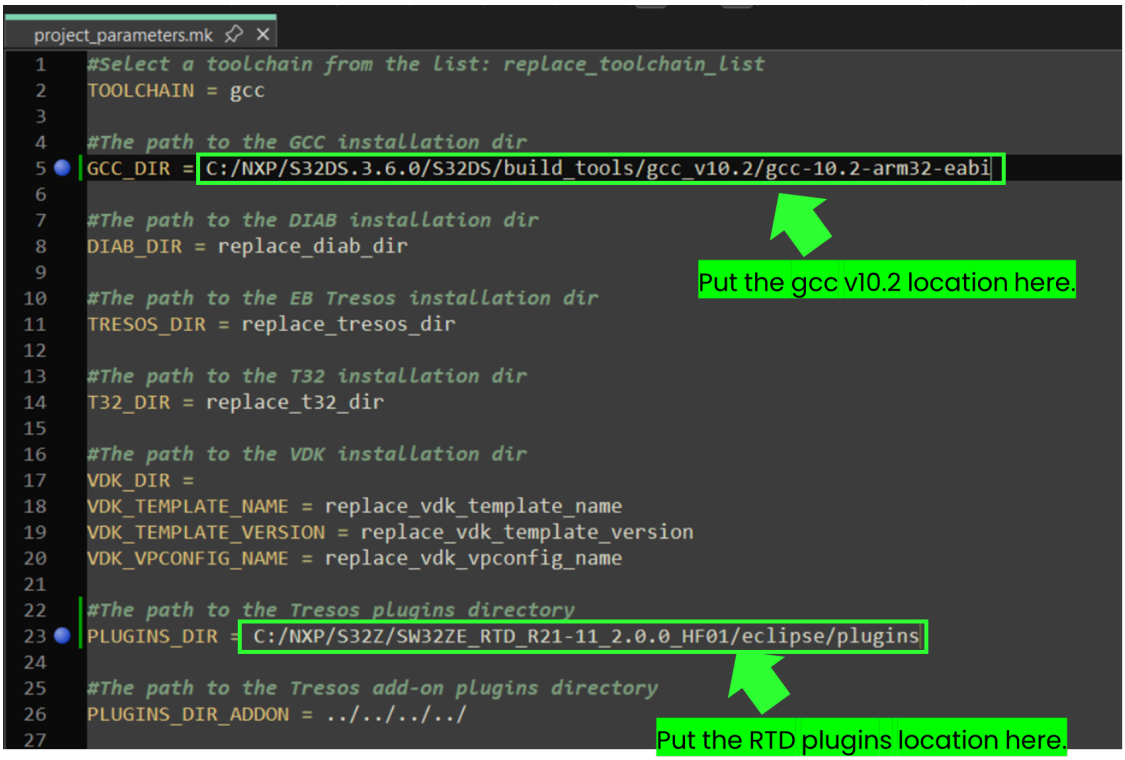

Figure 9. Customize project_parameters.mk

3 CAN2CAN: Build and Run the project with default config

3.1 Generate the Tresos config

The default Tresos config is located under the example project as shown below figure.

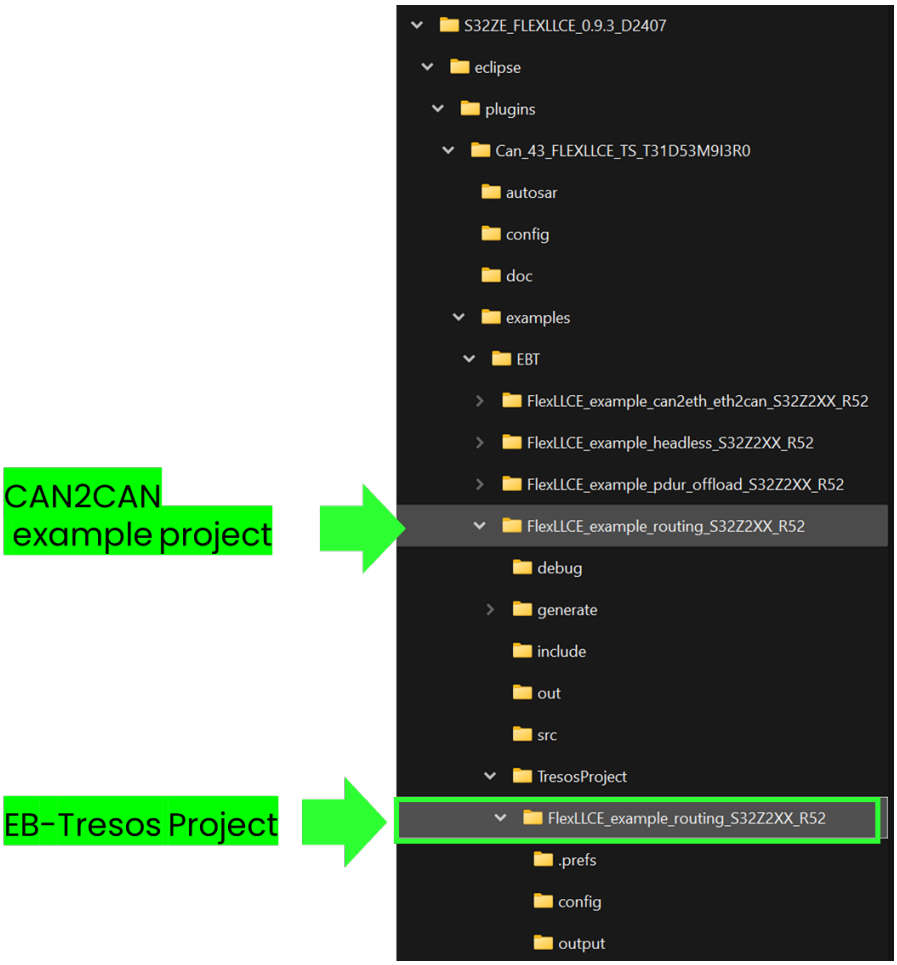

Figure 10. Tresos config location

Start your EB-Tresos and import the EB-Tresos Project as below.

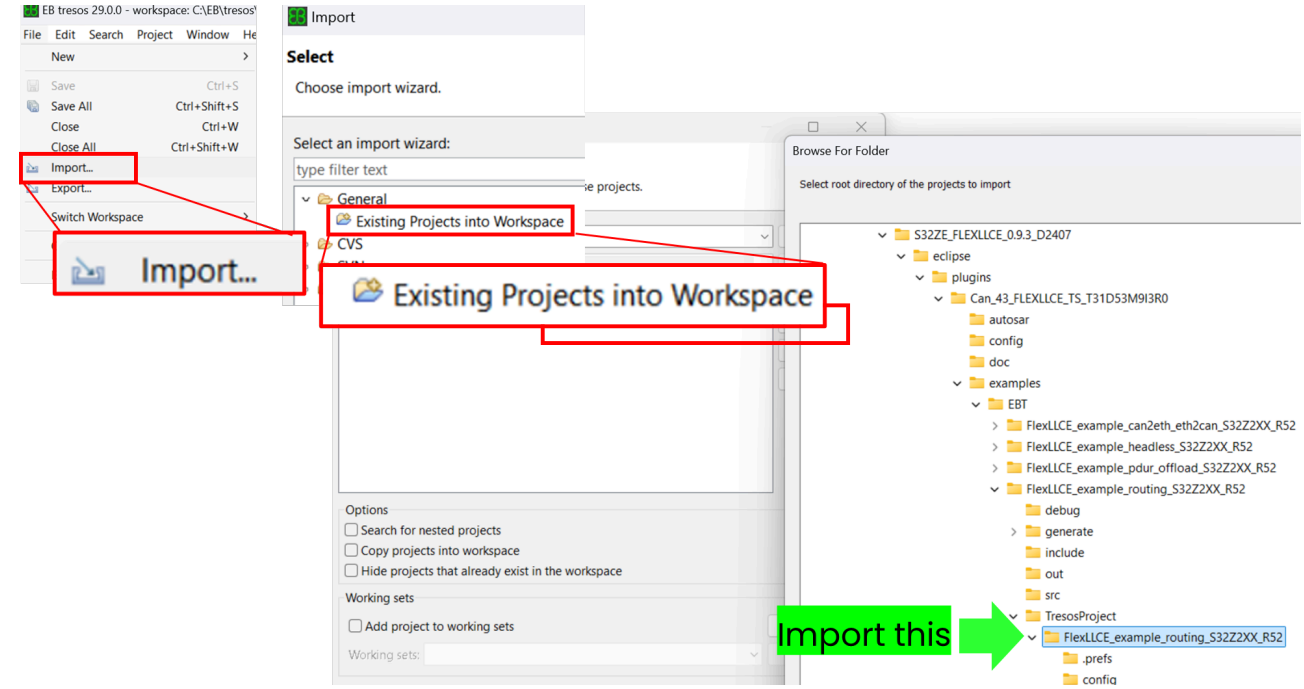

Figure 11. Import EB-Tresos project

Then, generate it as below figure.

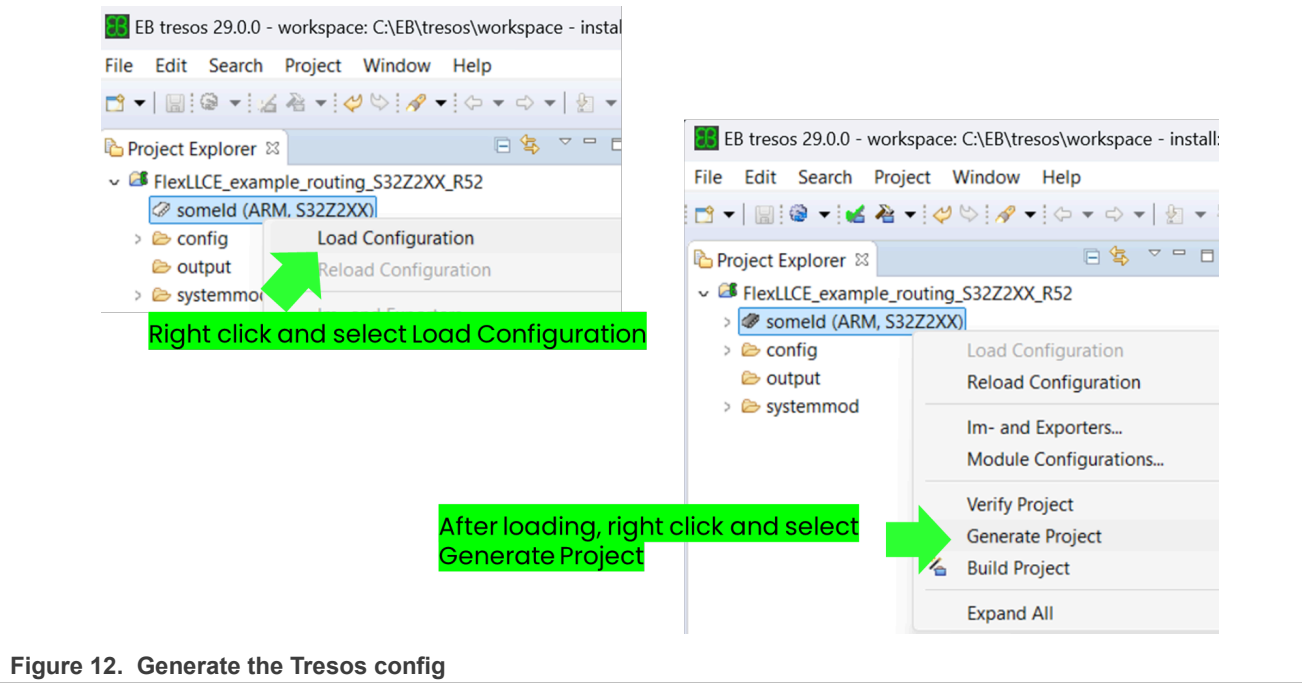

Figure 12. Generate the Tresos config

3.2 Build the example project with default config

On the Linux emulated environment such as Cygwin (https://cygwin.com/), move to the CAN2CAN example project folder and run the command “make build”. Optionally you can use -j command for multi thread processing, as shown in the following figure.

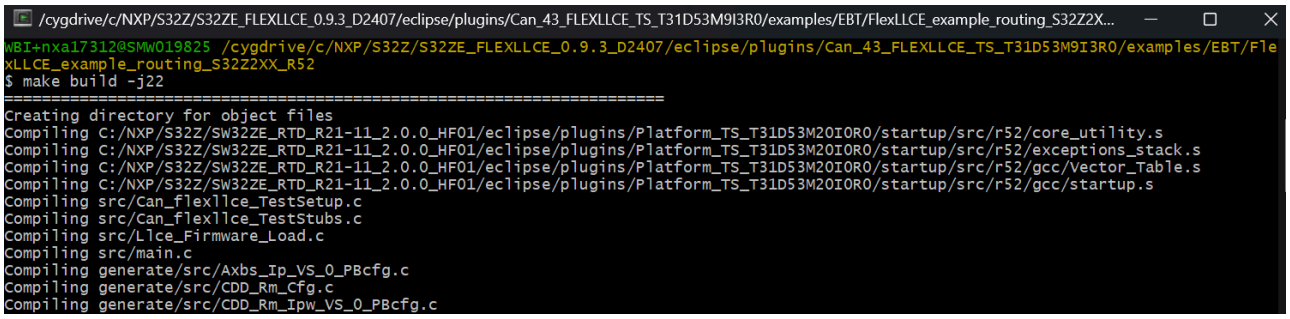

Figure 13. make

Then, you will get the elf file under the folder “out”

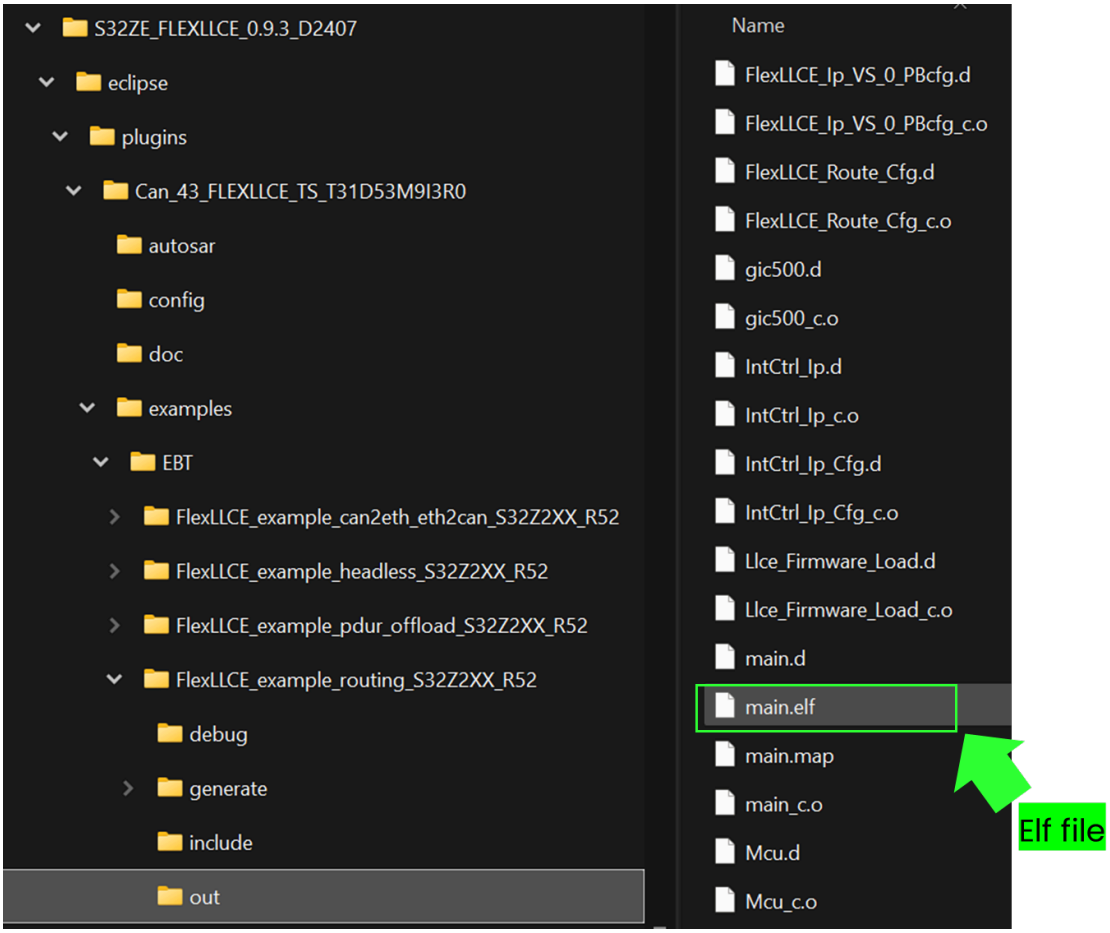

Figure 14. Built Elf file

3.3 Run the example with default config

Then, connect your Lauterbach debugger to the S32Z EVB. Start your TRACE32 and change the directly to “debug” folder under the CAN2CAN example project folder. Then run script “run.cmm”, which is loading the build elf file and run.

Figure 15. run.cmm

On TRACE32, put the following variables to the var.watch window.

CanIf_TxConfirmCnt : The number of Tx successful CAN frame

CanIf_RxIndicationCnt : The number of Rx successful CAN frame

Insert a brakepoint at line 85, which is just after testing Tx/Rx of all FlexCAN instances.

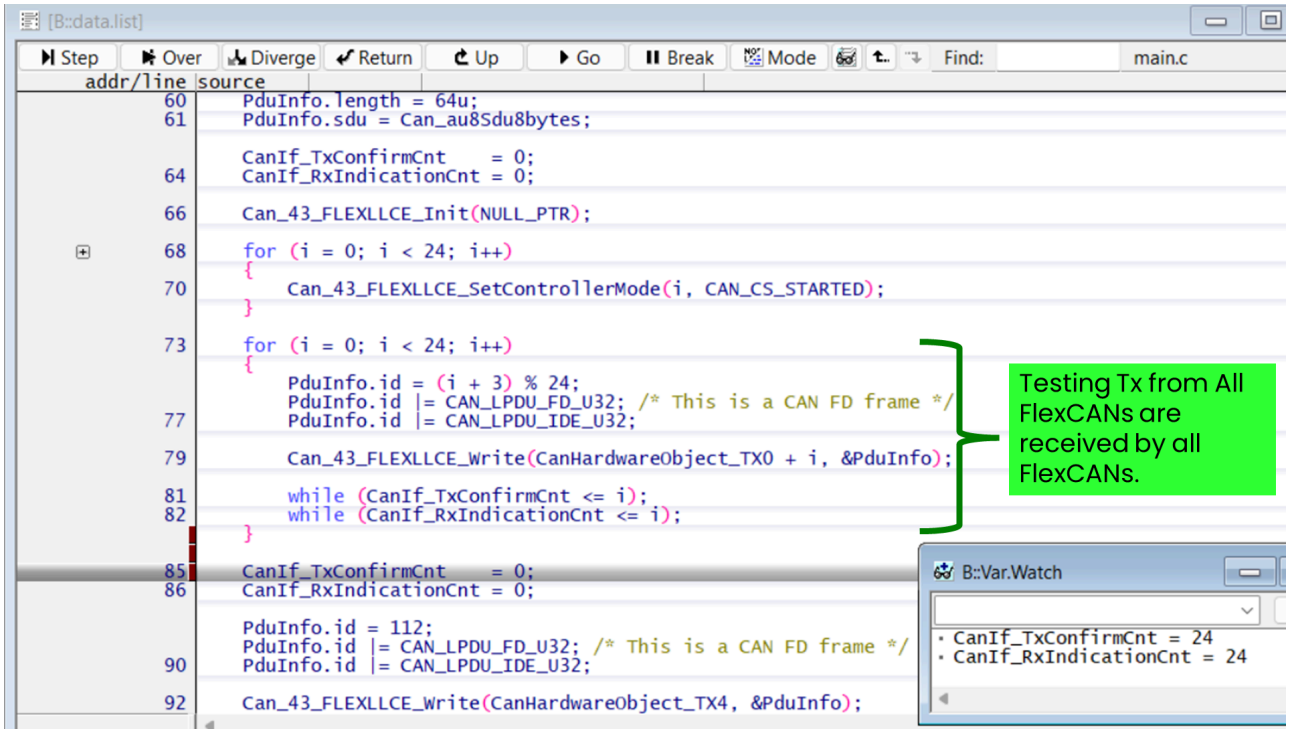

Figure 16. Brakepoint after testing Tx/Rx of all FlexCAN instances

The variable CanIf_TxConfirmCnt and CanIf_RxIndicationCnt are both 24. This is showing the all 24 FlexCAN instances successfully sent and received CAN frames via internal CAN bus of CANHUB.

If you look into the config on EB-Tresos studio GUI, you will find the CAN HUB configuration (configured on the Rm module) is connecting the FlexCAN once every 3 instances. FlexCAN0,3,6…21 are connected to the internal CAN bus 24. FlexCAN1,4,7…22 are connected to the internal CAN bus 25. FlexCAN2,5,8…23 are connected to the internal CAN bus 26. And the CanHardwareObject config in Can_43_FLEXLLCE module has the CanHarwareObject_RXn config which configures Frame ID n is to be received by FlexCAN instance n.

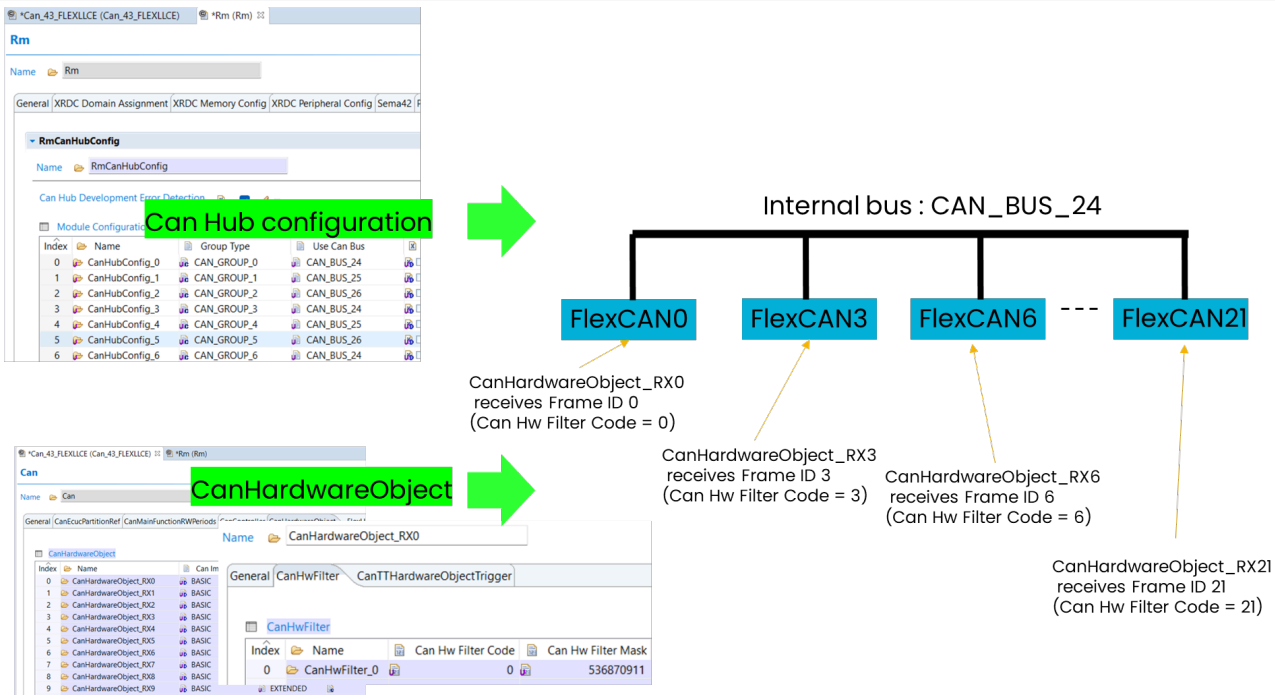

Figure 17. Default Tresos config

With above config, the The variable CanIf_TxConfirmCnt and CanIf_RxIndicationCnt would be both 24 when the brakepoint is at line 85.

Then, let's proceed to the CAN2CAN testing part. This original config is supposed to perform CAN2CAN as follows. If the FlexCAN_1 receives the CAN /FD frame with extended ID 112, it would be routed to FlexCAN_2 and FlexCAN_3. If the FlexCAN_1 receives the CAN /FD frame with extended ID 115, it would be routed to FlexCAN_2 and also notified the RX to the host core R52.

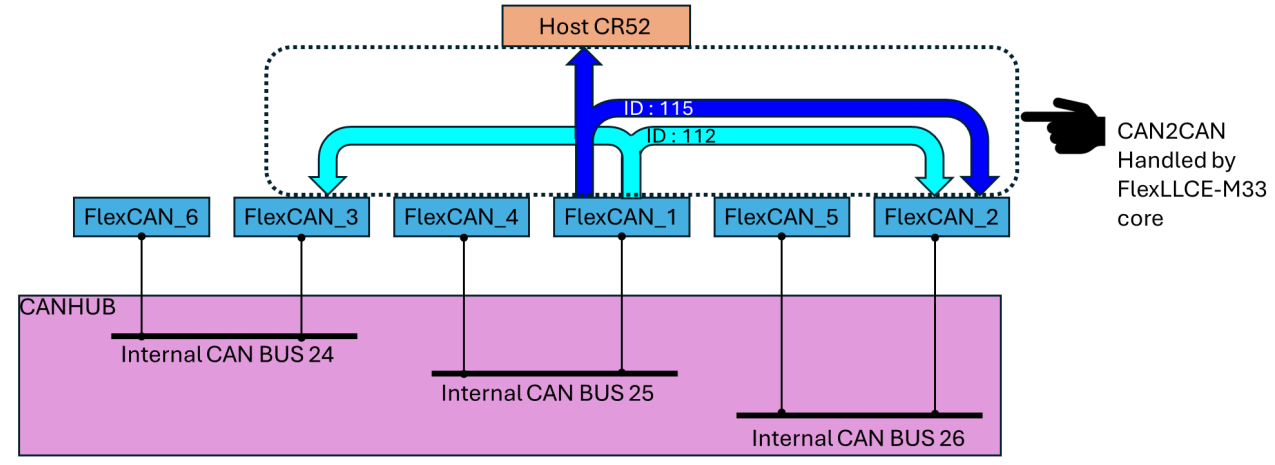

Figure 18. Default CAN2CAN config

- **Frame ID #112 : 2 Destination channels CAN2CAN without Host Reception**

When the brakepoint is at line 97 (as shown in the following figure), it's just after sending one extended CANFD frame with ID=112.

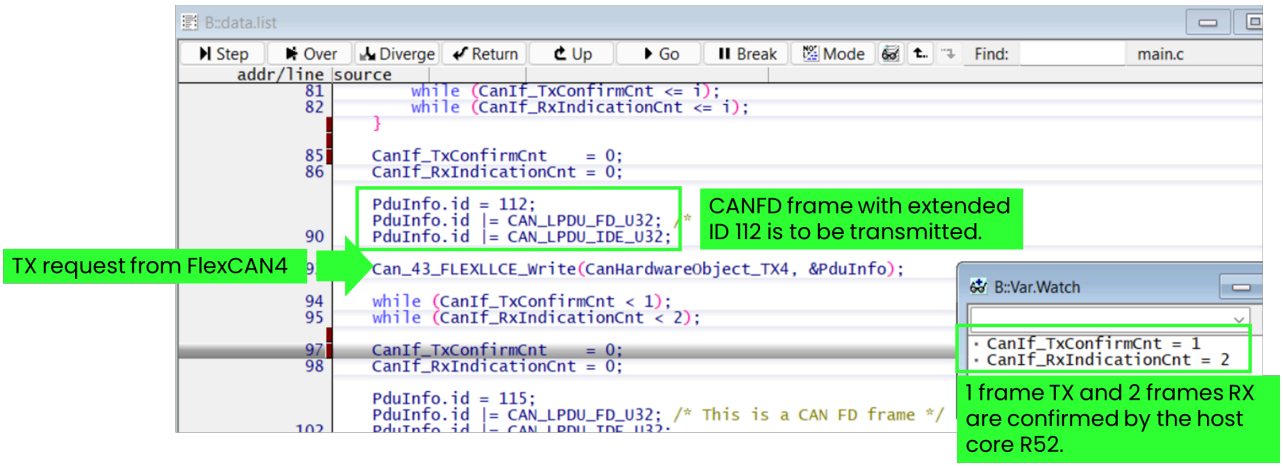

Figure 19. Brakepoint after CAN2CAN (Frame ID=112)

In the above snippet, CANFD frame with extended ID 112 is sent from FlexCAN4 in one time. This frame would be received by FlexCAN1 and it performs CAN2CAN. The var,watch window is showing that 1 TX and 2 RX are confirmed by the host core R52.

The corresponding hardware object for this CAN2CAN routing is configured as CanHardwareObject_RX112_1. This is for Reception of ID #112 on FlexCAN_1 as follows.

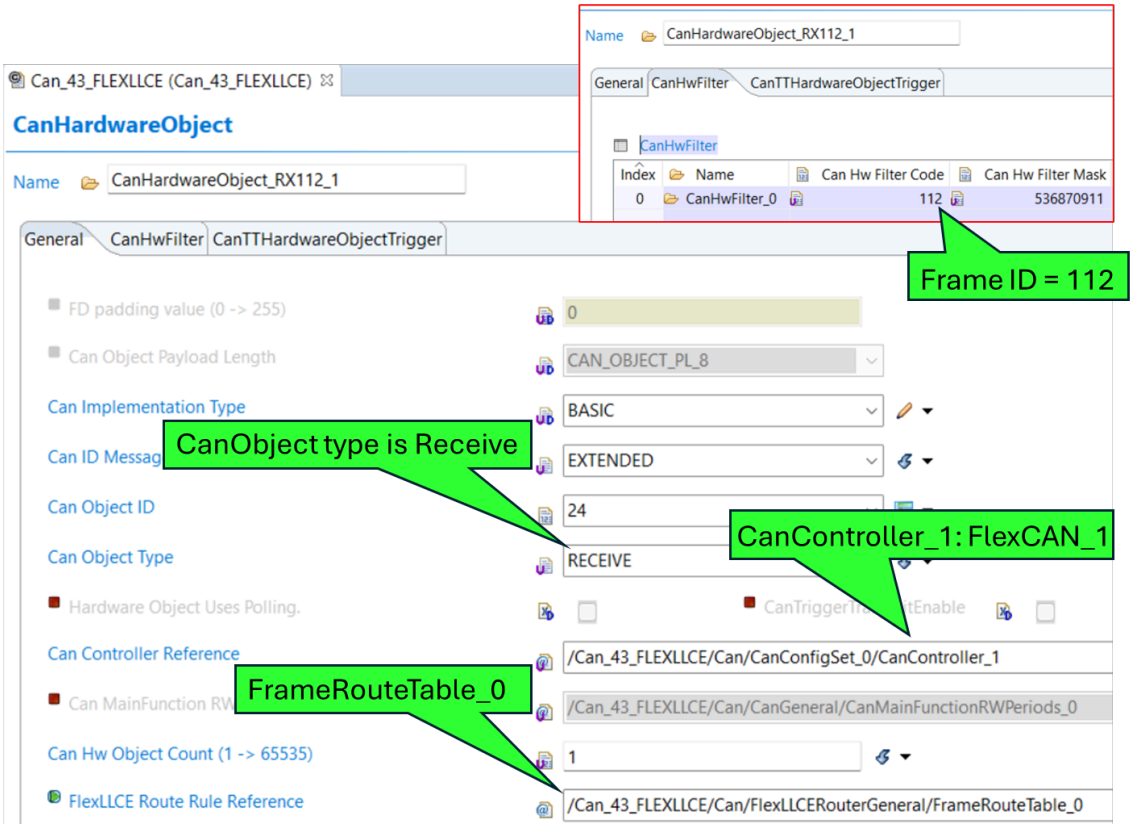

Figure 20. CanHardwareObject_RX112_1

Since the FlexLLCE Route Rule Reference is configured to FrameRouteTable_0, this Hardware Object is not handled as normal reception but handled as internal routing. In case of the internal routing, the host

does not need the intervention, hence the notification to the host for RX is configurable. If you look into the FrameRouteTable_0 config, you will see the Host Receive Enable is unchecked in this case as below. This means the Host Receive Enable is No, hence the host core is not notified at the #112 frame reception on FlexCAN_1.

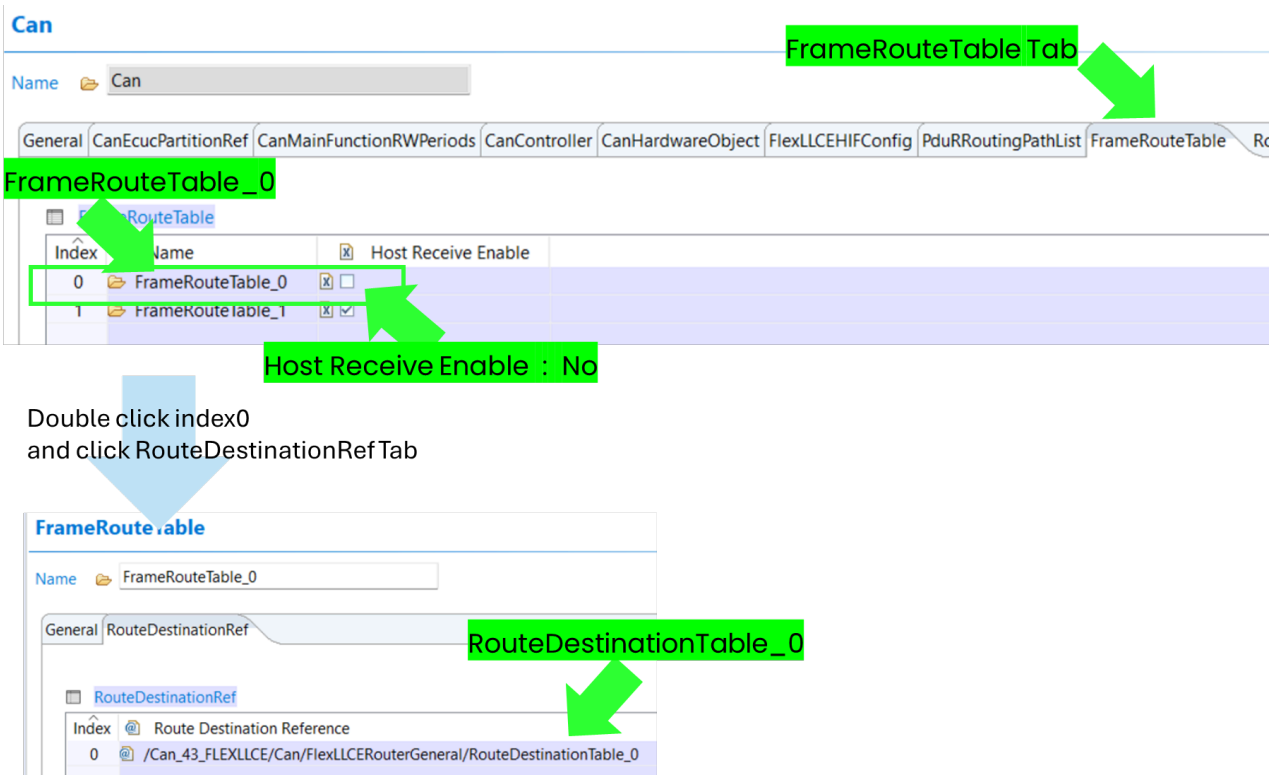

Figure 21. FrameRouteTable_0

In the above figure, if you double click the index0 to open FrameRouteTable_0 config window, and click RouteDestinationRef tab, you will see the FrameRouteTable_0 refers the RouteDestinationTable_0.

Then, if you select the RouteDestinationTable Tab, you will see the RouteDestinationTable_0 is configured to enable CAN2CAN. And the destination CAN channel is configured to FlexCAN_2 and FlexCAN_3 as below figure.

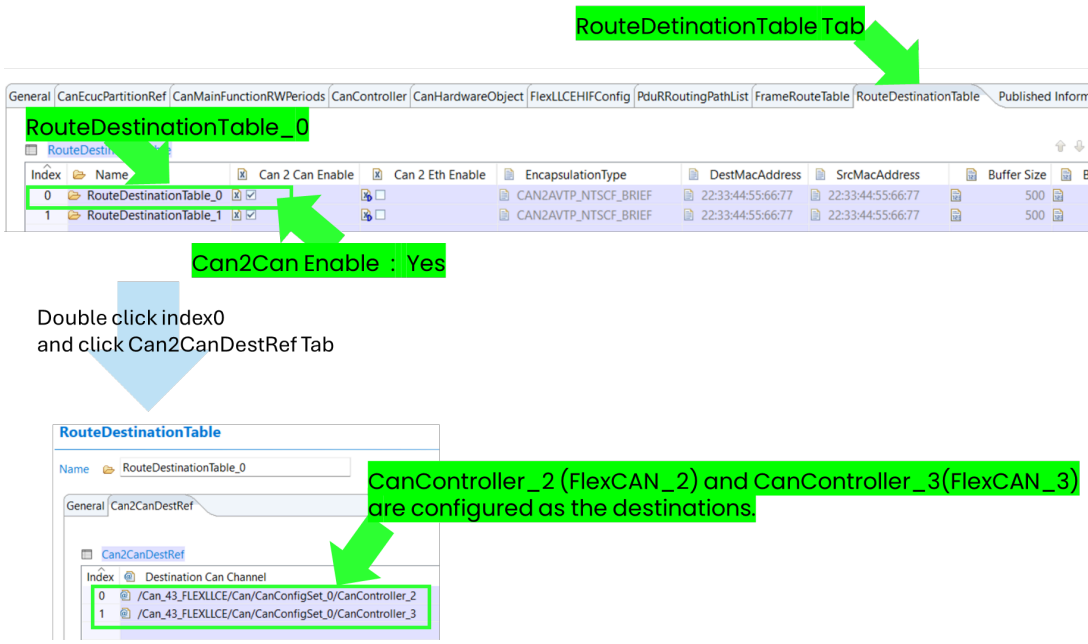

Figure 22. RouteDestinationTable_0

Following diagrams are showing each steps of this CAN2CAN routing.

Below is showing the FlexCAN_1 is receiving the CANFD Frame with ID 112(Extended). After the Tx, the Host Core R52 receives the TxAck notification. Then CanIf_TxConfirmationCnt is incremented by 1.

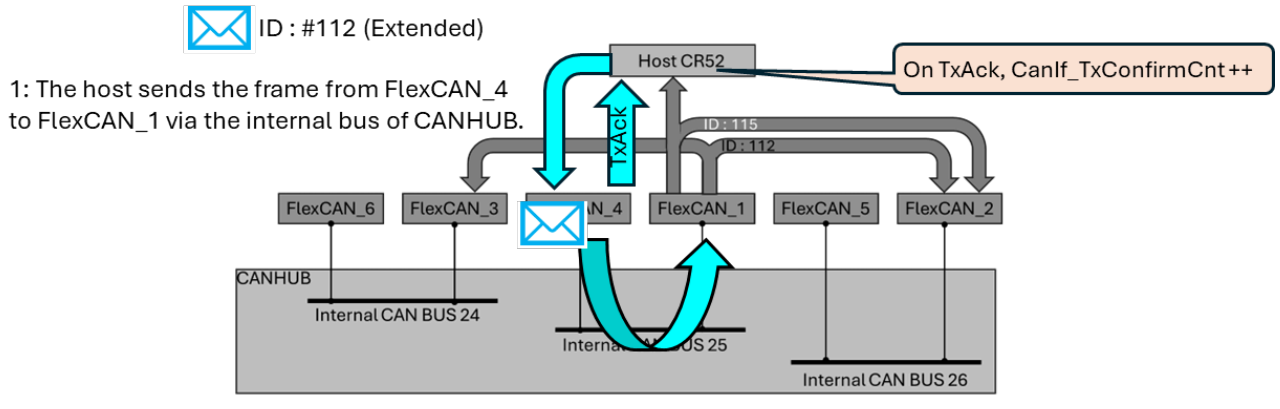

Figure 23. CAN2CAN (Frame #112) step 1

Then, as below, CAN2CAN routing would be executed to FlexCAN_2 & 3 as configured.

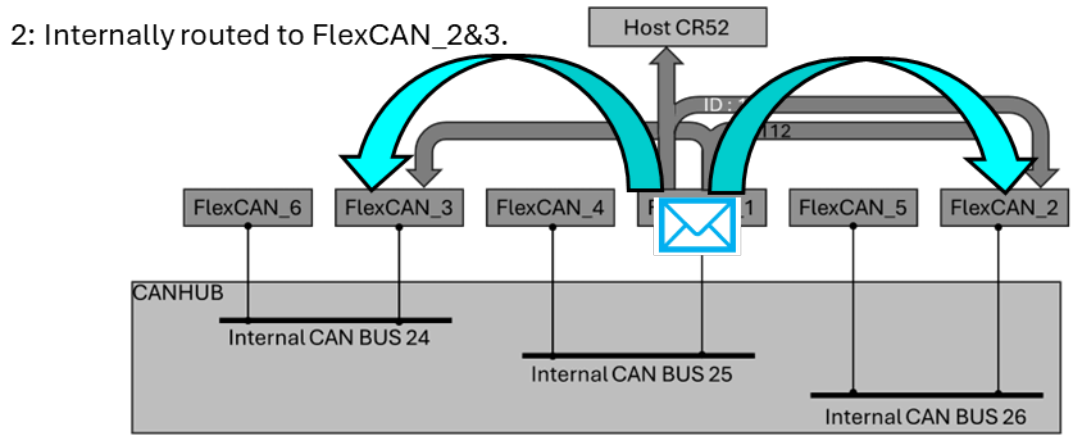

Figure 24. CAN2CAN (Frame #112) step 2

Then, as below, the destination CAN channel (FlexCAN_2 & 3) would send the routed CAN/FD frame via the internal bus 26 & 24 respectively.

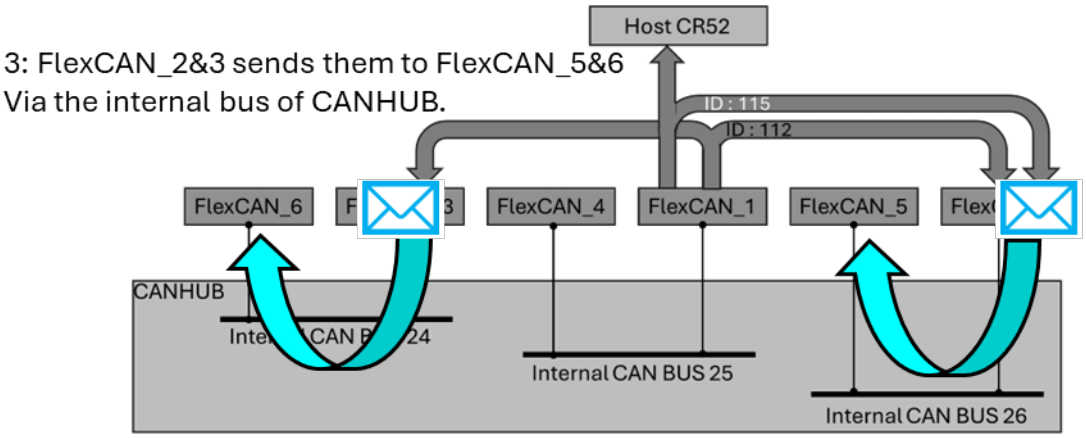

Figure 25. CAN2CAN (Frame #112) step 3

Then, as below, the FlexCAN_5 & 6 receives the frames.

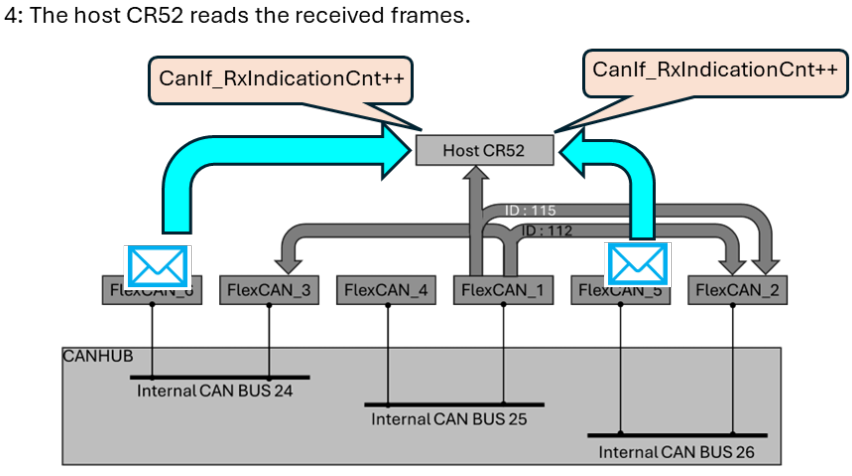

Figure 26. CAN2CAN (Frame #112) step 4

Those are configured in CanHardwareObject_RX112_2 & 3 respectively as below hence they are notified to the Host core and CanIf_RxIndicationCnt is incremented twice.

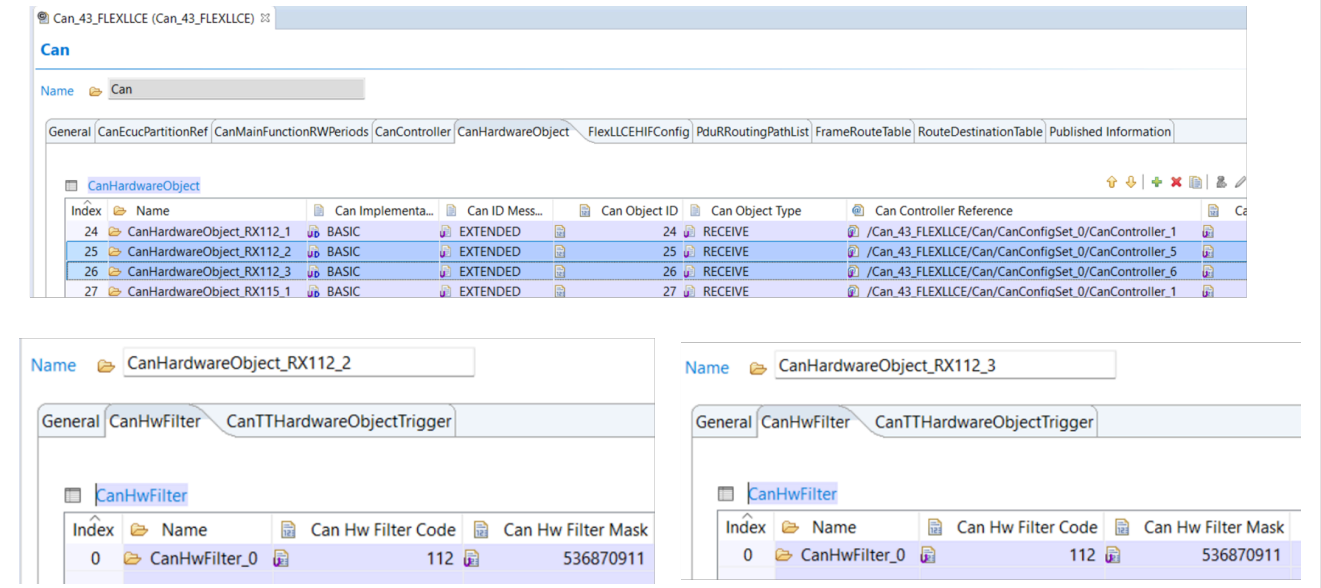

Figure 27. CanHardwareObject for normal RX

- **Frame ID #115 : One Destination channel CAN2CAN with Host Reception**

When the brakepoint is at line 110, it's just after sending one extended CANFD frame with ID=115.

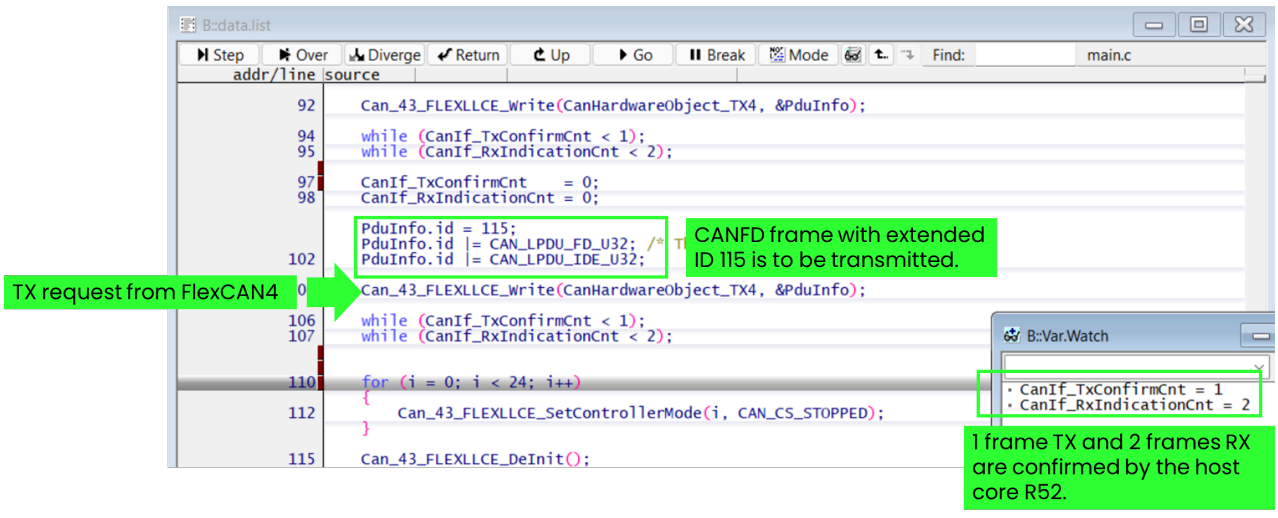

Figure 28. Brakepoint after CAN2CAN (Frame ID=115)

In the above snippet, CANFD frame with extended ID 115 is sent from FlexCAN4 in one time. This frame would be received by FlexCAN1 and it performs CAN2CAN. The var.watch window is showing that 1 TX and 2 RX are confirmed by the host core R52.

The corresponding hardware object for this CAN2CAN routing is configured as CanHardwareObject_RX115_1. This is for Reception of ID #115 on FlexCAN_1 as follows.

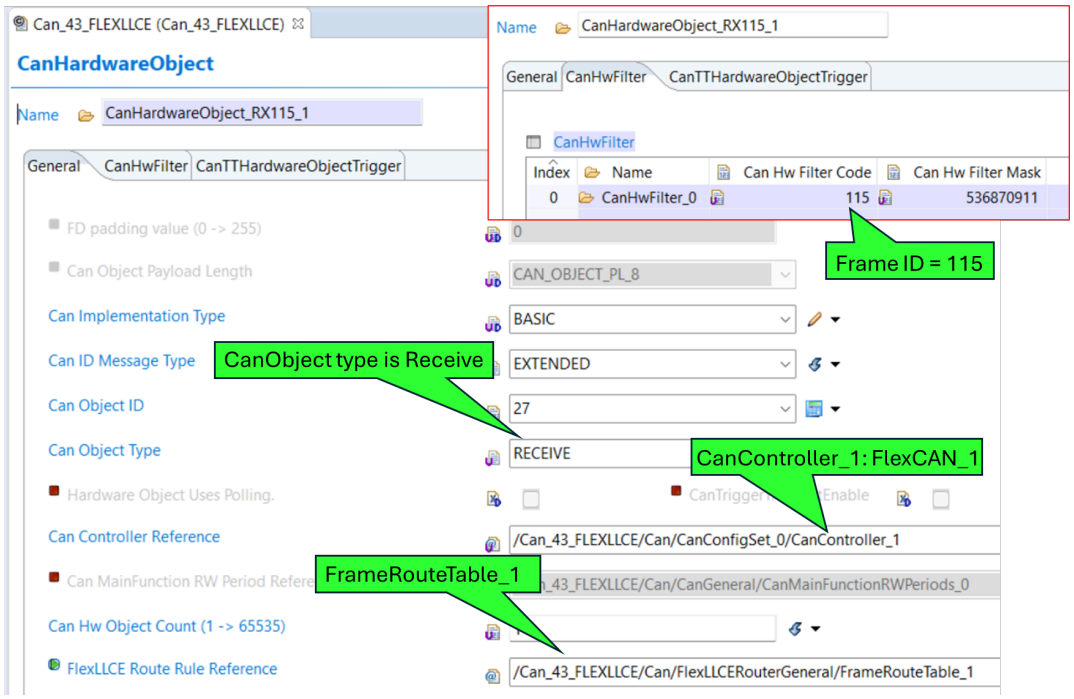

Figure 29. CanHardwareObject_RX115_1

Since the FlexLLCE Route Rule Reference is configured to FrameRouteTable_1, this Hardware Object is not handled as normal reception but handled as internal routing. In case of the internal routing, the host does not need the intervention, hence the notification to the host for RX is configurable. If you look into the FrameRouteTable_1 config, you will see the Host Receive Enable is checked in this case as below. This means the Host Receive Enable is Yes, hence the host core would be notified at the #115 frame reception on FlexCAN_1.

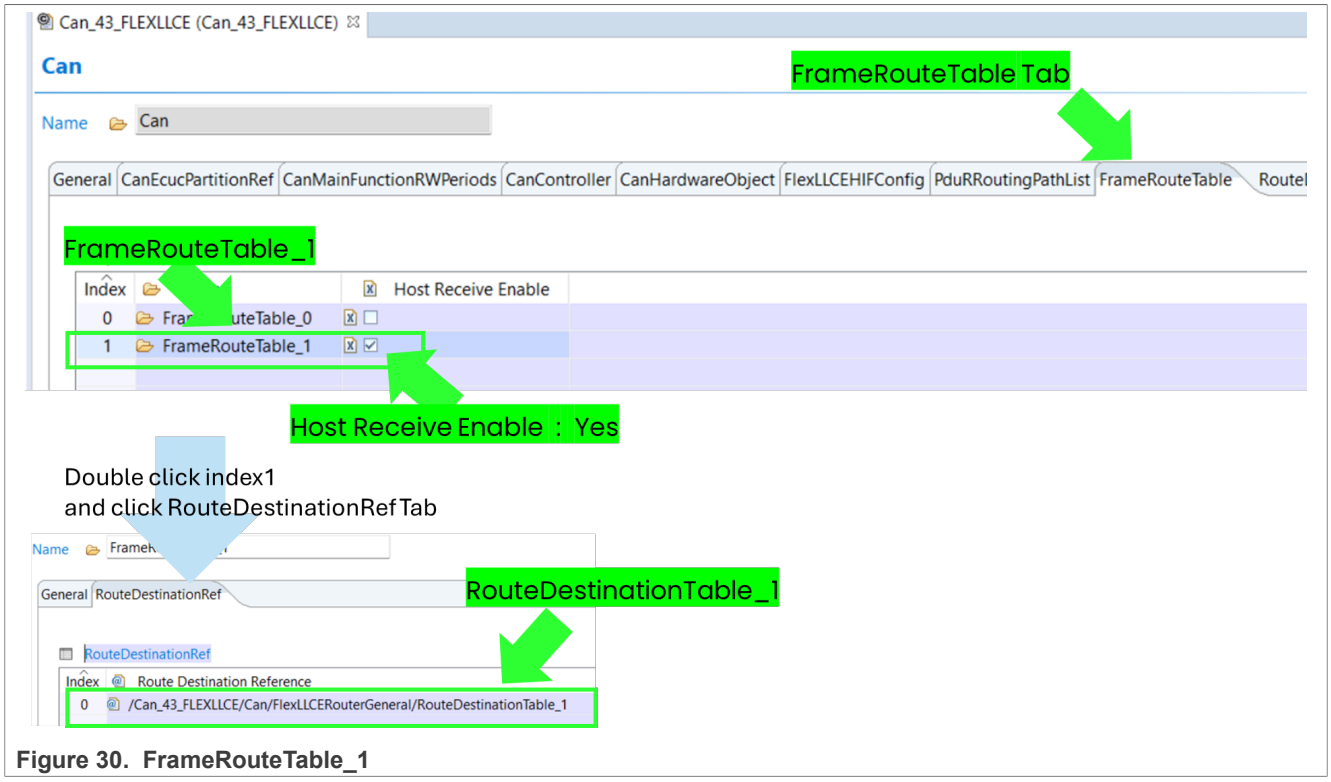

Figure 30. FrameRouteTable_1

In the above figure, if you double click the index1 to open FrameRouteTable_1 config window, and click RouteDestinationRef tab, you will see the FrameRouteTable_1 refers the RouteDestinationTable_1.

Then, if you select the RouteDestinationTable Tab, you will see the RouteDestinationTable_1 is configured to enable CAN2CAN. And the destination CAN channel is configured to FlexCAN_2 as below figure.

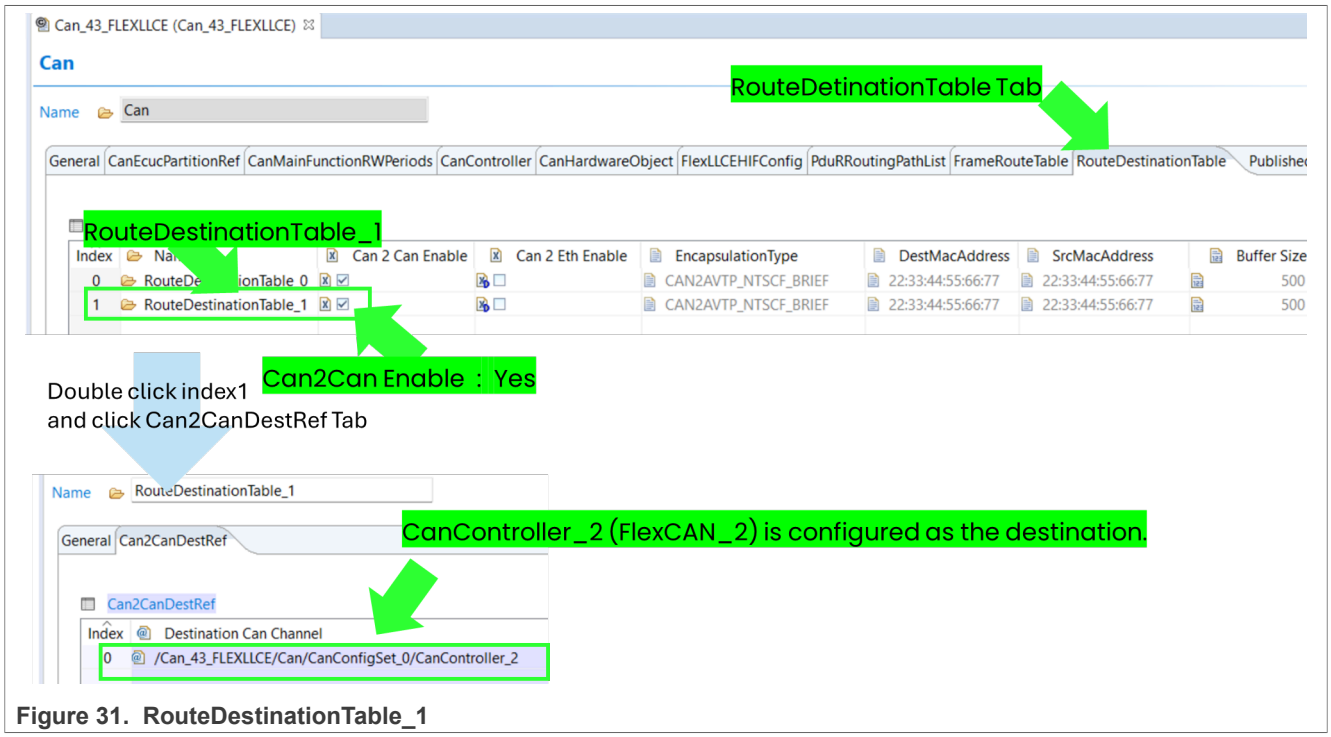

Figure 31. RouteDestinationTable_1

Following diagrams are showing each steps of this CAN2CAN routing.

Below is showing the FlexCAN_1 is receiving the CANFD Frame with ID 115(Extended). After the Tx, the Host Core R52 receives the TxAck notification. Then CanIf_TxConfirmationCnt is incremented by 1.

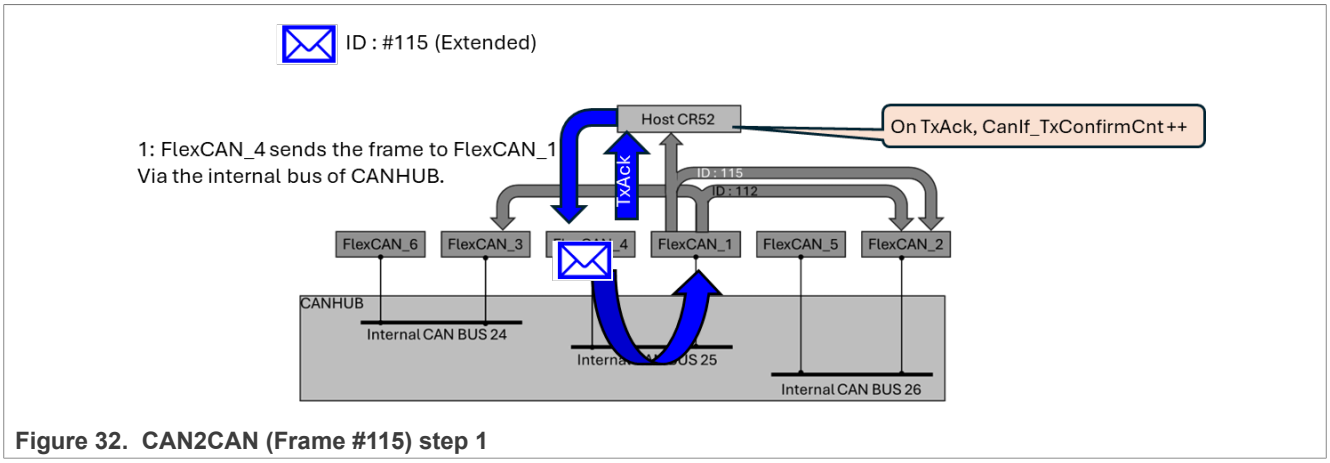

Figure 32. CAN2CAN (Frame #115) step 1

Then, as below, CAN2CAN routing would be executed to FlexCAN_2 as configured. Please note that the FlexLLCE core notifies the RX to the Host core at this moment, since the corresponding FrameRouteTable is configured with HostReceiveEnable=Yes. Then, CanIf_RxIndicationCnt is incremented by 1.

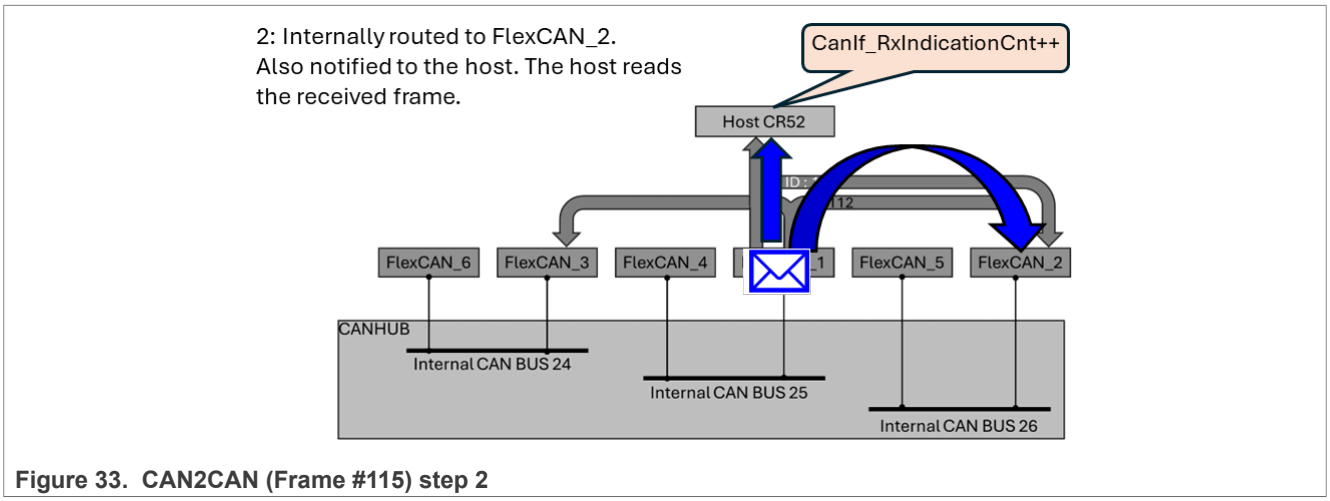

Figure 33. CAN2CAN (Frame #115) step 2

Then, as below, the FlexCAN_2 sends the routed CAN/FD frame to the FlexCAN_5 via the internal CAN BUS 26.

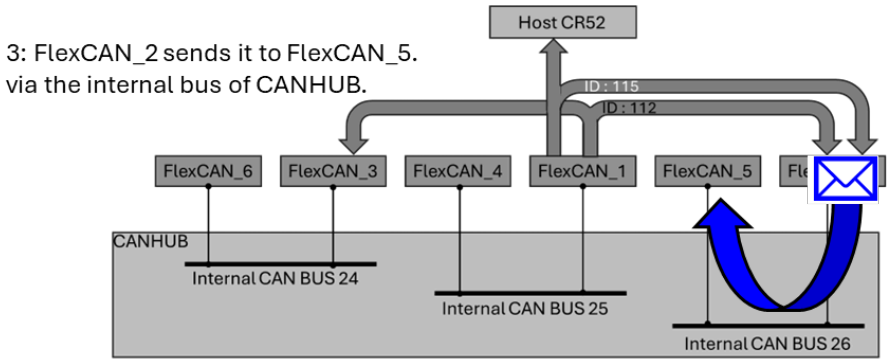

Figure 34. CAN2CAN (Frame #115) step 3

Then, as below, FlexCAN_5 receives the frame and the FlexLLCE core notifies it to the Host core. CanIf_RxIndicationCnt is incremented by 1.

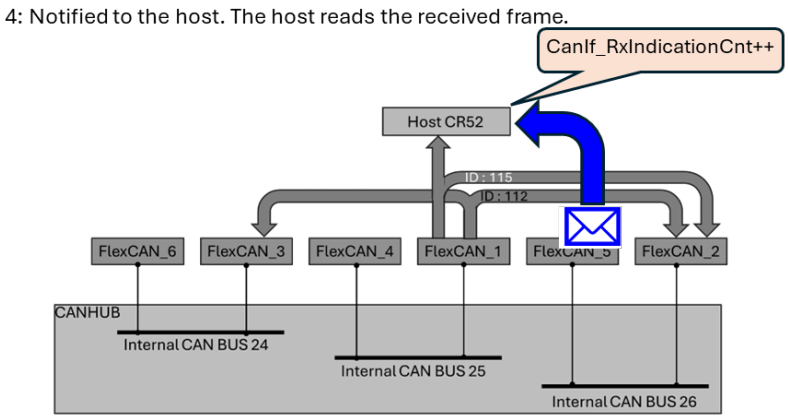

Figure 35. CAN2CAN (Frame #115) step 4

3.4 Run the example with the external loopback config

If you modify the CANHUB config on the EB-Tresos, you can configure the external CAN loopback instead of the internal CAN bus config. Eventually you can observe the CAN frames behavior of the CAN2CAN example using your logic analyzer.

The overview diagram of the external loopback config is as below. The CAN pin 0-5 are allocated to each FlexCANs instead of the internal CAN buses. On the EVB, they can be connected as the external loopback. If you probe each CAN_TX signals on the EVB, you can observe the CAN bus activity of each FlexCANs.

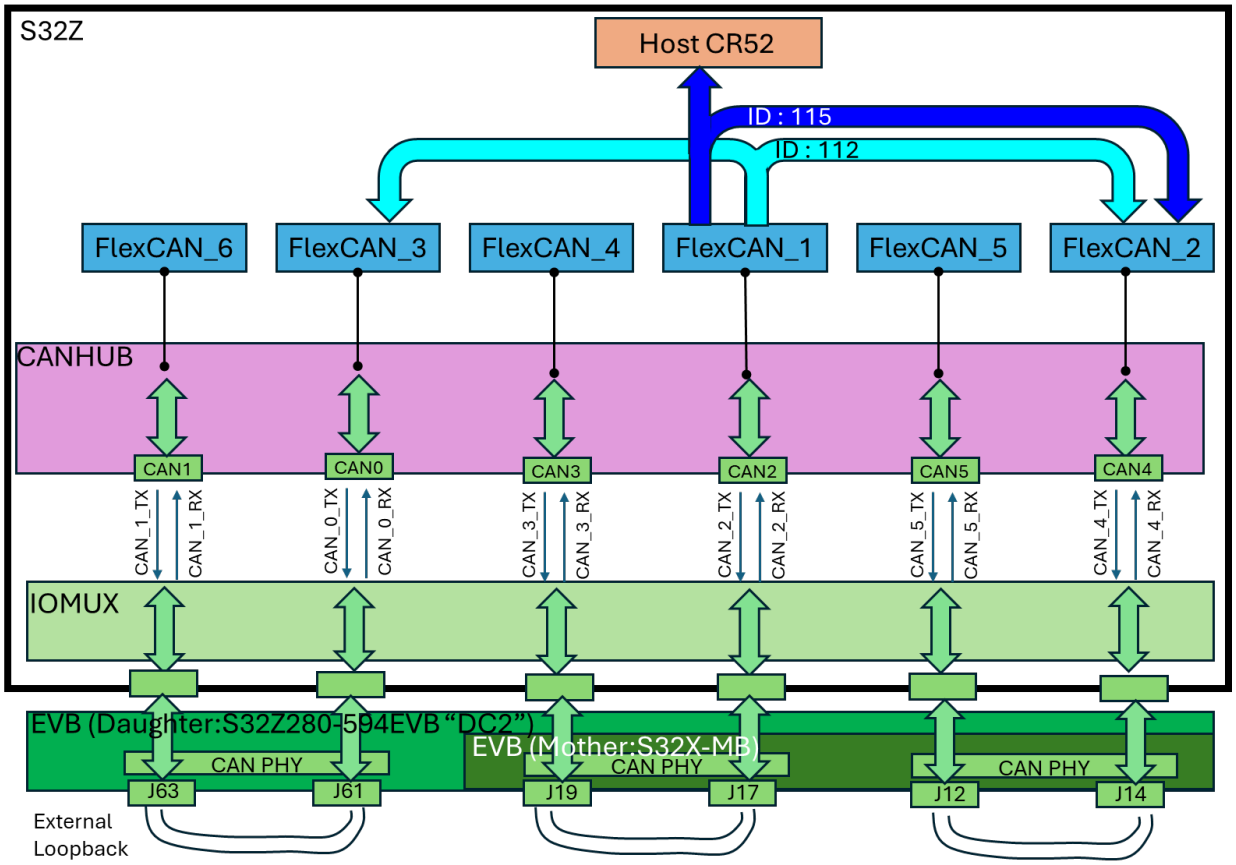

Figure 36. Overview of the external loopback config

The EB-Tresos config is for using CAN0/1/2/3/4/5 buses of CANHUB. So, you should connect the external loopback between CAN0 and CAN1 on the S32Z daughter card and between CAN2 and CAN3, CAN4 and CAN5 respectively on the S32X-MB mother board.

On the daughter card (S32Z27X-DC), please refer the board picture and corresponding schematic as below. CAN0's CAN-H and CAN-L are J61 pins. J61.pin1 is CAN-H, pin3 is CAN-L. CAN1 is J63. Please connect those pins each other as an external loopback. And probe CAN_0_TX at J62.pin 1-2. Also probe CAN_1_TX at J62.pin5-6.

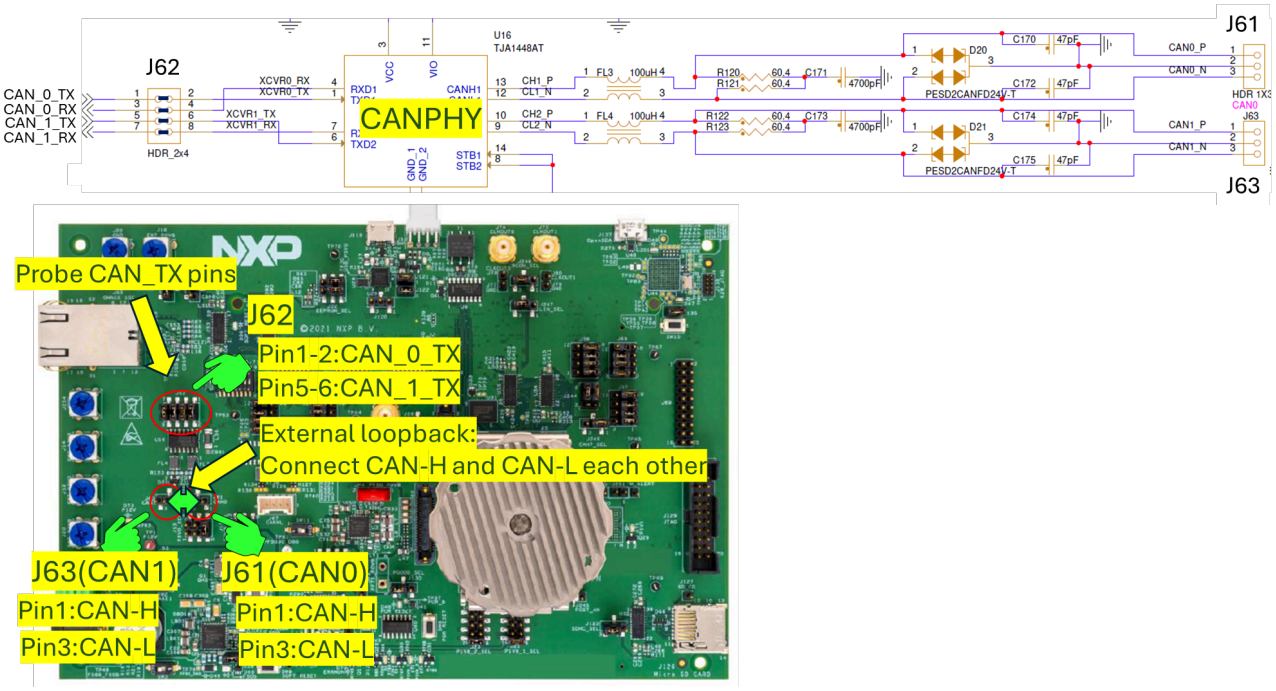

Figure 37. Loopback and probe CAN-TX pins on S32Z27X-DC

On the mother board (S32X-MB), please refer the board picture and corresponding schematic as below. CAN2's CAN-H and CAN-L are J17 pins. CAN3 is J19. CAN4 is J14. CAN5 is J12. Please connect those pins each other as an external loopback. And probe CAN_2_TX at J18.pin 1-2, CAN_3_TX at J18.pin5-6, CAN_4_TX at J13.pin5-6 and CAN_5_TX at J13.pin1-2.

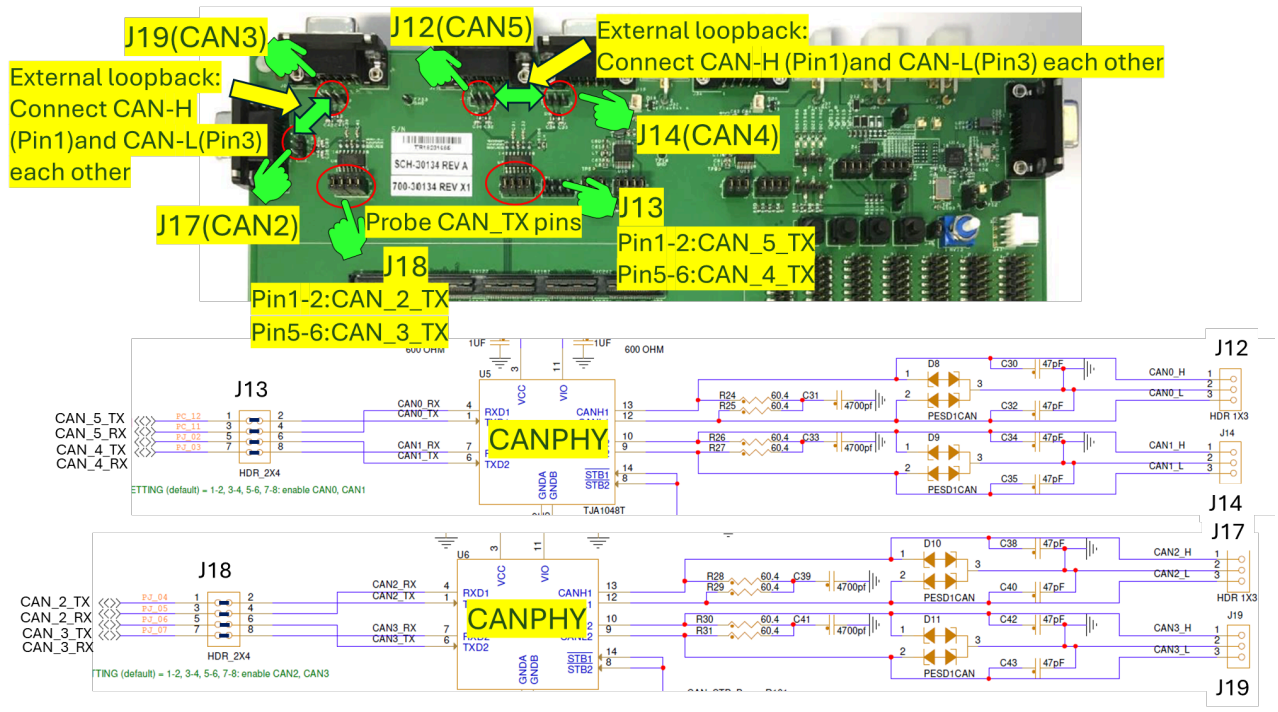

Figure 38. Loopback and probe CAN-TX pins on S32X-MB

3.4.1 How to modify the EB-Tresos config from the internal bus setting to the external loopback setting

Open the Rm module of the imported original config on the EB-Tresos Studio. Please select the Can Hub Configuration tab and you will see the RmCanHubConfig table. Then, as below, for CAN_GROUP_1-6, please configure CAN_BUS_0 – 5 instead of CANBUS_24-26.

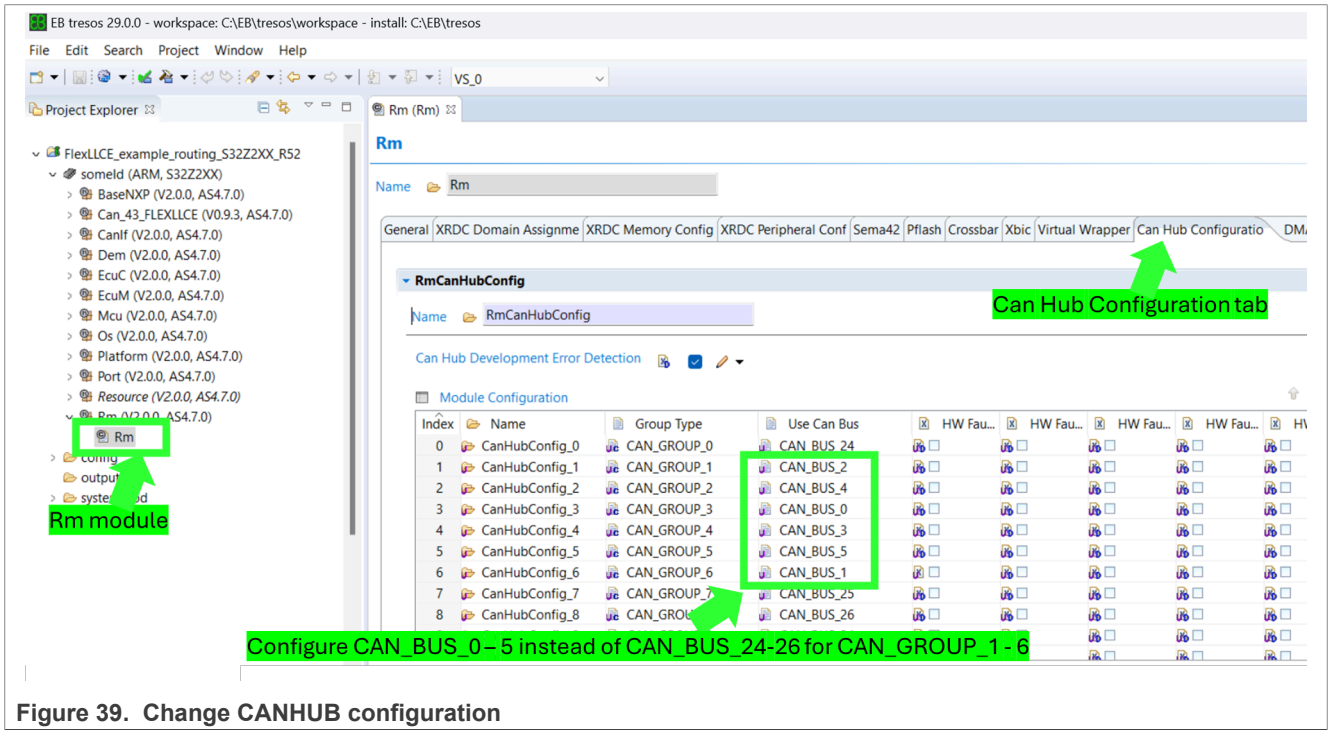

Figure 39. Change CANHUB configuration

Next is the configuration of the IOMUX. Please open the Port module and select the PortContainer tab. Then, double click the index 0 as below.

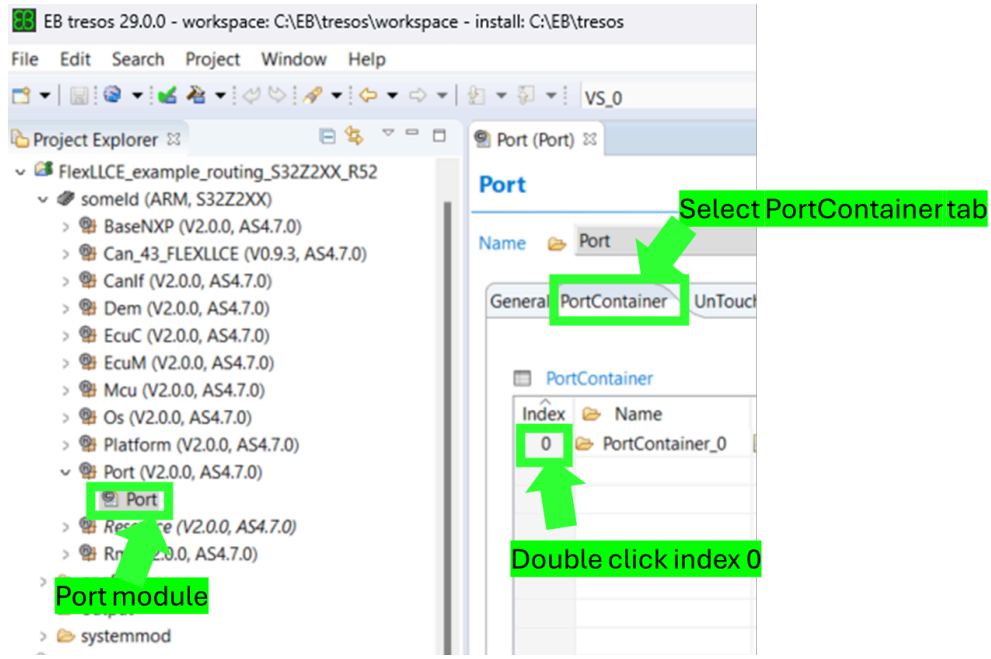

Figure 40. Port module

The Tx pin and Rx pin for CAN0 and CAN1 are already configured in the default config. Hence, click “+” 8 times to add Tx/Rx pins for another 4 CAN channels as below.

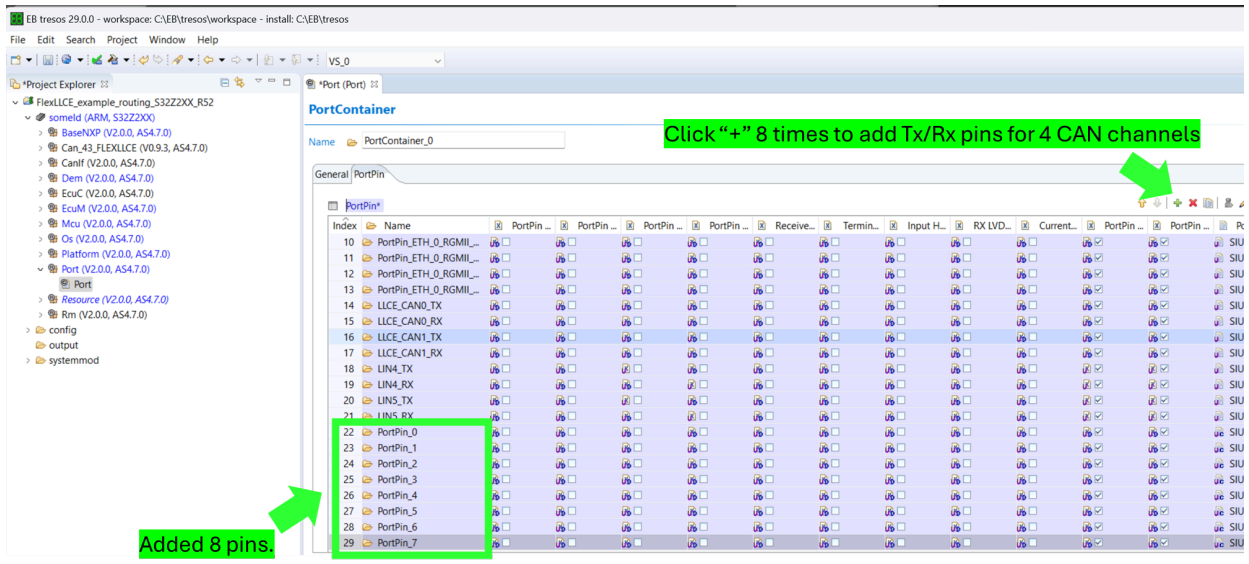

Figure 41. Add 8 pins config

Then, change names of the added 8 pins as below.

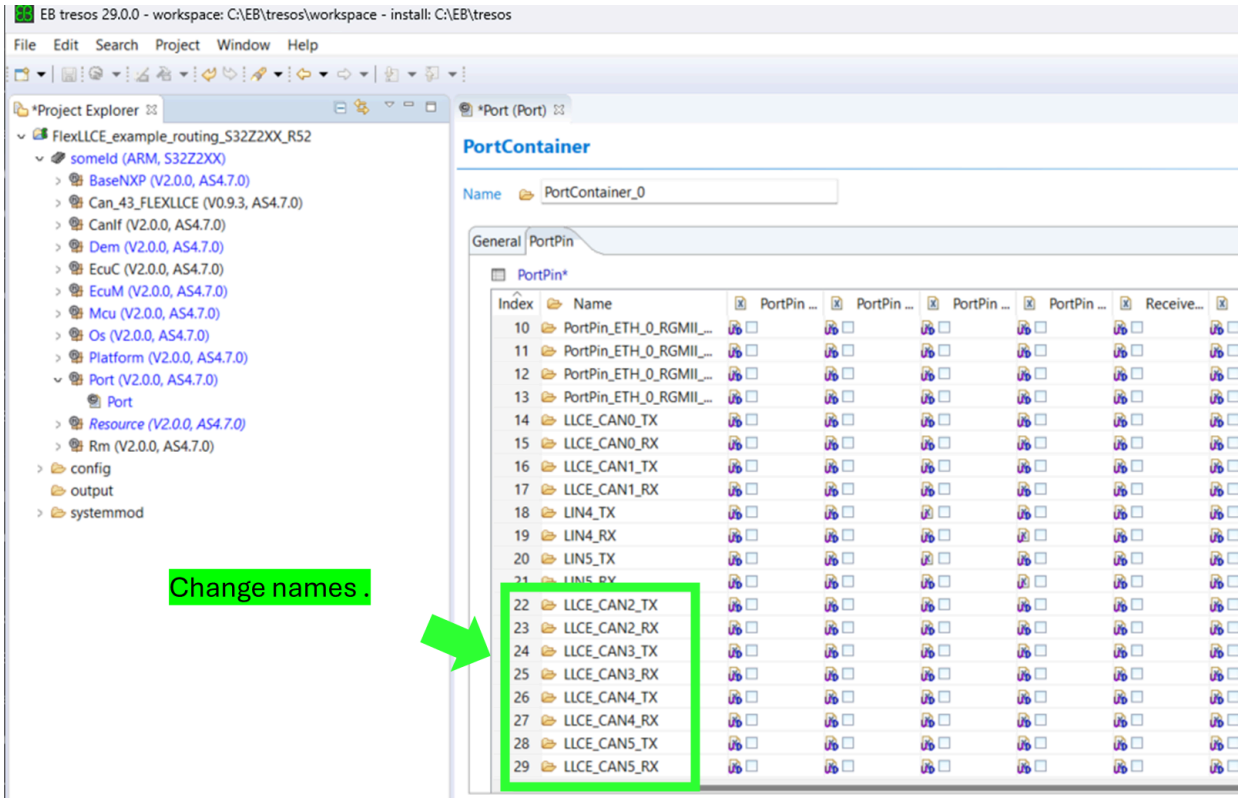

Figure 42. Change names of the added 8 pins

Double click the index 22 to open LLCE_CAN2_TX config window. Then, change the config of PortPin SIUL2 Ingerface, PortPin Mscr and PortPin Mode as below.

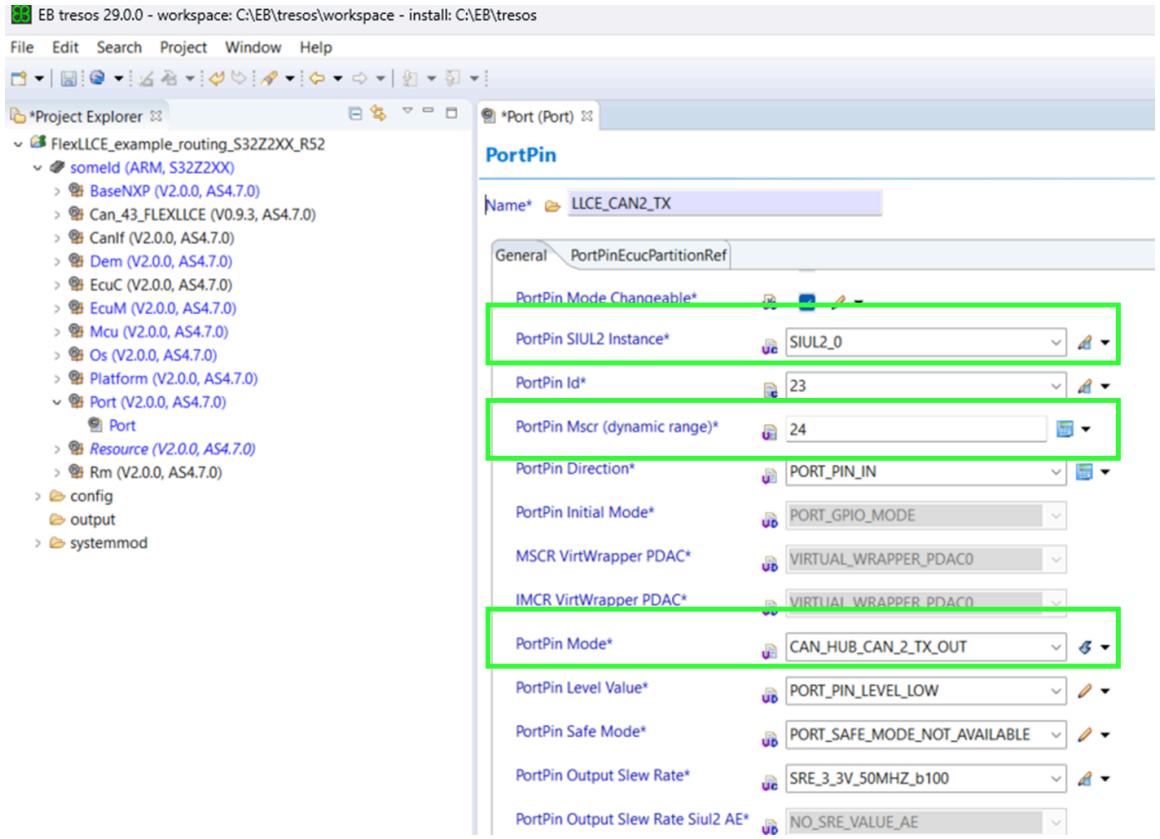

Figure 43. LLCE_CAN2_TX pin IOMUX config

Also for other CAN pins, configure these 3 parameters according to below table.

Name	PortPin SIUL2 Instance	Port Pin Mscr	PortPin Mode
LLCE_CAN2_TX	SIUL2_0	24	CAN_HUB_CAN_2_TX_OUT
LLCE_CAN2_RX	SIUL2_0	25	CAN_HUB_CAN_2_RX_IN
LLCE_CAN3_TX	SIUL2_1	34	CAN_HUB_CAN_3_TX_OUT
LLCE_CAN3_RX	SIUL2_1	35	CAN_HUB_CAN_3_RX_IN
LLCE_CAN4_TX	SIUL2_4	114	CAN_HUB_CAN_4_TX_OUT
LLCE_CAN4_RX	SIUL2_4	115	CAN_HUB_CAN_4_RX_IN
LLCE_CAN5_TX	SIUL2_1	76	CAN_HUB_CAN_5_TX_OUT
LLCE_CAN5_RX	SIUL2_1	77	CAN_HUB_CAN_5_RX_IN

Figure 44. CAN_TX/RX pins config parameters

For LLCE_CAN4_TX and LLCE_CAN4_RX, please change PortPin Output Slew Rate config as below.

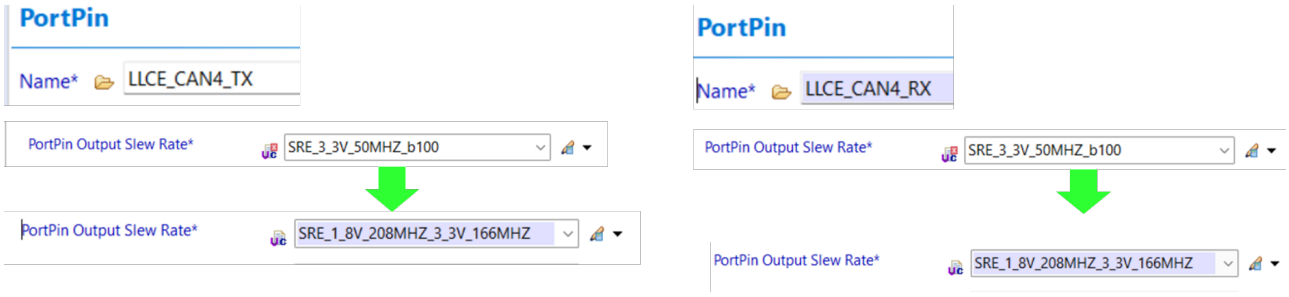

Figure 45. PortPin Output Slew Rate config for CAN4

Select General tab and put 30 as PortNumberOfPortPins.

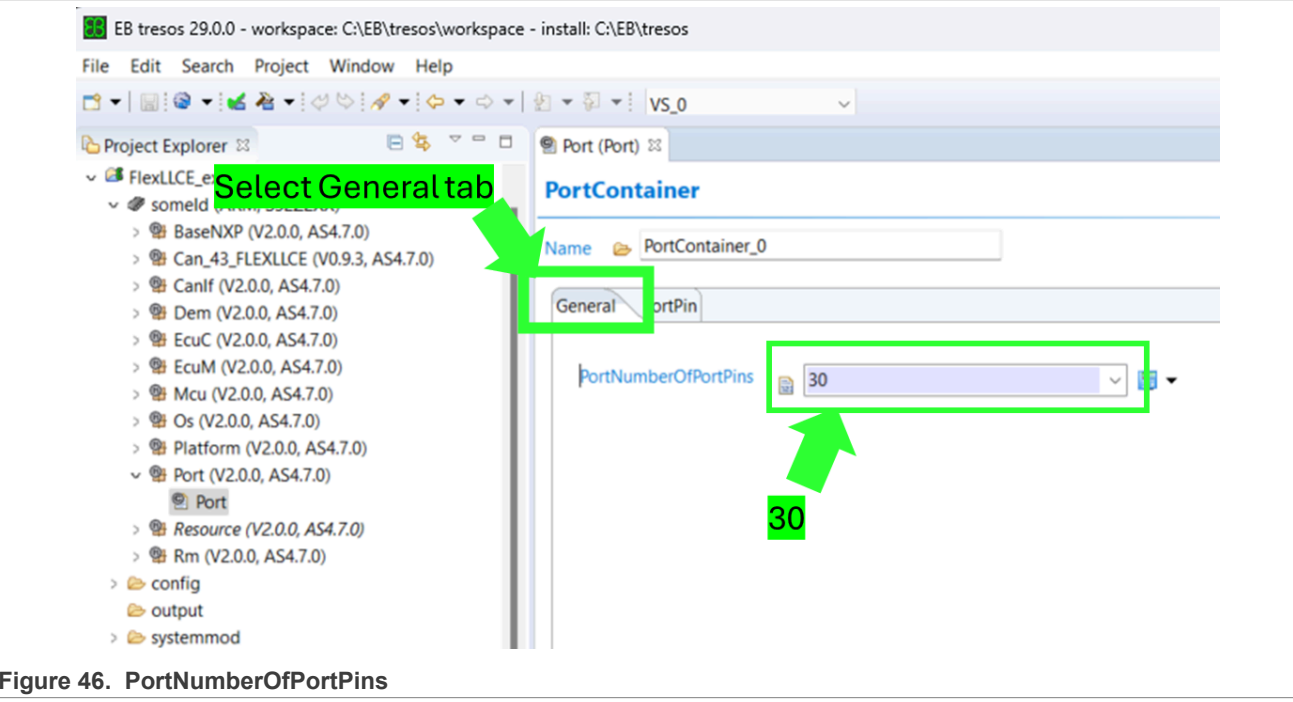

Figure 46. PortNumberOfPortPins

Then, generate the config and ensure no errors.

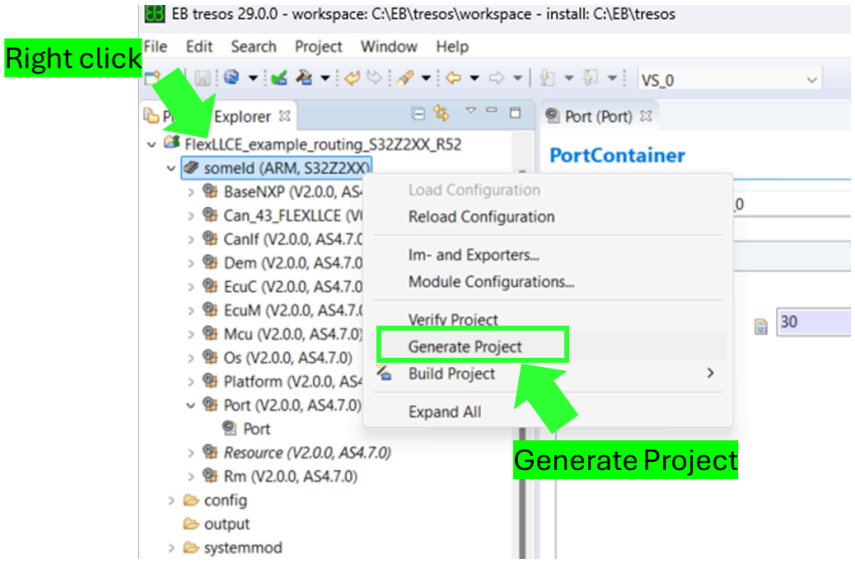

Figure 47. Generate config

3.4.2 Modify main.c

In the default main.c, there is a snippet which executes the transmission test from all FlexCANs. This is designed for the internal bus config and will stuck under the external loopback config hence you should comment out that part. The location of the main.c and the snippet to be commented out is as below.

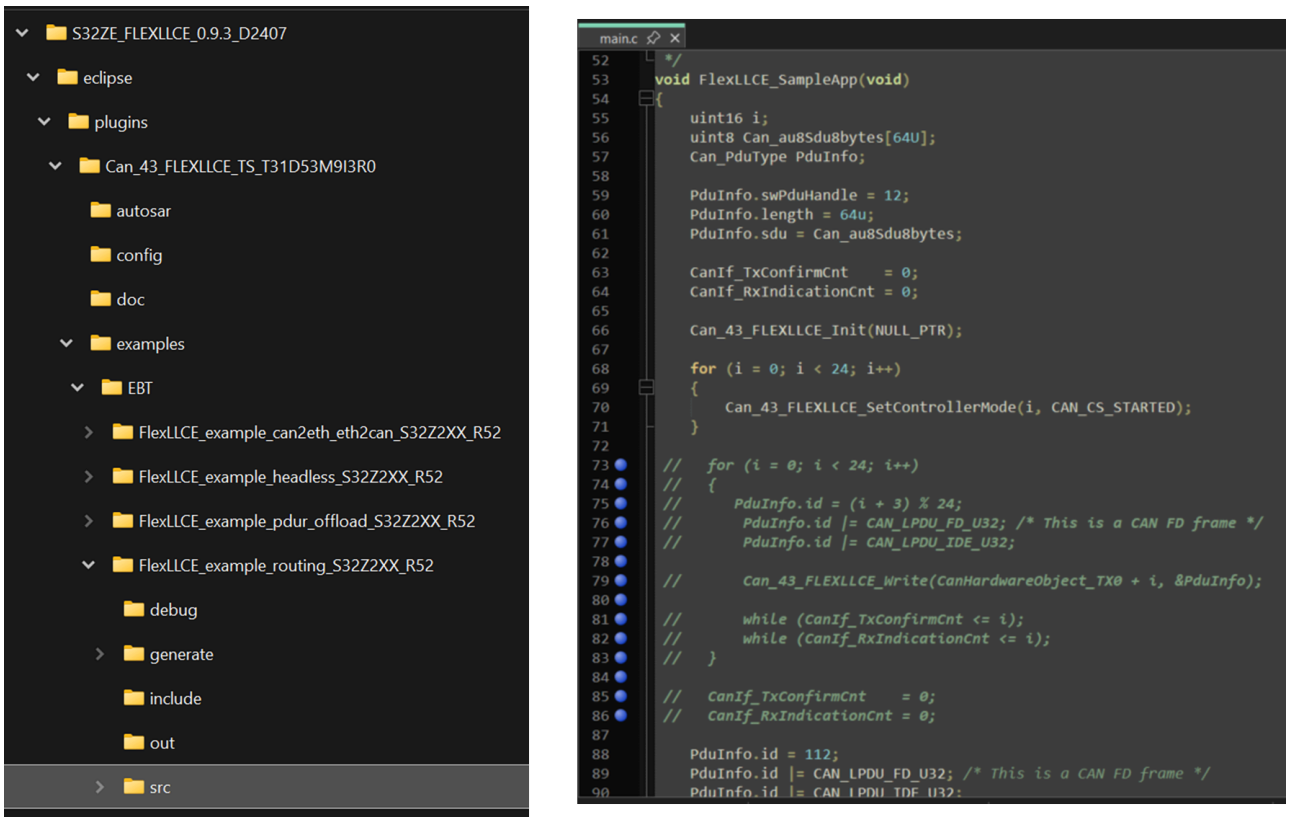

Figure 48. Comment out the unnecessary part in the main.c

3.4.3 Build, run and Capture CAN2CAN activity

As previously explained, move to the CAN2CAN example project folder and run the command “make build”.

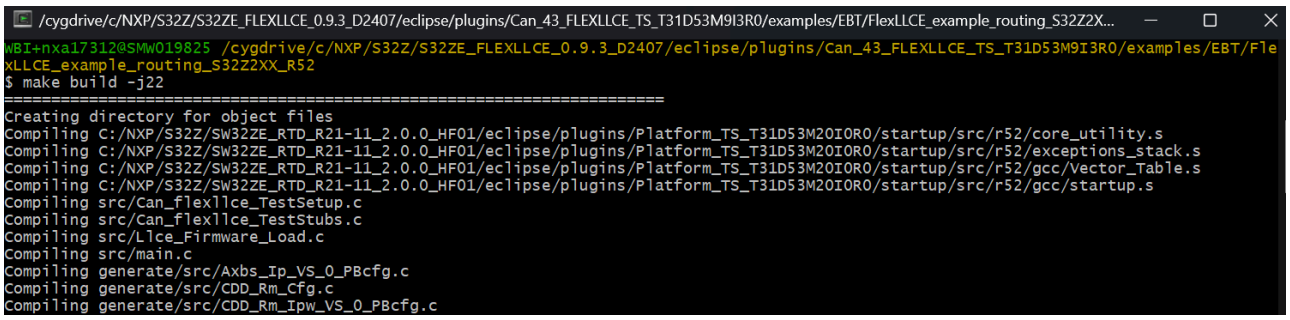

Figure 49. Build

Then, connect your Lauterbach debugger to the S32Z EVB. Start your TRACE32 and change the directly to “debug” folder under the CAN2CAN example project folder. Then run script “run.cmm”, which is loading the build elf file and run.

Figure 50. run.cmm

- **Frame ID #112 : 2 Destination channels CAN2CAN without Host Reception**

When the brakepoint is at line 97 (as shown in the following figure), it’s just after sending one extended CANFD frame with ID=112. At that time, the variable CanIf_TxConfirmCnt and CanIf_RxIndicationCnt are 1 and 2 respectively as same as internal CAN bus config.

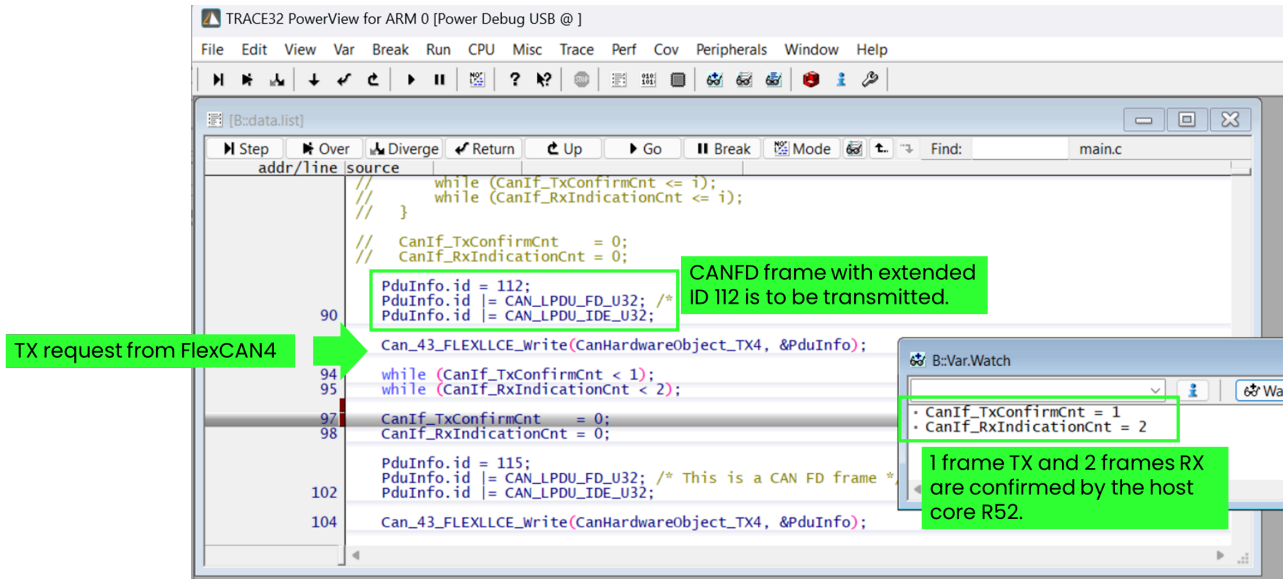

Figure 51. Brakepoint after CAN2CAN (Frame ID=112)

If you captured the waveform with the logic analyzer, you will see as shown in the following figure.

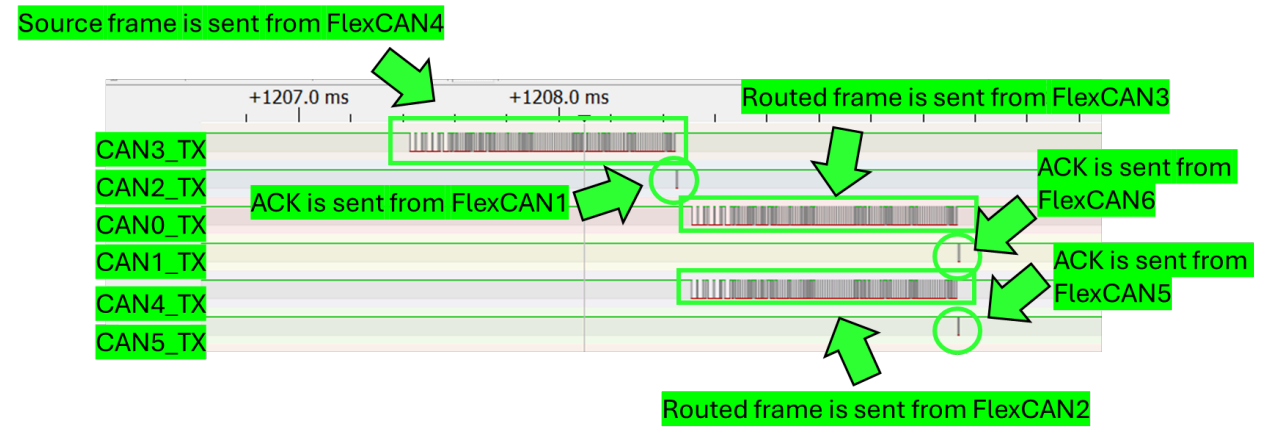

Figure 52. Waveform of CAN2CAN (Frame ID=112)

As above, you will see the FlexCAN1 receives the source frame and the 2 routed frames are sent from FlexCAN3 (via CAN0_TX) and from FlexCAN2 (via CAN4_TX).

- **Frame ID #115 : One Destination channel CAN2CAN with Host Reception**

When the brakepoint is at line 110 (as shown in the following figure), it's just after sending one extended CANFD frame with ID=115. At that time, the variable CanIf_TxConfirmCnt and CanIf_RxIndicationCnt are 1 and 2 respectively as same as internal CAN bus config.

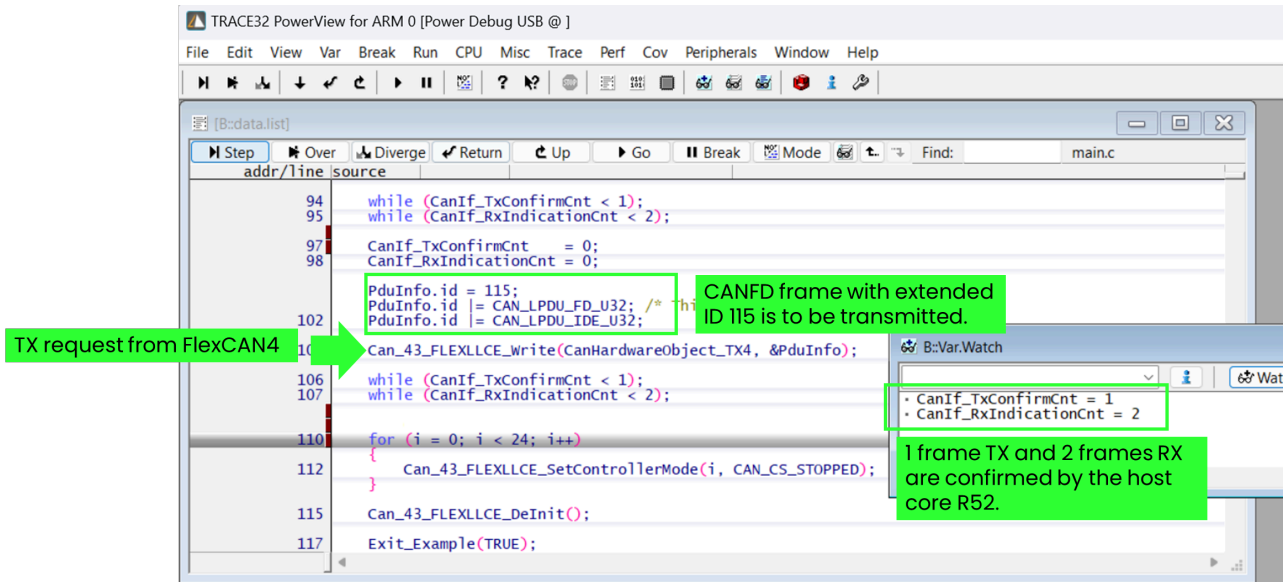

Figure 53. Brakepoint after CAN2CAN (Frame ID=115)

If you captured the waveform with the logic analyzer, you will see what in shown in the following figure.

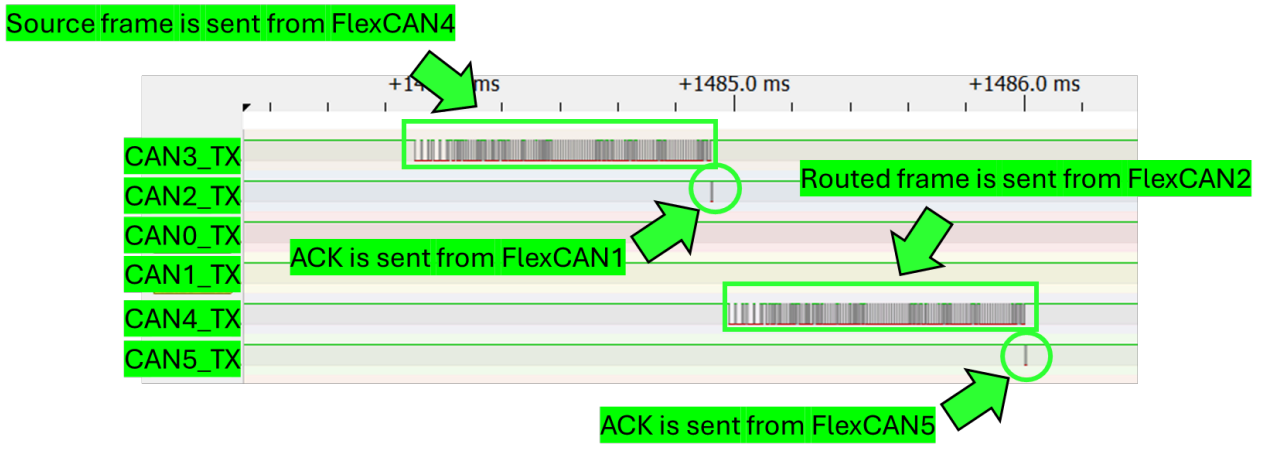

Figure 54. Waveform of CAN2CAN (Frame ID=115)

As above, you will see the FlexCAN1 receives the source frame and the routed frames are sent from FlexCAN2 (via CAN4_TX).

4 CAN2ETH/ETH2CAN: Build and Run the project with default config

4.1 Generate the Tresos config

The default Tresos config is located under the example project as shown in the below figure.

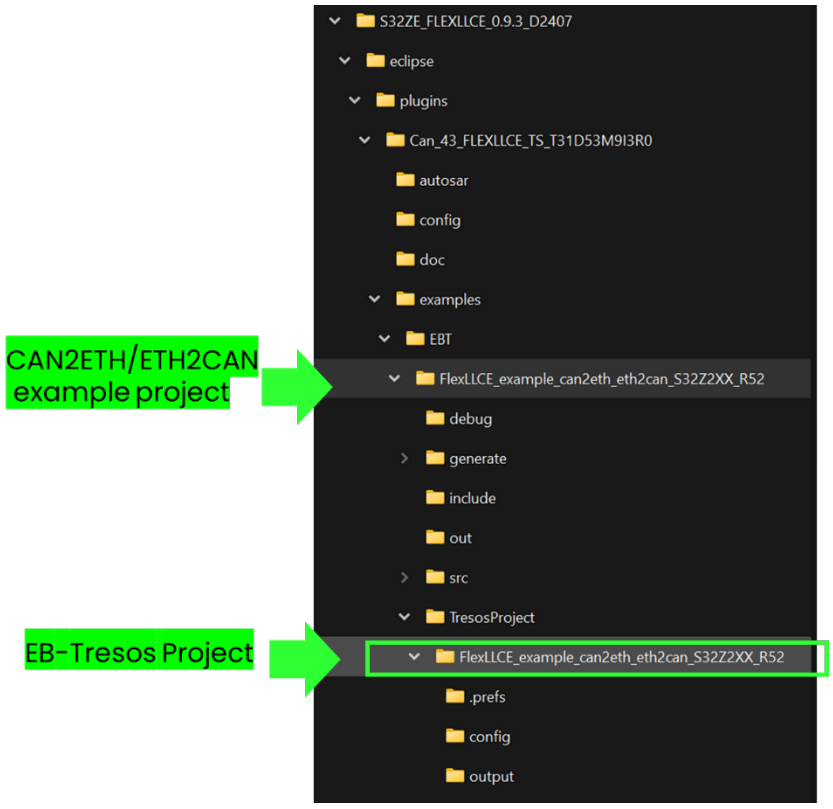

Figure 55. Tresos config location

Start your EB-Tresos and import the EB-Tresos Project as below.

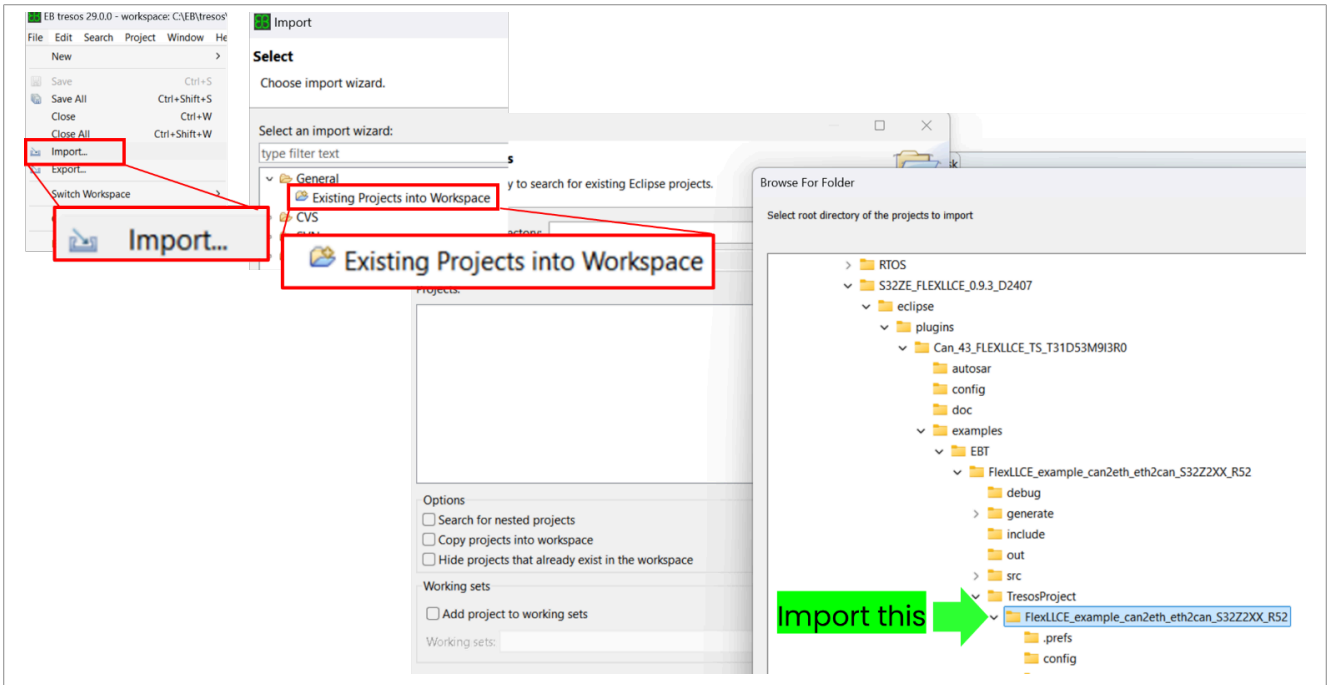

Figure 56. Import EB-Tresos project

Then, generate it as below figure.

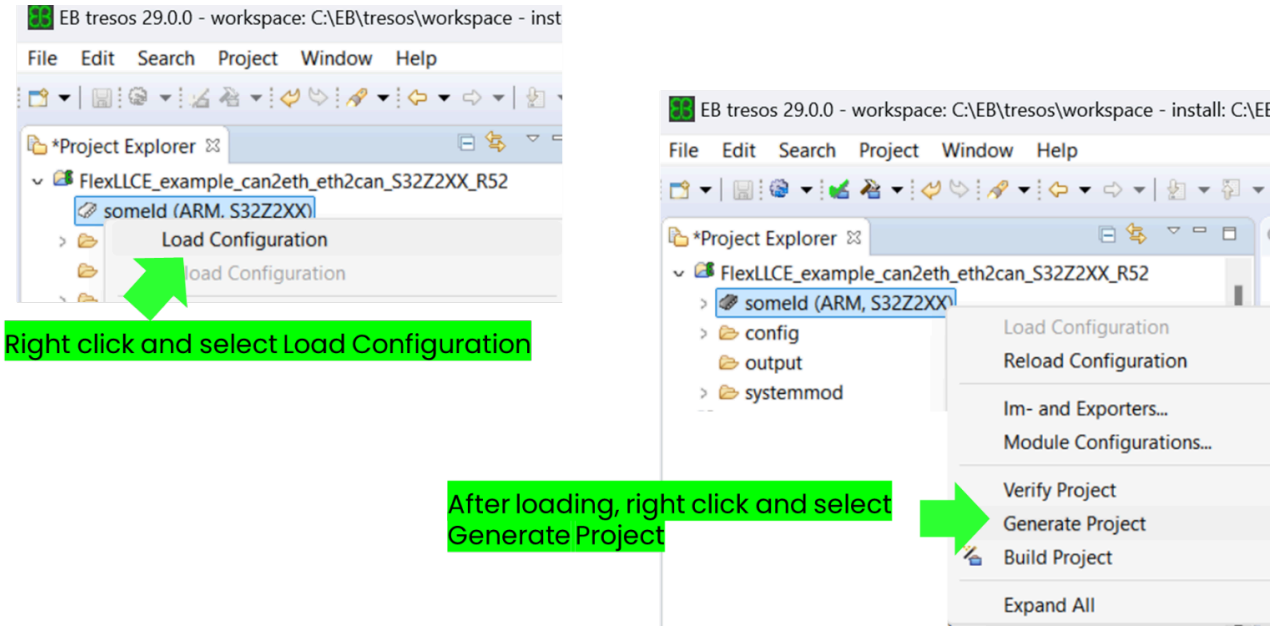

Figure 57. Generate the Tresos config

4.2 Build the example project with default config

On the Linux emulated environment such as Cygwin (https://cygwin.com/), change directory to the CAN2ETH/ETH2CAN example project folder and run the command “make build”. Optionally you can use -j command for multi thread processing.

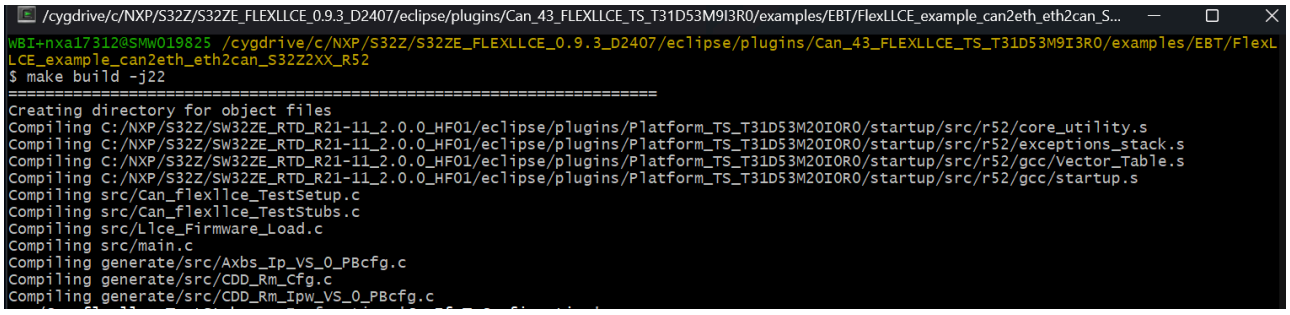

Figure 58. make

Then, you will get the elf file under the folder “out”

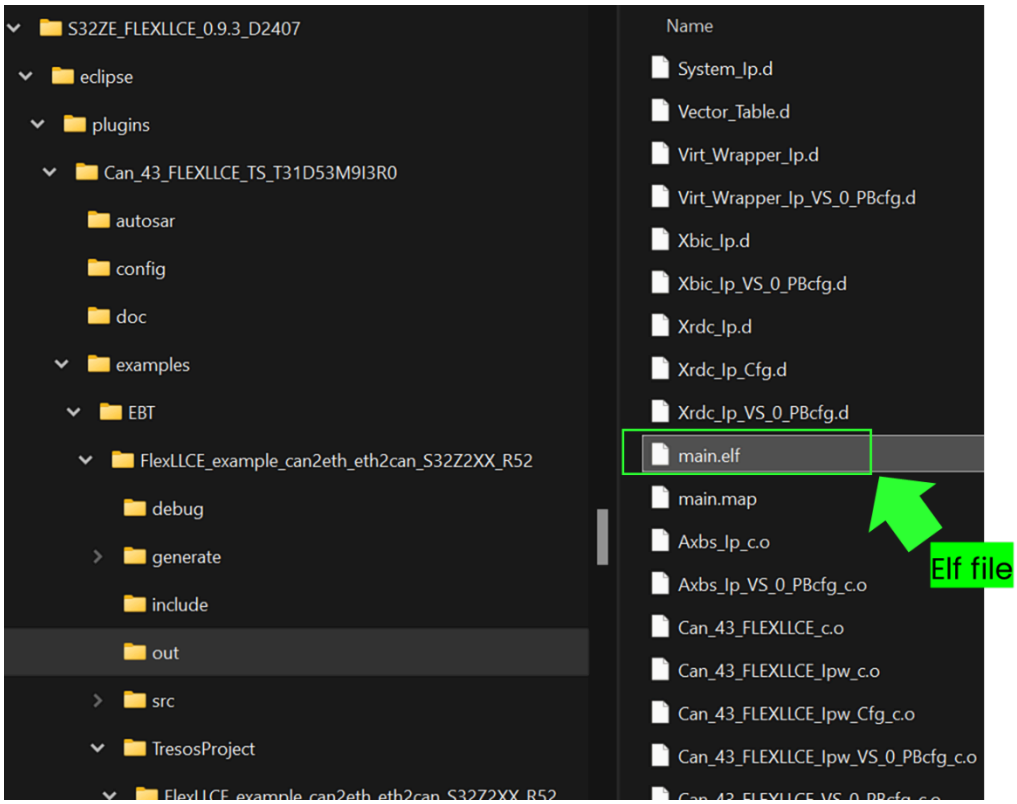

Figure 59. Built Elf file

4.3 Run the example with default config

Then, connect your Lauterbach debugger to the S32Z EVB. Start your TRACE32 and change the directory to “debug” folder under the CAN2ETH/ETH2CAN example project folder. Then run script “run.cmm”, which is loading the build elf file and run.

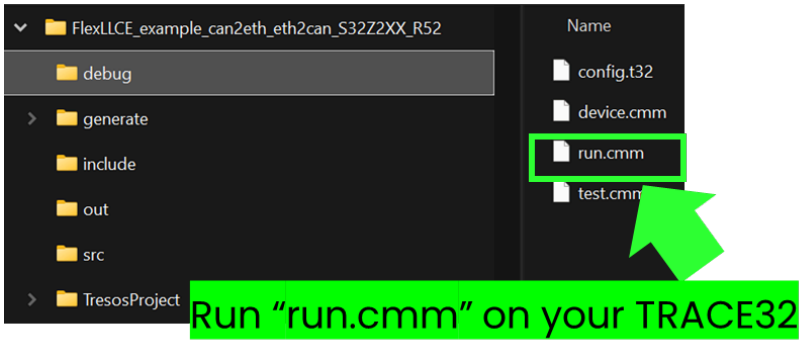

Figure 60. run.cmm

On TRACE32, put the following variables to the var.watch window.

CanIf_TxConfirmCnt : The number of Tx successful CAN frame

CanIf_RxIndicationCnt : The number of Rx successful CAN frame

Insert a brakepoint at line 85, which is just after testing Tx/Rx of all FlexCAN instances as shown in the following figure.

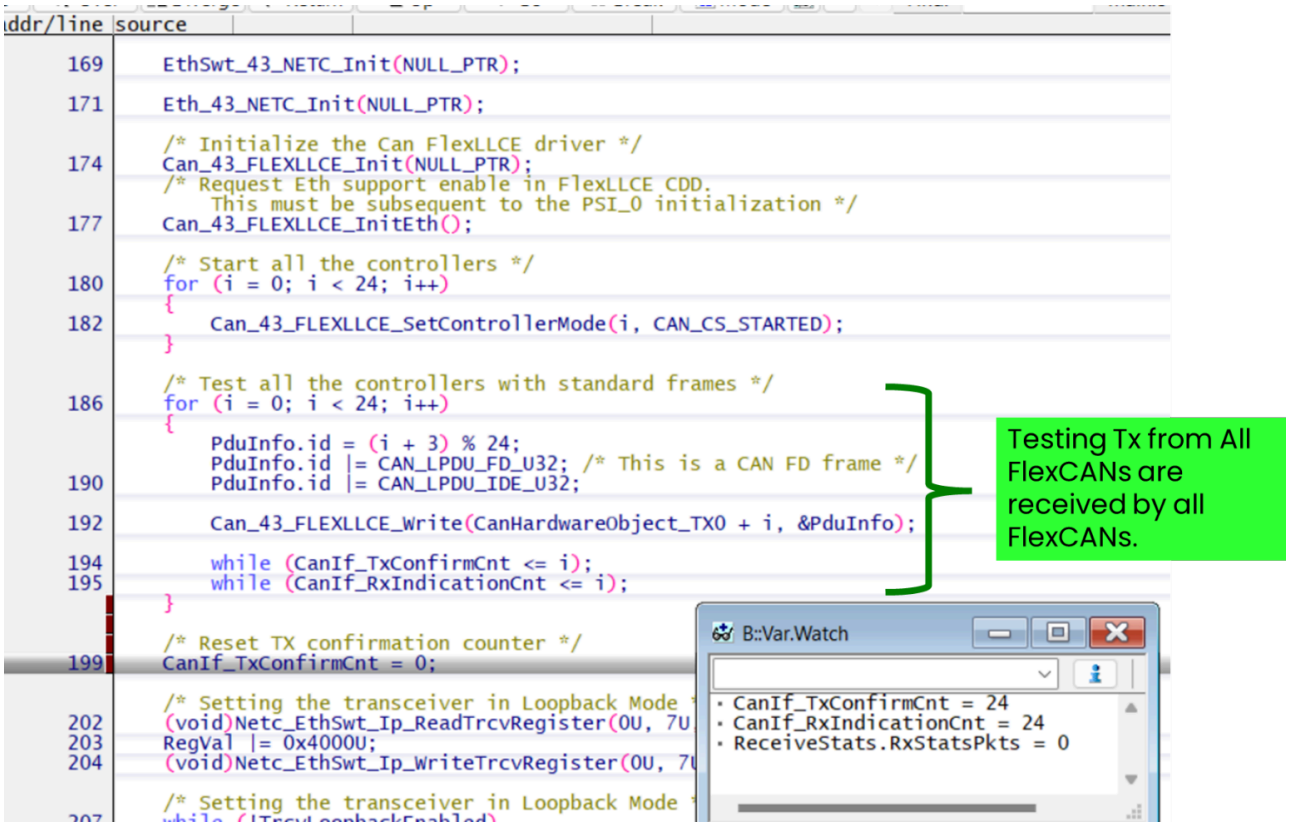

Figure 61. Brakepoint after testing Tx/Rx of all FlexCAN instances

The variables CanIf_TxConfirmCnt and CanIf_RxIndicationCnt are both 24. This is showing that all the 24 FlexCAN instances have successfully sent and received CAN frames via internal CAN bus of CANHUB.

If you look into the config on EB-Tresos studio GUI, you will find the CAN HUB configuration (configured on the Rm module) is connecting the FlexCAN peripheral outputs to three CAN buses. FlexCAN0,3,6…21 are connected to the internal CAN bus 24. FlexCAN1,4,7…22 are connected to the internal CAN bus 25. FlexCAN2,5,8…23 are connected to the internal CAN bus 26. And the CanHardwareObject config in Can_43_FLEXLLCE module has the CanHarwareObject_RXn config which configures Frame IDn to be received by FlexCAN instance n.

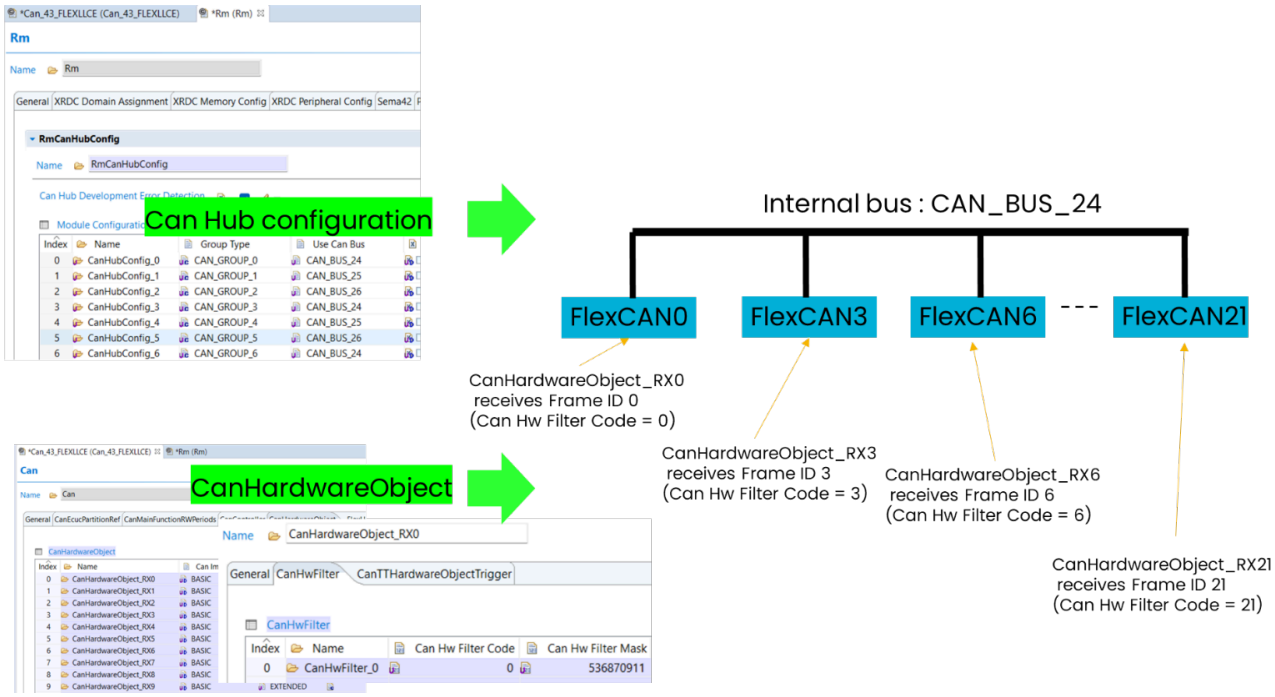

Figure 62. Default Tresos config

With above config, the The variable CanIf_TxConfirmCnt and CanIf_RxIndicationCnt would be both 24 when the brakepoint is at line 199.

Then, insert another brakepoint at line 244, which is just after executing CAN2ETH.

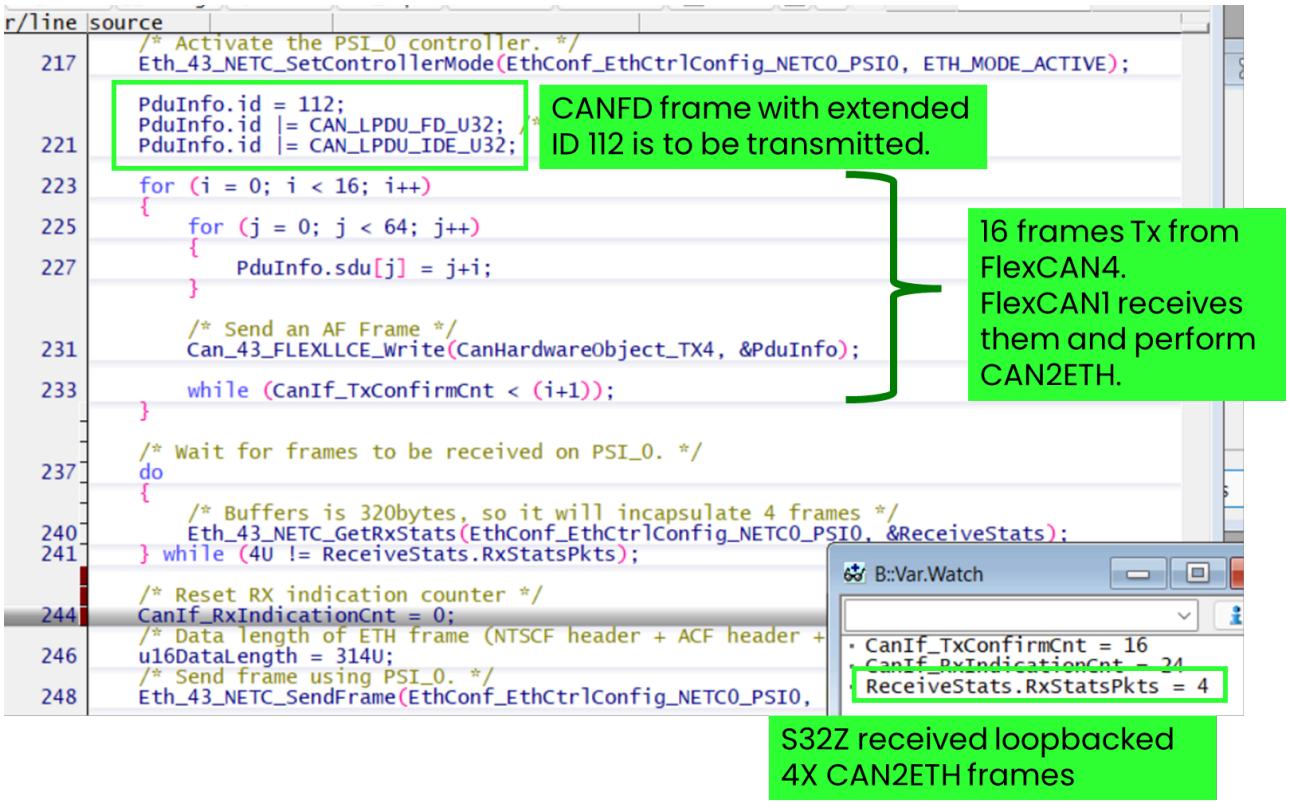

Figure 63. Brakepoint after CAN2ETH

In the sample project, CANFD frame with extended ID 112 is sent from FlexCAN4 four times. These frames would be received by FlexCAN1, encapsulated in an Ethernet frame and transmitted. The var,watch window is showing that the 4X CAN2ETH frame were sent.

If you look into the Tresos config, you will find that the CanHardwareObject_RX112_1 as shown in following figure.

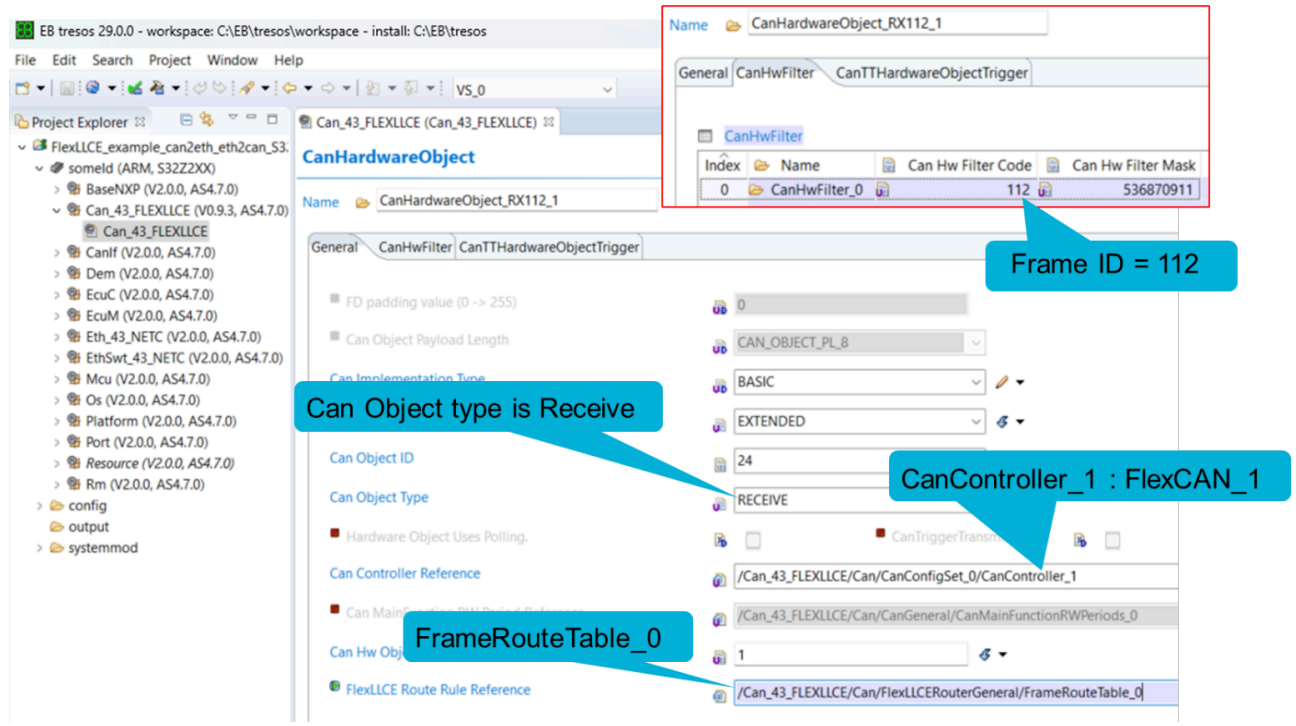

Figure 64. CanHardwareObject_RX112_1

The FrameRouteTable_0 represents the routing offload preference for this CAN Hardware Object. You will find the config of FrameRouteTable_0 as follows. This configuration enables CAN2ETH. The CAN2ETH would be triggered when ethernet frame buffer gets filled. When a buffer cannot accept another 64-byte CAN frame (maximum size), that buffer is considered full. The buffer size is 320bytes and eventually 4 X 64bytes CANFD frame would be encapsulated into one Ethernet frame.

The FlexLLCE FW supports IEEE1722 AVTP Control Format (aka ACF). ACF has two types of headers, time-synchronous (aka TSCF : Time Synchronous Control Format) and non–time-synchronous (aka NTSCF : Non Time Synchronous Control Format). The TSCF header has the avtp_timestamp field which enables synchronous delivery of control messages and is useful in applications where synchronous behavior is required. The NTSCF header has no timestamp information and it's useful for delivering messages as efficiently and quickly as possible and where synchronization of control is not required. The ACF payload follows the header. The ACF payload is a concatenation of one or more arbitrary ACF messages. The IEEE1722 standard defines 2 ACF message types for CAN/CANFD. One is the ACF_CAN, and the other one is the ACF_CAN_BRIEF. The ACF_CAN has the message_timestamp field which is the time of receipt for the CAN/CANFD message.

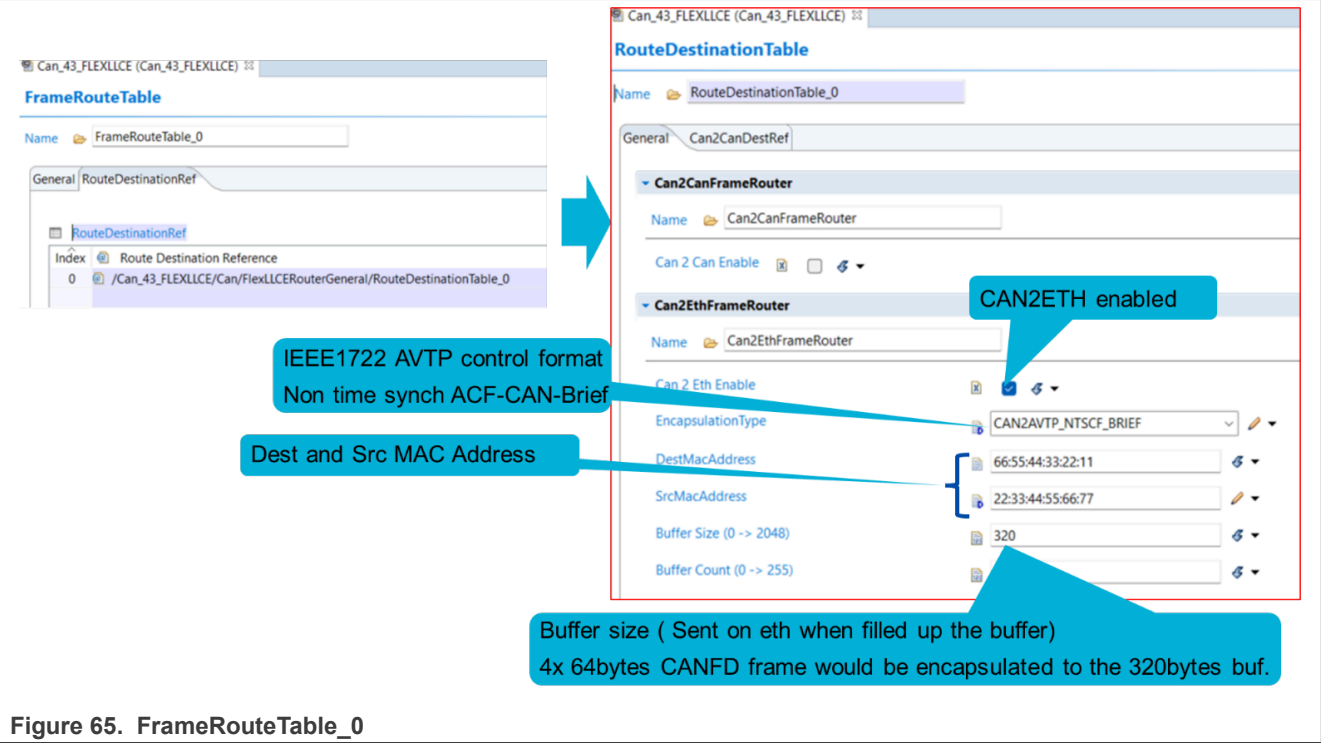

Figure 65. FrameRouteTable_0

Then, insert a brakepoint at line 253, which is just after executing ETH2CAN. As below figure, this default example project is sending the Eth2Can_Frame by the API Eth_43_NETC_SendFrame(). Since the main.c configures the Ethernet PHY to the loopback mode, the sent Eth2Can_Frame would be sent back to S32Z. Eventually it would be received by FlexLLCE via NETC and would decapsulate and send 4 ACF-CAN.

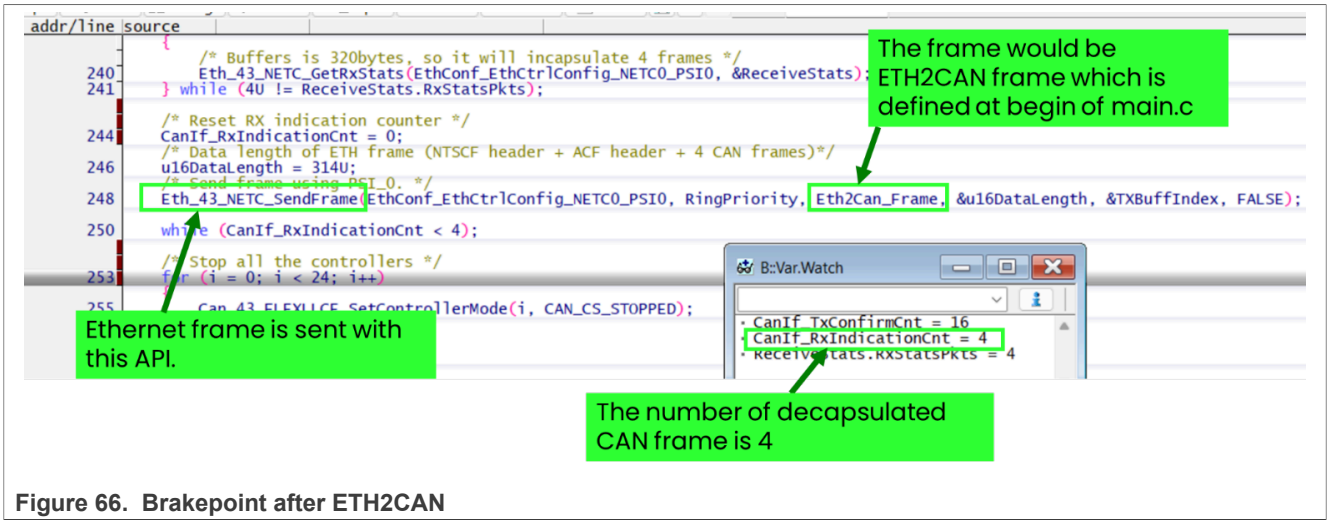

Figure 66. Brakepoint after ETH2CAN

The Eth2Can_Frame is defined in the beginning of main.c as below figure.

```
static uint8 Eth2Can_Frame[320U] = {
    0x66, 0x55, 0x44, 0x33, 0x22, 0x22, /* Destination MAC address */
    0x22, 0x33, 0x44, 0x55, 0x66, 0x77, /* Source MAC address */
    0x22, 0xF0, /* Ethernet Type (AVTP - 0x22F0) */
    0x82, /* Subtype_field (NTSCF - 0x82) */
    0x01, 0x20, /* (3'b - version) (1'b - sv) (1'b - r) (11'b - ntscf_data_length => 0x0120 - 288 bytes)   */
    0x00, /* sequence_num */
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, /* stream_id */
    0x04, 0x12, /* (7'b - acf_msg_type => 0x2 - ACF_CAN_BRIEF) (9'b - acf_msg_len => 0x12 - 18U quadlets of data) */
    0x0E, /* (2'b - pad) (1'b - mtv) (1'b - rtr) (1'b - eff) (1'b - brs) (1'b - fdf) (1'b - esi) */
    0x01, /* (3'b - rsv) (5'b - can_bus_id) */
    0x00, 0x00, 0x00, 0x6F, /* can_msg_id */
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, /* can_msg_payload[0-15] */
    0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F, /* can_msg_payload[16-31] */
    0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F, /* can_msg_payload[32-47] */
    0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F, /* can_msg_payload[48-63] */
    0x04, 0x12, /* Start second CAN/CAN FD ACF message */
    0x0E,
    0x01,
    0x00, 0x00, 0x00, 0x6F,
    0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10,
    0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F, 0x20,
    0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F, 0x30,
    0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F, 0x40,
    0x04, 0x12,  /* Start third CAN/CAN FD ACF message */
    0x0E,
    0x01,
    0x00, 0x00, 0x00, 0x6F,
    0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11,
    0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F, 0x20, 0x21,
    0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F, 0x30, 0x31,
    0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F, 0x40, 0x41,
    0x04, 0x12, /* Start fourth CAN/CAN FD ACF message */
    0x0E,
    0x01,
    0x00, 0x00, 0x00, 0x6F,
    0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12,
    0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F, 0x20, 0x21, 0x22,
    0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F, 0x30, 0x31, 0x32,
    0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F, 0x40, 0x41, 0x42,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00 /* 6 remaining bytes */
};
```

Figure 67. Eth2Can_Frame definition

4.4 Run the example with the external loopback config

As previously stated, the default setting of the example project is not sending the CAN/ETH frames outside EVB hence you could not observe the actual CAN/ETH frames on the peripheral pins. This chapter guides how to change the setup to make it observable and run the project. Tested board was S32Z27X-DC as in the below figure. To run the modified example project, connect RJ-45 to your PC via the Ethernet cable, and connect J61 and J63 as a CAN external loopback connection. Then, you can observe the Ethernet frame by the Ethernet frame capturing tool such as Wireshark. Also you can observe CAN frames by probing J62 (3.3V single end Tx / Rx signals) with a logic analyzer.

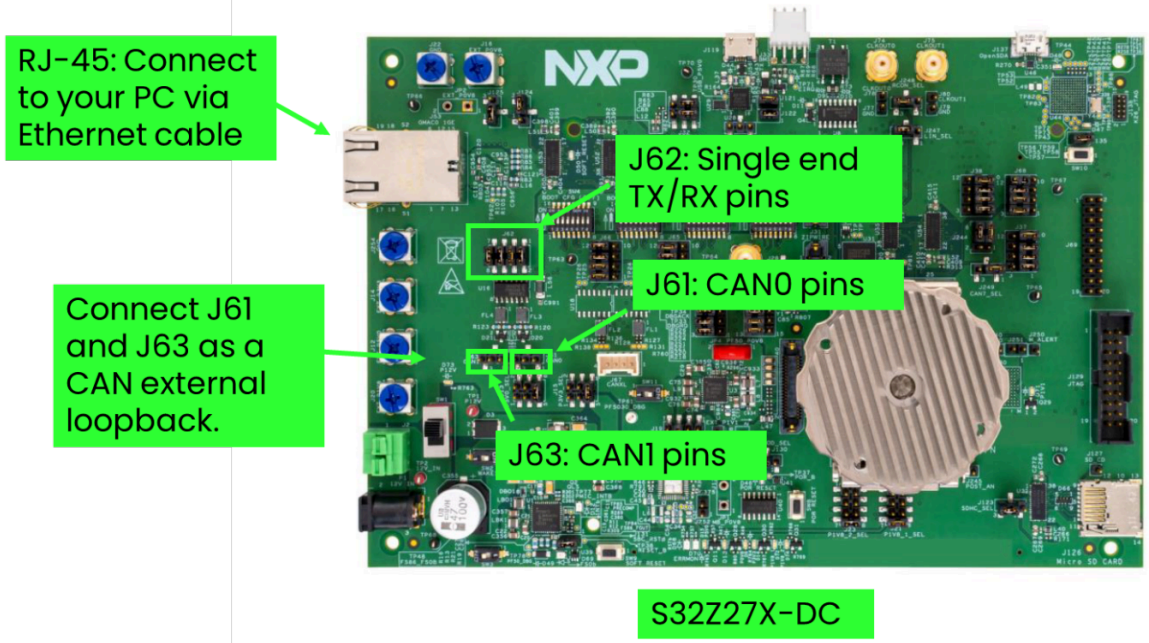

Figure 68. S32Z27X-DC EVB

Corresponding schematic is as below.

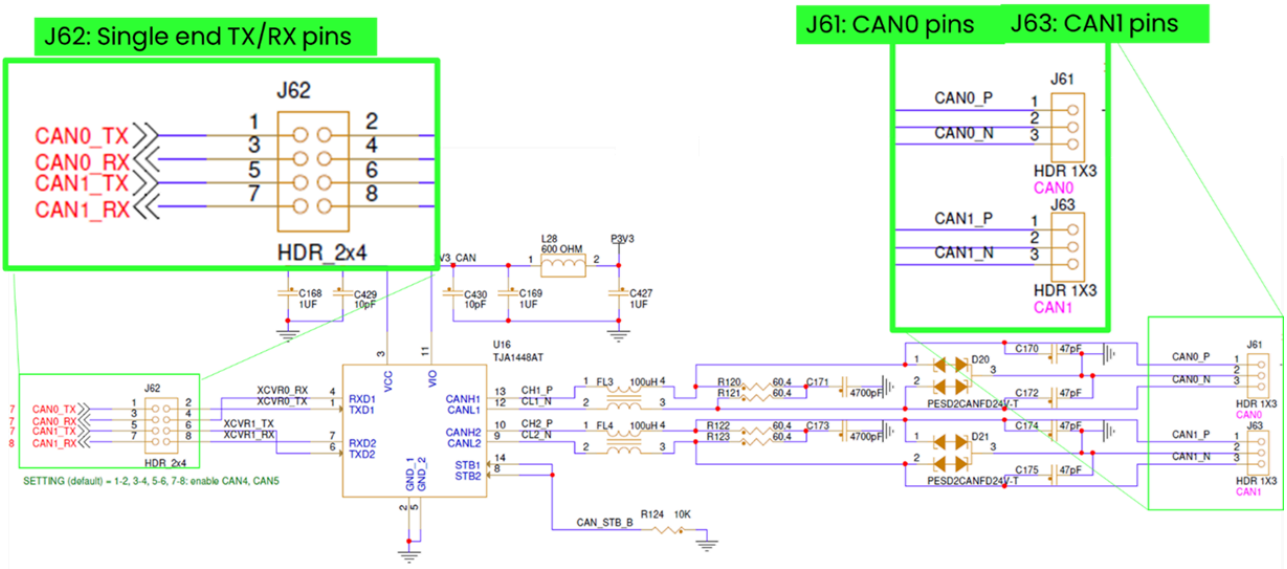

Figure 69. S32Z27X-DC EVB CAN Schematic

As for your PC's LAN adapter setting, Ensure the NPCAP is enabled. Disabling other protocols are recommended to suppress unnecessary traffic on the Ethernet connection between S32Z and your PC.

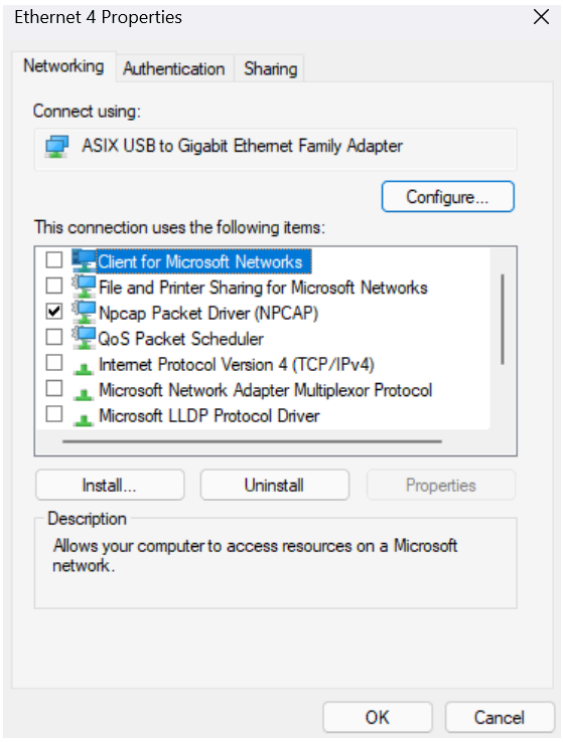

Figure 70. Ethernet Adapter setting

4.4.1 How to modify the EB-Tresos config from the internal bus setting to the external loopback setting

On your EB-Tresos Studio, import and load the default Tresos Config as previously explained. Then, change the CANHUB config as below figure.

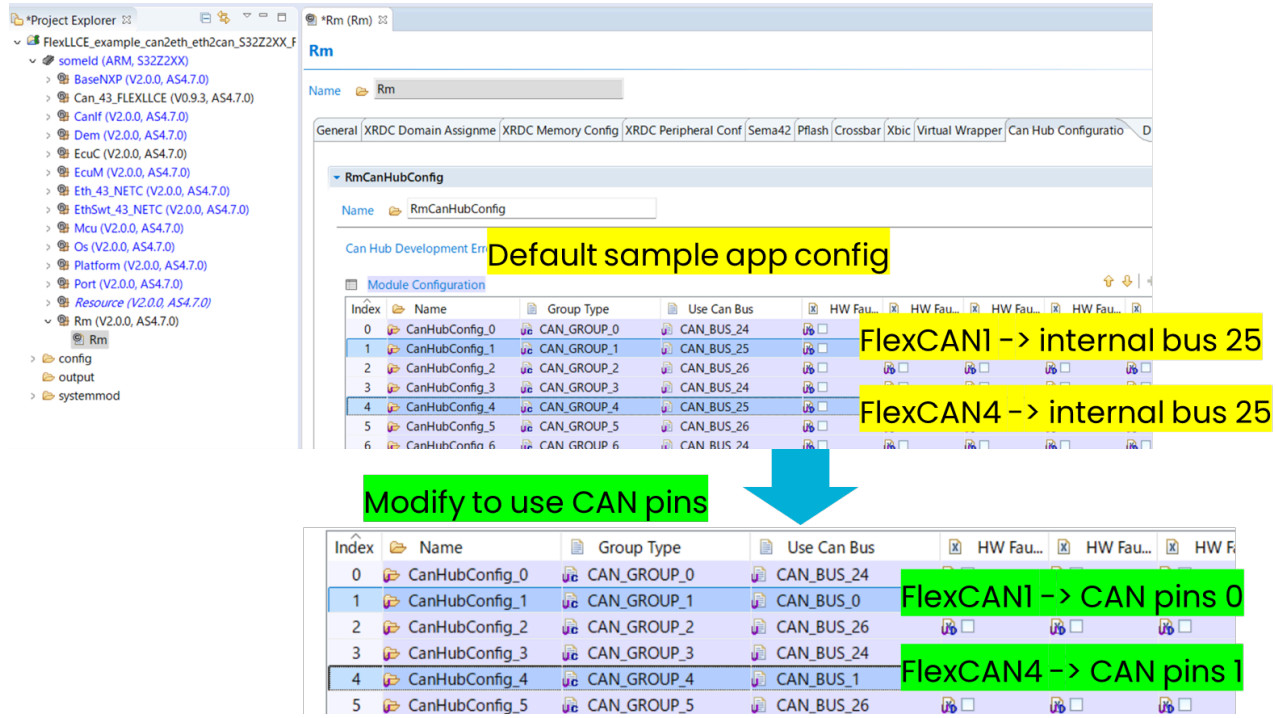

Figure 71. Tresos Config modification

Once the modification is done, generate it.

4.4.2 Modify main.c

Main,c resides under the folder src of the project. In default, main.c sets the Ethernet PHY on the EVB in the loopback mode. So, delete those corresponding snippet to cancel the loopback mode. Delete the lines 185-215 and 236-272 in the main.c as below figure.

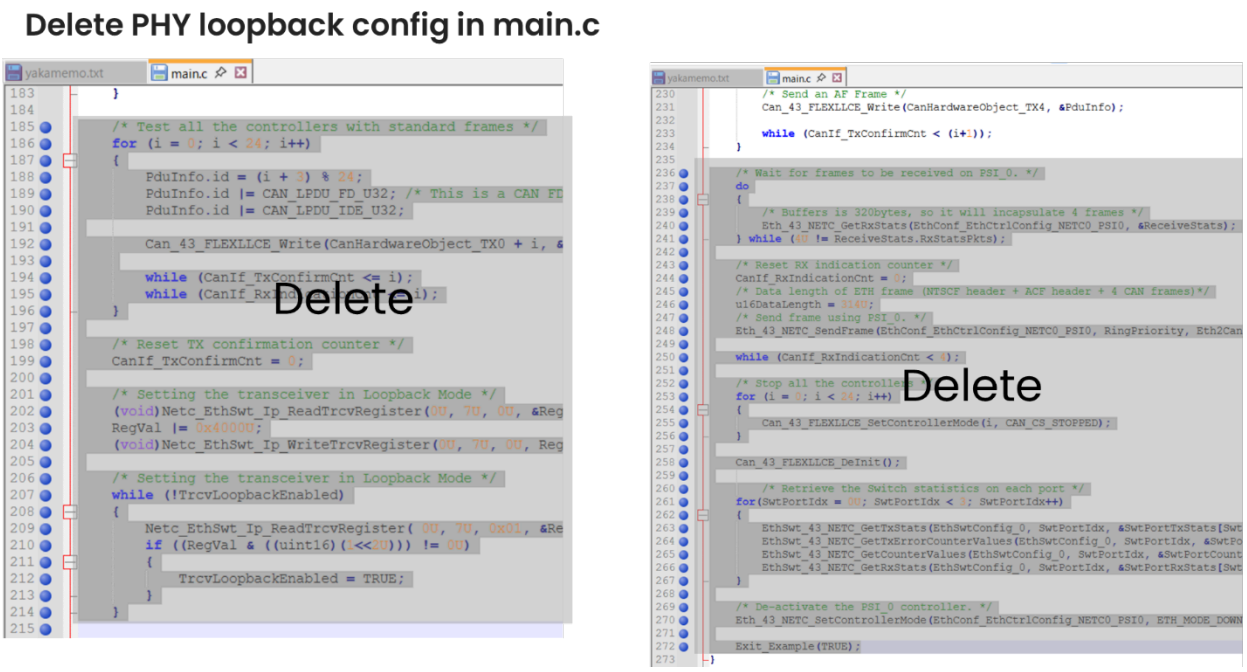

Figure 72. Modification of main.c

4.4.3 Build, run and Capture CAN2ETH frames

On the Linux emulated environment such as Cygwin, run the command “make build” as previously described. Then, run “run.cmm” under the folder “debug” as previously described. If you press Go on the TRACE32 after starting capture by the Wireshark and the Logic analyzer probing CAN0-RXD, you will see as below figures.

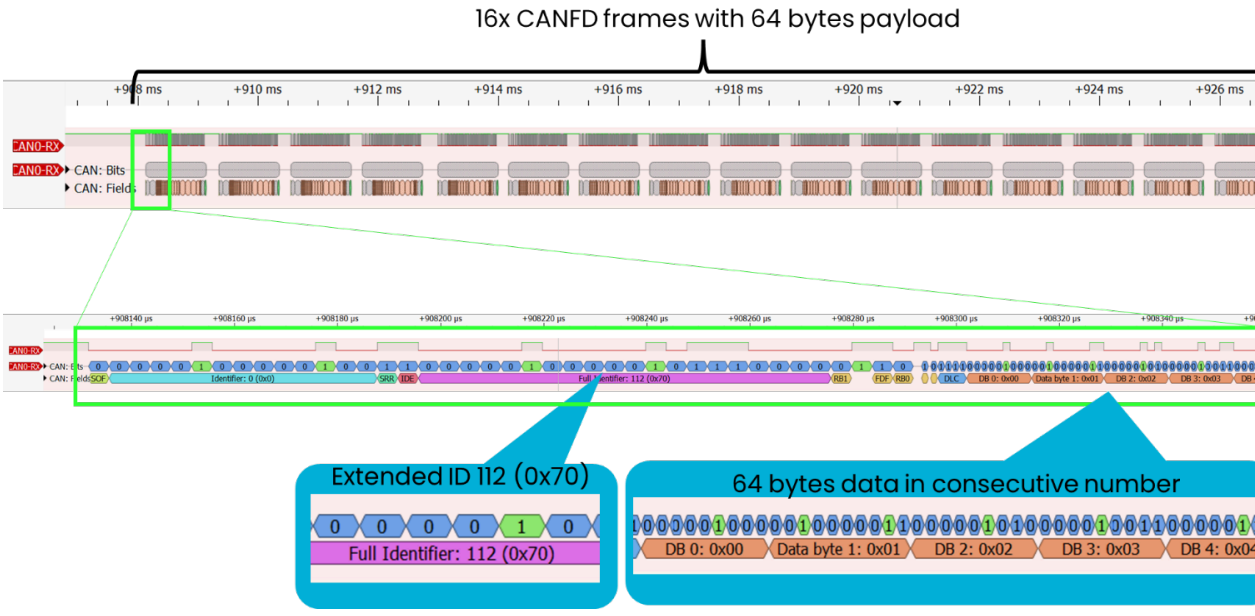

Figure 73. Captured CAN frames on Logic Analyzer

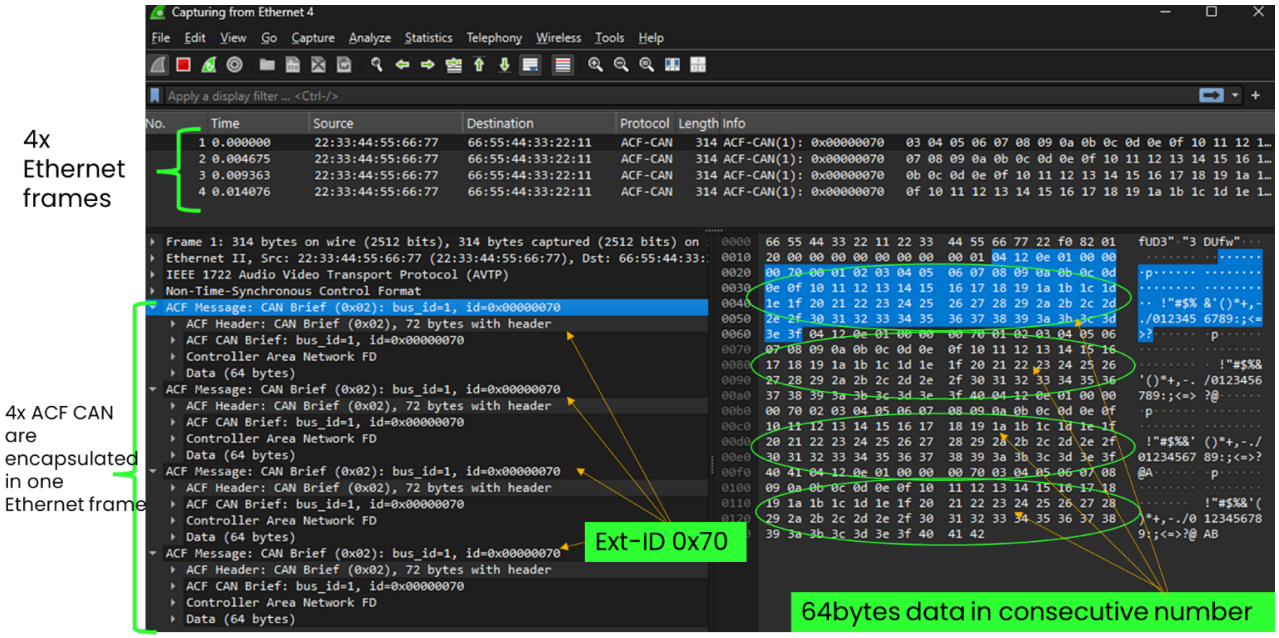

Figure 74. Captured CAN2ETH frames on Wireshark

4.4.4 Capture ETH2CAN frame activity

FlexLLCE is using VSI7 of NETC. The MAC address of VSI7 is configured in the Tresos config (default value is 66:55:44:33:22:22). If you change the destination address of the captured CAN2ETH frame to the configured value, you can create the ETH2CAN frame for this project.

If you would like to change the MAC address of VSI7, refer below figure, which is showing the MAC address configuration location in the EB-Tresos Studio GUI.

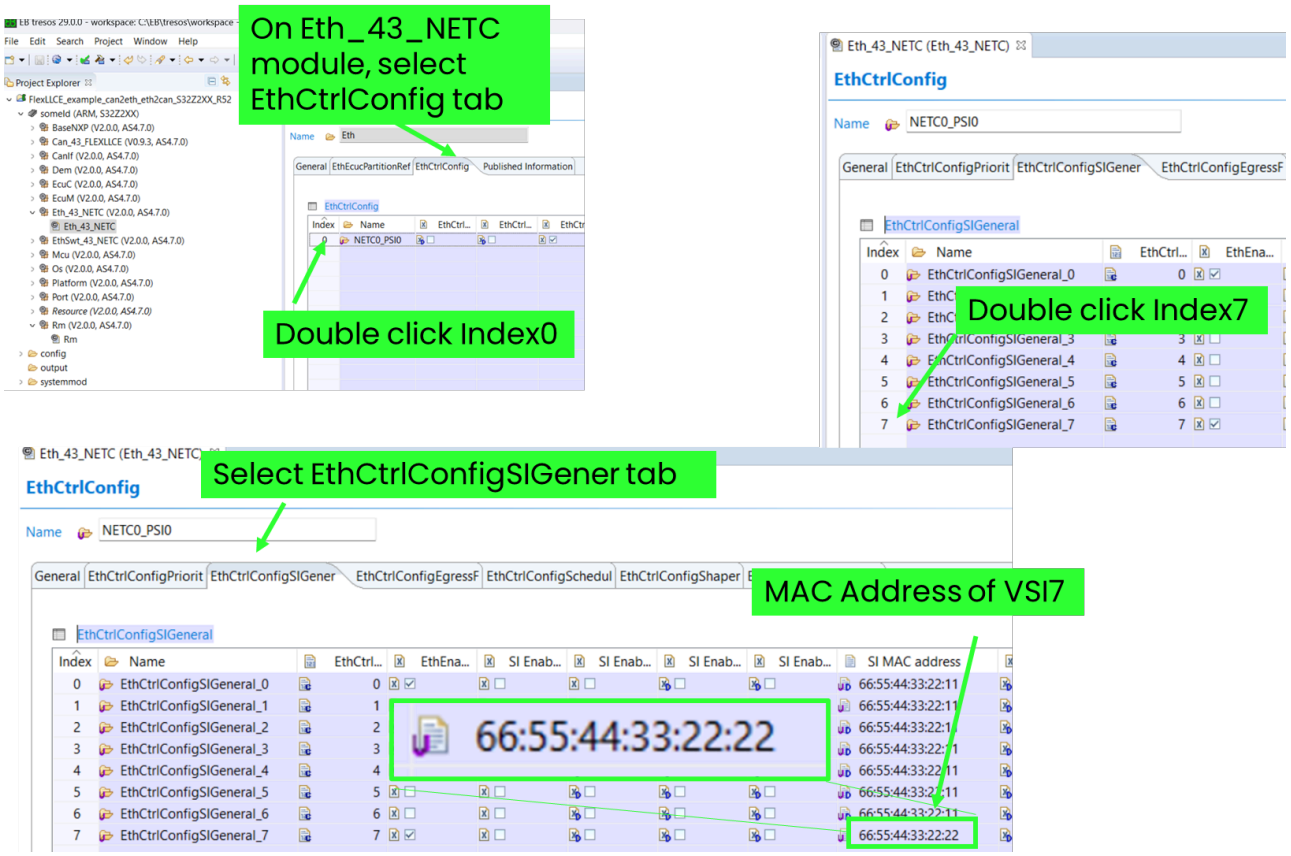

Figure 75. MAC address of VSI7

So, if you change the destination MAC in the captured CAN2ETH packet as below and send it to the S32Z as below, you will observe the 4x CANFD frames sent from CAN1 pins. (This example also changes the Src MAC actually but it's not mandatory.)

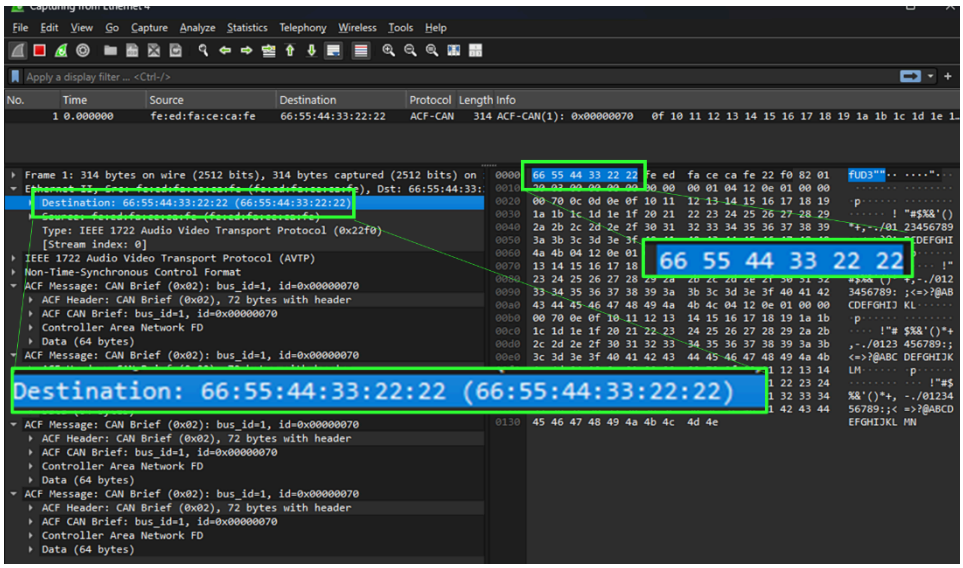

Figure 76. Change MAC address

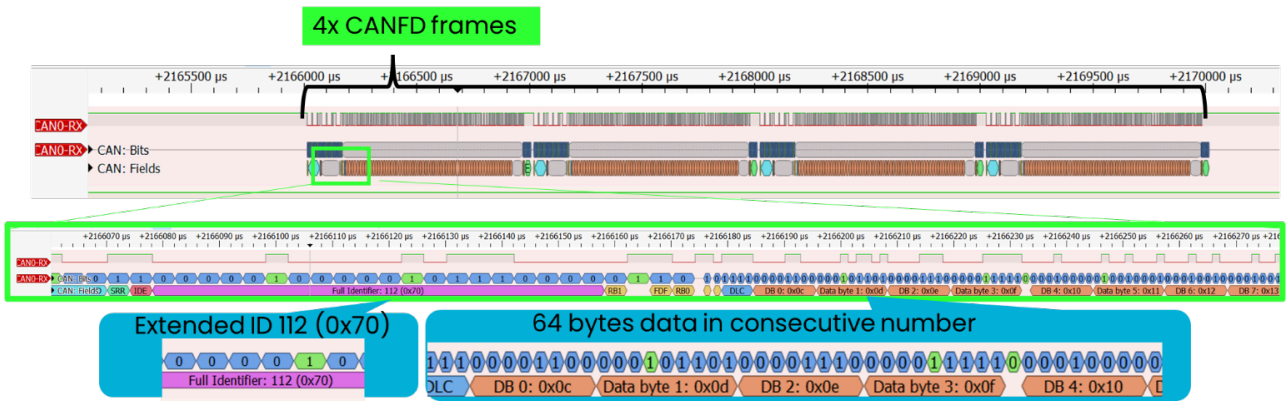

Figure 77. Decapsulated CAN frames by ETH2CAN

5 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

6 Revision history

Table 1. Revision history

Document ID	Release date	Description
AN14711 v.1.0	04 June 2025	Initial release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Tables

Figures

Contents