S32G-VNP-RDB3

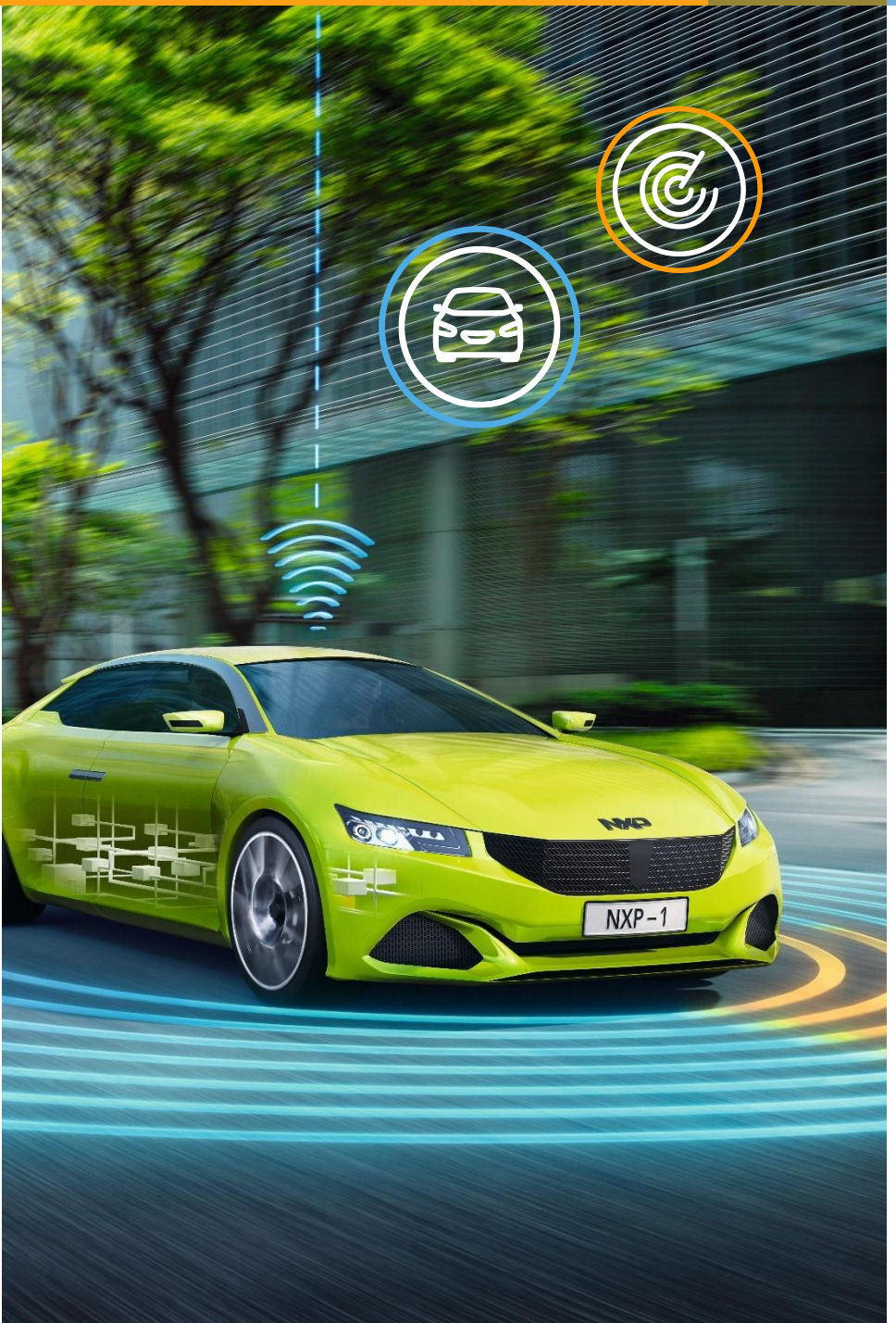
REAL TIME DRIVER

EXAMPLE ENABLEMENT GUIDE

PUBLIC

MARCH 2023

SECURE CONNECTIONS
FOR A SMARTER WORLD



CONTENTS

- Hands on UART Real Time Driver example
- Hands on ETH Real Time Driver example
- Hands on CAN Real Time Driver example

HARDWARE REQUIREMENT AND SOFTWARE INSTALLATION

Hardware Requirement

- S32G-VNP-RDB3
- S32 Debug Probe
- AD/DC power supply
- Serial port cable for UART example

Software Installation

- Install S32DS 3.5 according to S32G-VNP-RDB3 Software Enablement Guide
- Install SW32G_RTD_4.4_4.0.0_D2210(RTD) according to S32G-VNP-RDB3 Software Enablement Guide

Hands on UART Example

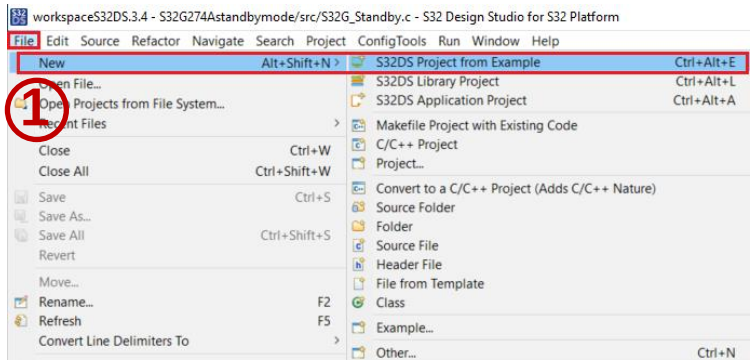
SECURE CONNECTIONS
FOR A SMARTER WORLD

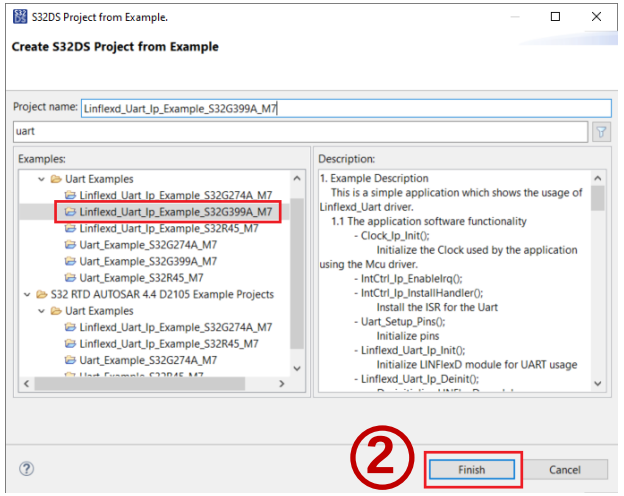
HANDS ON UART: OBJECTIVE

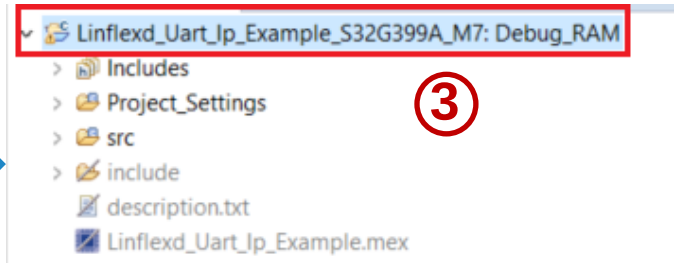
- How to import the UART example into S32DS
- How to configure the clock of UART via S32DS
- How to configure the UART module via S32DS
- How to debug the UART example with S32 debug probe

HANDS ON UART: IMPORT UART EXAMPLE PROJECT

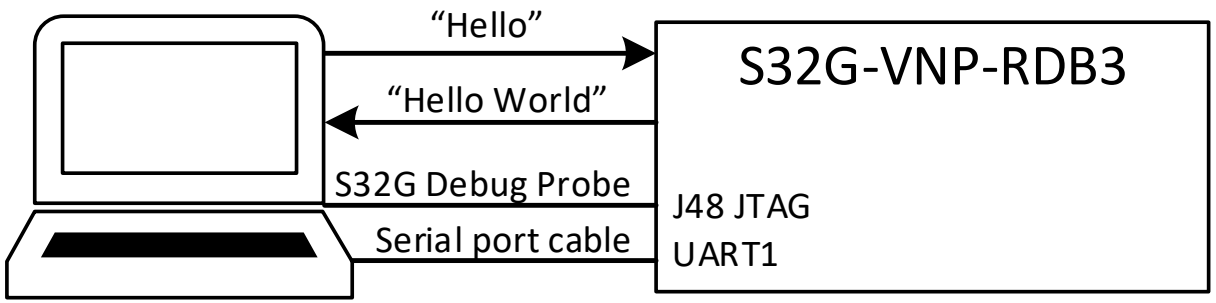
Open S32 Design Studio, go to "File -> New -> S32DS Project From Example". Select "**Linflexd_Uart_Ip_Example_S32G399A_M7**" example, click on "Finish". The project should now be copied into the current workspace.





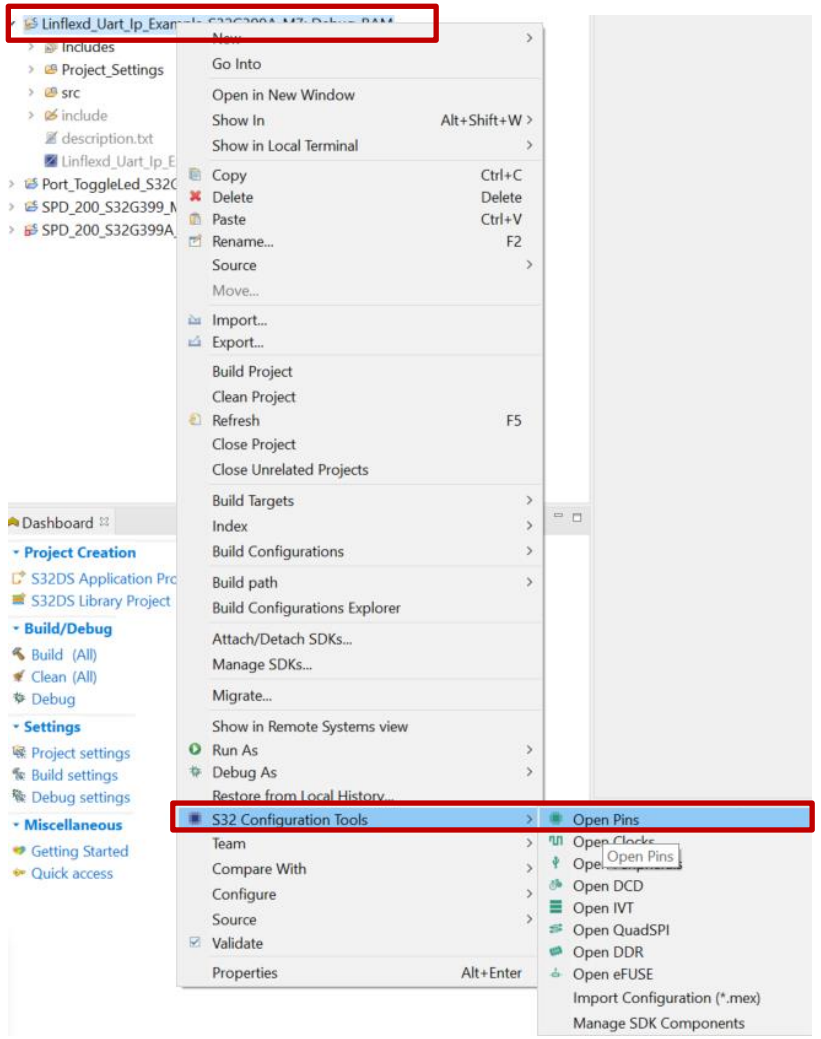


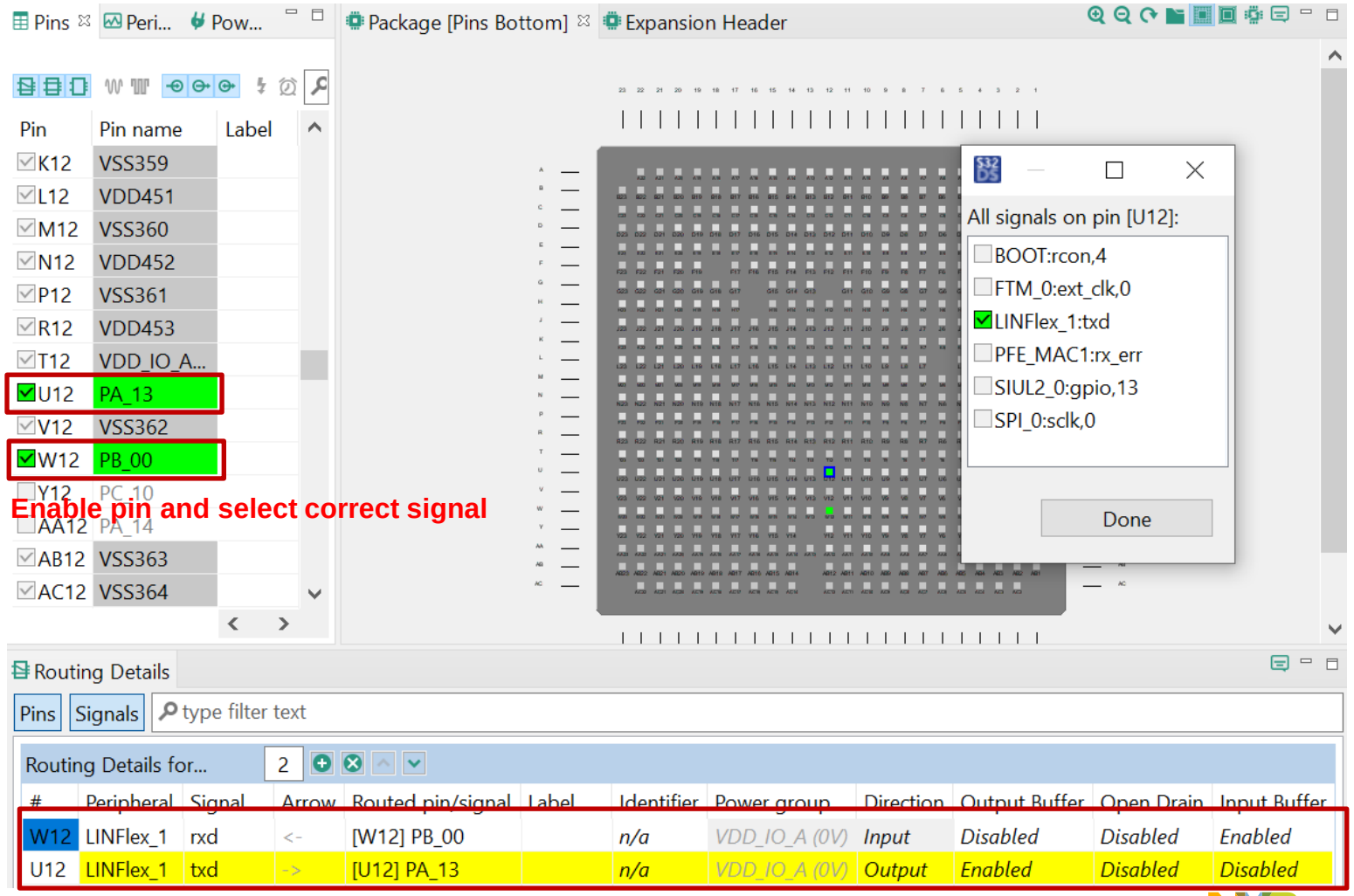
The "Linflexd_Uart_Ip_Example_S32G399A_M7" example is a simple application which shows the usage of UART driver.



HANDS ON UART: PIN CONFIGURATION

Open the **Pins configuration tool**. According to schematic of RDB3, configure pins routing. By default, this example already has corresponding pin routing configuration.





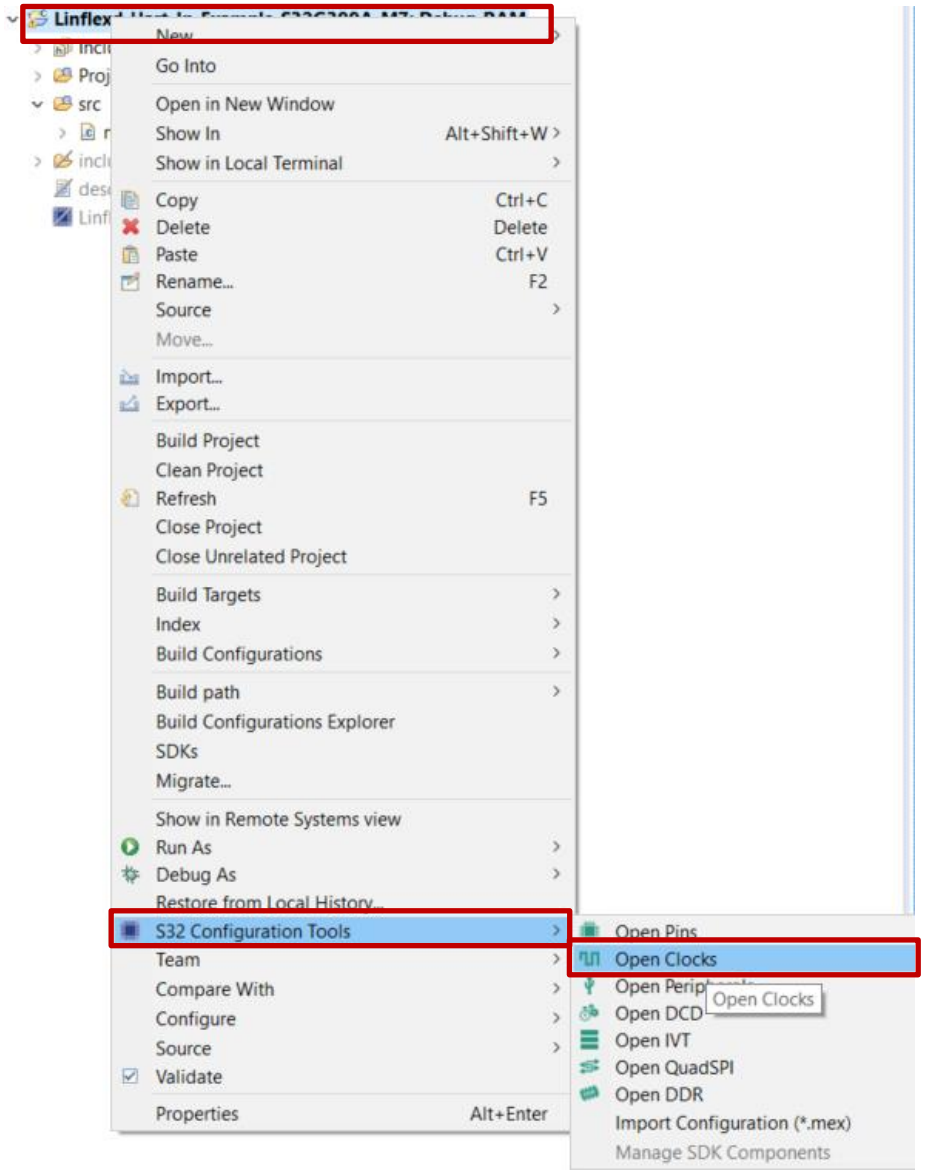
Enable pin and select correct signal

Configure pin's attributes

HANDS ON UART: CLOCK CONFIGURATION 1

Open the Clocks Diagram:

- Right-click the Project
- Select S32 Configuration Tools
- Select Open Clocks

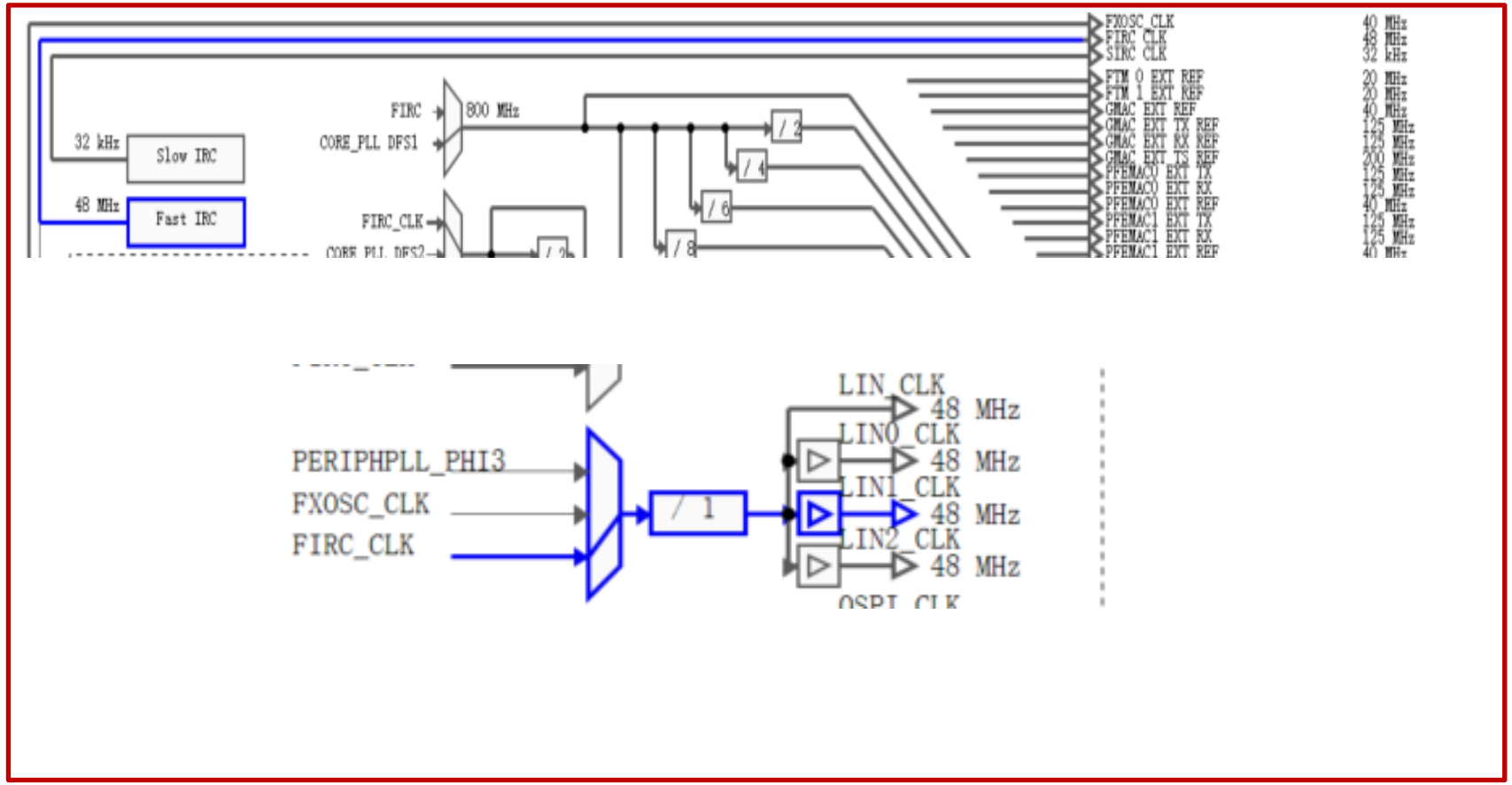


HANDS ON UART: CLOCK CONFIGURATION 2

Open the **Peripheral Clock View**, and double-click the Lin module. The **Clocks Diagram** will show the clock tree of the LinFlexD. The default clock configuration of UART is 48 MHZ which comes from FIRC directly.

Ov... Pe... Co... Re... De... Cl...

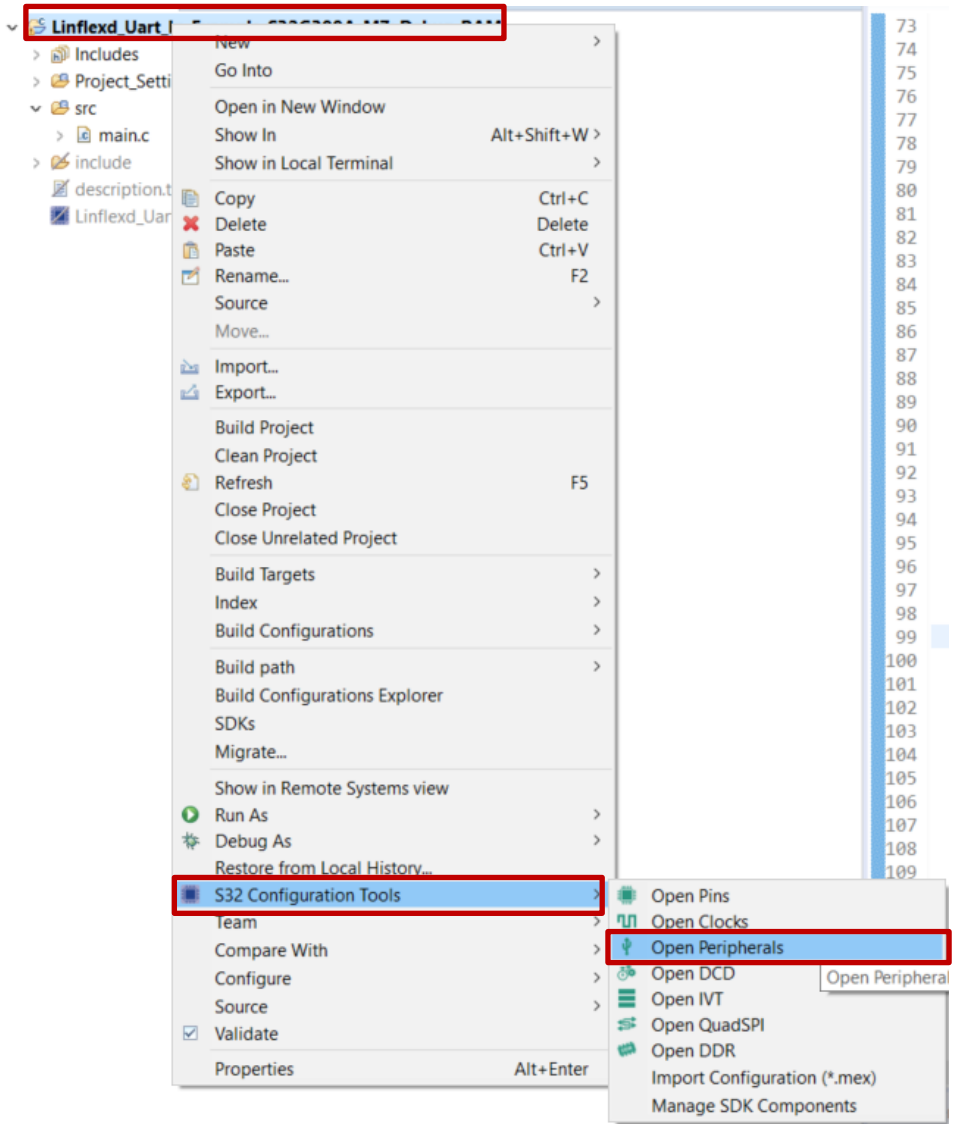
Clock Name	Enable	Source	Divider	Freq
IIC3_CLK	☑	XBA...	/ 1	133.
IIC4_CLK	☑	XBA...	/ 1	133.
LBIST0_CLK	☑	COR...	/ 16	50 M
LBIST1_CLK	☑	COR...	/ 16	50 M
LBIST2_CLK	☑	COR...	/ 16	50 M
LBIST3_CLK	☑	COR...	/ 16	50 M
LBIST4_CLK	☑	COR...	/ 16	50 M
LBIST5_CLK	☑	COR...	/ 16	50 M
LBIST6_CLK	☑	COR...	/ 16	50 M
LBIST7_CLK	☑	COR...	/ 16	50 M
LIN0_CLK	☑	FIRC...	/ 1	48 M
LIN1_CLK	☑	FIRC...	/ 1	48 M
LIN2_CLK	☑	FIRC...	/ 1	48 M
OCOTP0_CLK	☑	FIRC...	/ 1	48 M
PFEMAC0_RX_CLK	☑	FIRC...	/ 1	48 M
PFEMAC0_TX_CLK	☑	FIRC...	/ 1	48 M
PFEMAC1_RX_CLK	☑	FIRC...	/ 1	48 M
PFEMAC1_TX_CLK	☑	FIRC...	/ 1	48 M



HANDS ON UART: UART CONFIGURATION 1

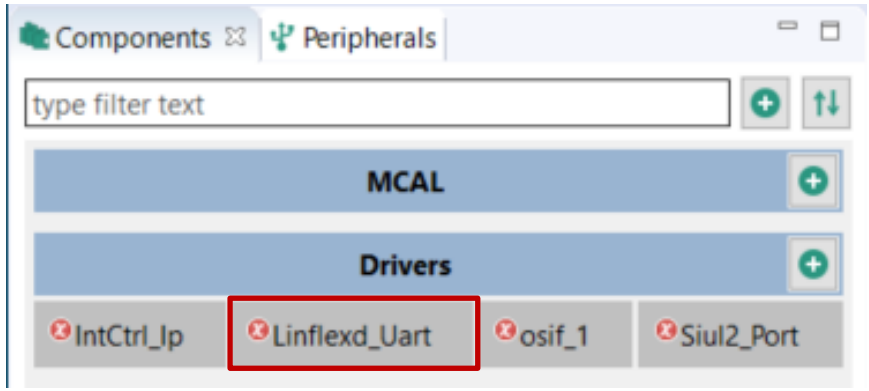
Open the Peripherals Diagram:

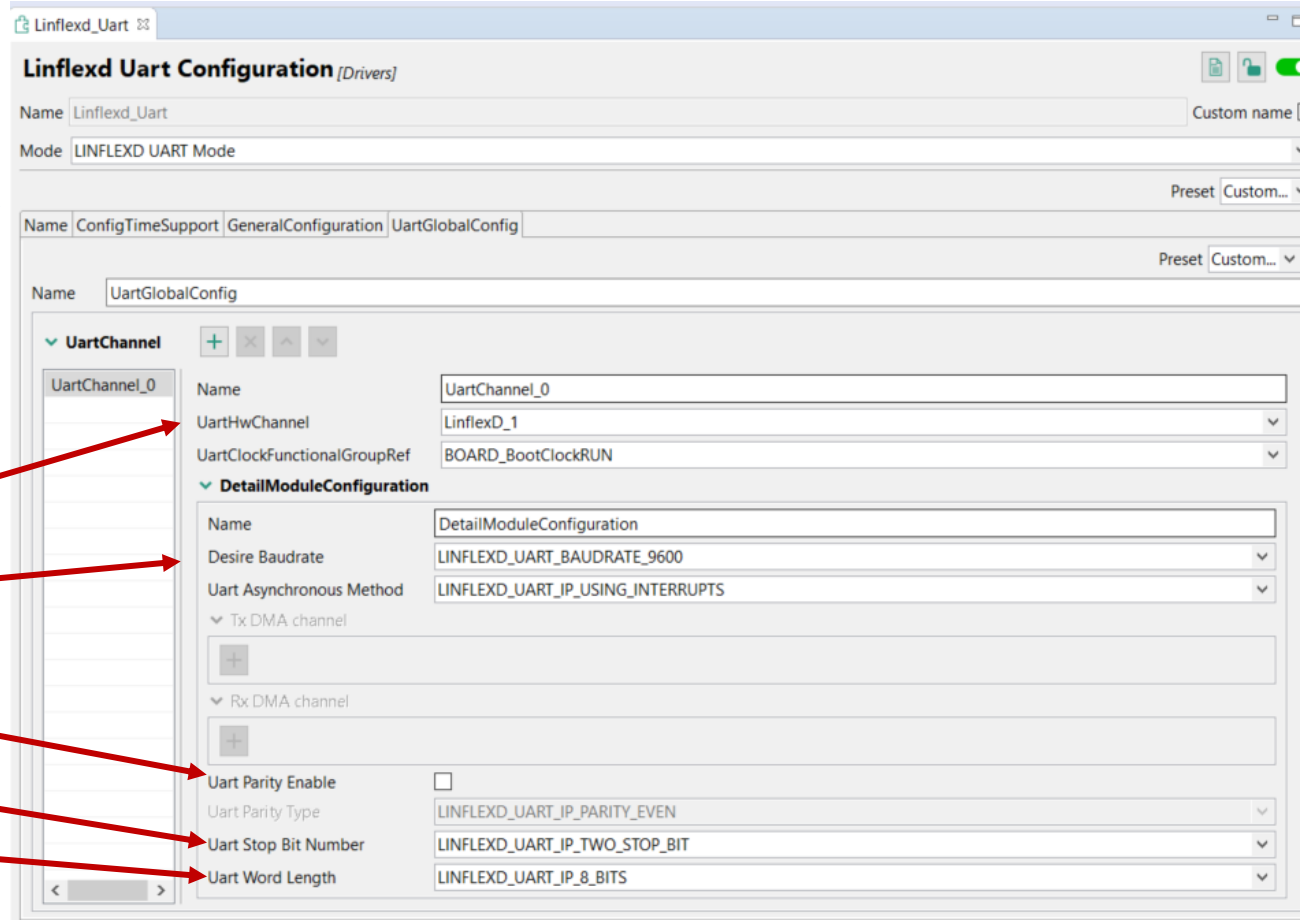
- Right-click the Project
- Select S32 Configuration Tools
- Select Open Peripherals



HANDS ON UART: UART CONFIGURATION 2

The **Components** shows all drivers which used by this example, the **Linflexd_Uart** Component includes the configuration of UART driver





UART module configuration :

- Configure Uart Hardware Channel
- Configure Baudrate
- Configure Uart Parity Enable option
- Configure Uart Stop Bit Number
- Configure Uart Word Length

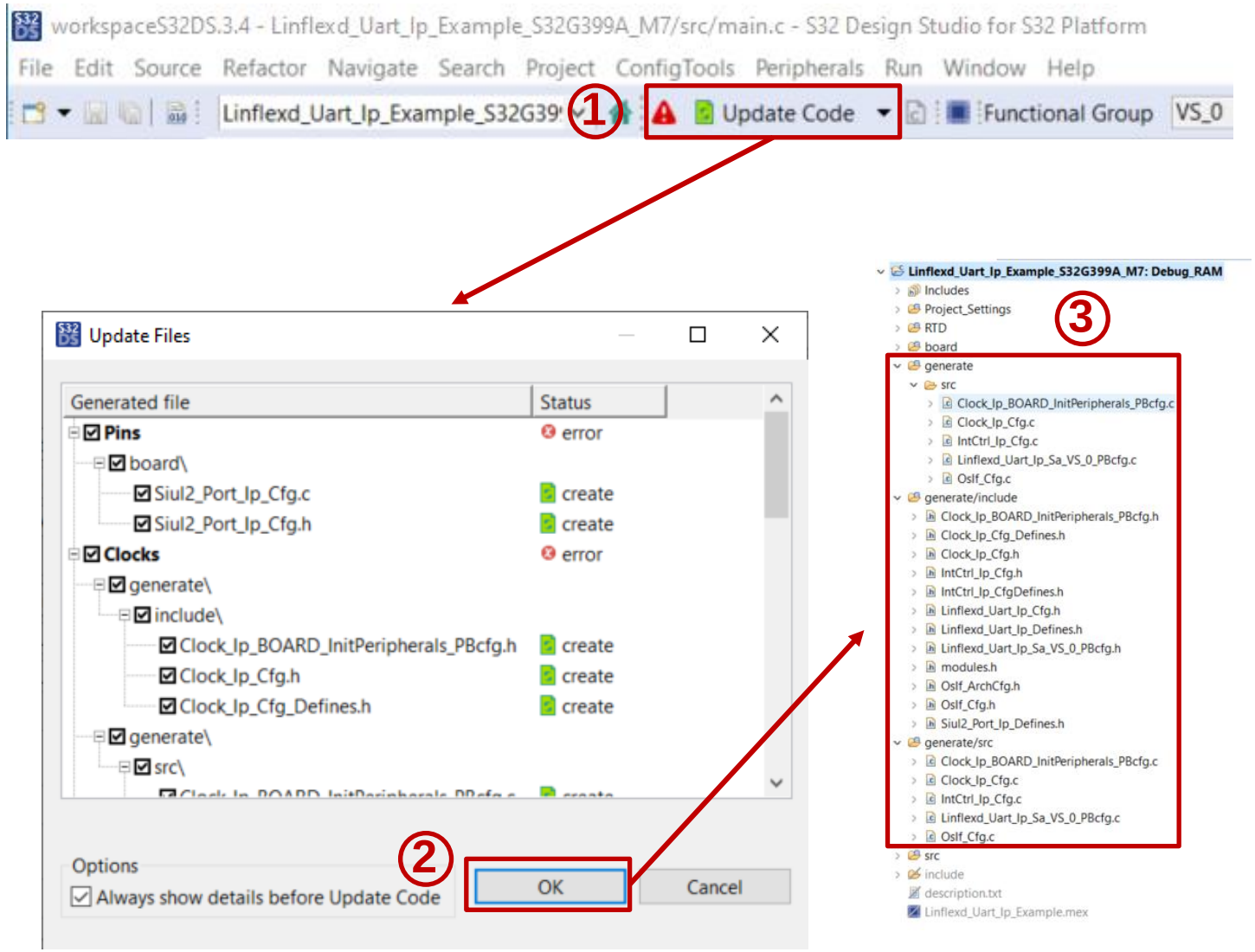
HANDS ON UART: UPDATE CODE

Generate code method:

1. Open the view of any configuration tool, like Pins, then click **Update Code** (ensure desired project is selected)

2. The Update Files window pops up. It shows the detailed update information. Click **OK** button.

3. The configuration .c and .h files will be generated in "generate" folder.



HANDS ON UART: APPLICATION CODE 1

Dissecting the main.c

```c
int main(void)
{
    /* Write your code here */
    /* Buffer to store the received message */
    uint8 rxBuff[RX_MSG_LEN];
    uint32 varRemainingBytes;
    Linflexd_Uart_Ip_StatusType transmitStatus = LINFLEXD_UART_IP_STATUS_ERROR;
    Linflexd_Uart_Ip_StatusType receiveStatus = LINFLEXD_UART_IP_STATUS_ERROR;
    /* Initialize and configure clocks
    *  -   Setup system clocks, dividers
    *  -   see clock manager component for more details
    */
    Clock_Ip_Init(&Mcu_aClockConfigPB[0]);

    /* Interrupts controller initialization */
    IntCtrl_Ip_Init(&IntCtrlConfig_0);
    IntCtrl_Ip_ConfigIrqRouting(&intRouteConfig);

    /* Initialize pins */
    Siul2_Port_Ip_Init(NUM_OF_CONFIGURED_PINS0, g_pin_mux_InitConfigArr0);

    /* Initialize LINFlexD module for UART usage */
    Linflexd_Uart_Ip_Init(LINFLEXD_INSTANCE, &Linflexd_Uart_Ip_xHwConfigPB_1_VS_0);

    /* Send the greeting message to console */
    Linflexd_Uart_Ip_AsyncSend(LINFLEXD_INSTANCE, (uint8*)WELCOME_MSG, strlen(WELCOME_MSG));

    /* Wait for data send */
    while(Linflexd_Uart_Ip_GetTransmitStatus(LINFLEXD_INSTANCE, &varRemainingBytes) == LINFLEXD_UART_IP_STATUS_BUSY);
    transmitStatus = Linflexd_Uart_Ip_GetTransmitStatus(LINFLEXD_INSTANCE, &varRemainingBytes);
```

MCU clock initialization and Interrupt initialization

Initialize pins

Initialize LINFlexD module

Send the greeting message to console

HANDS ON UART: APPLICATION CODE 2

Dissecting the main.c

```
/* Infinite loop */
for( ;; )
{
    /* Get the message sent by user from the console, using a-sync method */
    Linflexd_Uart_Ip_AsyncReceive(LINFLEXD_INSTANCE, rxBuff, strlen("Hello"));
    /* Wait for data receive */
    while(Linflexd_Uart_Ip_GetReceiveStatus(LINFLEXD_INSTANCE, &varRemainingBytes) == LINFLEXD_UART_IP_STATUS_BUSY);
    receiveStatus = Linflexd_Uart_Ip_GetReceiveStatus(LINFLEXD_INSTANCE, &varRemainingBytes);
    /* If the user typed "Hello", reply with the "Hello world!" message again */
    if(strcmp((char*)rxBuff, "Hello") == 0)
    {
        Linflexd_Uart_Ip_AsyncSend(LINFLEXD_INSTANCE, (uint8*)MSG, strlen(MSG));
        /* Wait for data send */
        while(Linflexd_Uart_Ip_GetTransmitStatus(LINFLEXD_INSTANCE, &varRemainingBytes) == LINFLEXD_UART_IP_STATUS_BUSY);
        break;
    }
    else
    {
        Linflexd_Uart_Ip_AsyncSend(LINFLEXD_INSTANCE, (uint8*)ERROR_MSG, strlen(ERROR_MSG));
        /* Wait for data send */
        while(Linflexd_Uart_Ip_GetTransmitStatus(LINFLEXD_INSTANCE, &varRemainingBytes) == LINFLEXD_UART_IP_STATUS_BUSY);
        break;
    }
}
```

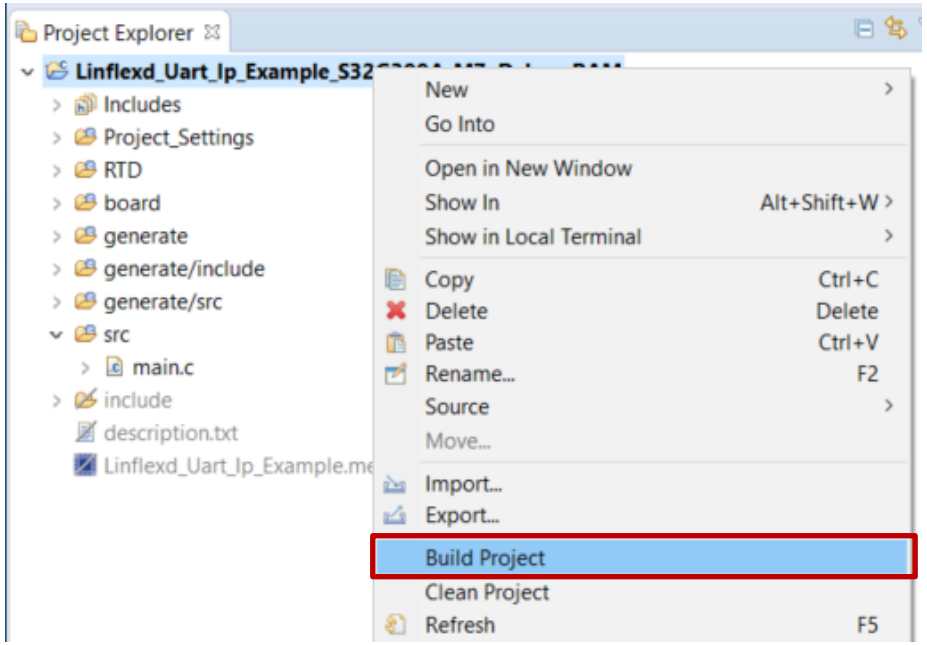
Receive data from user

Echo the received data back

HANDS ON UART: BUILD AND DEBUG 1

Build the target :

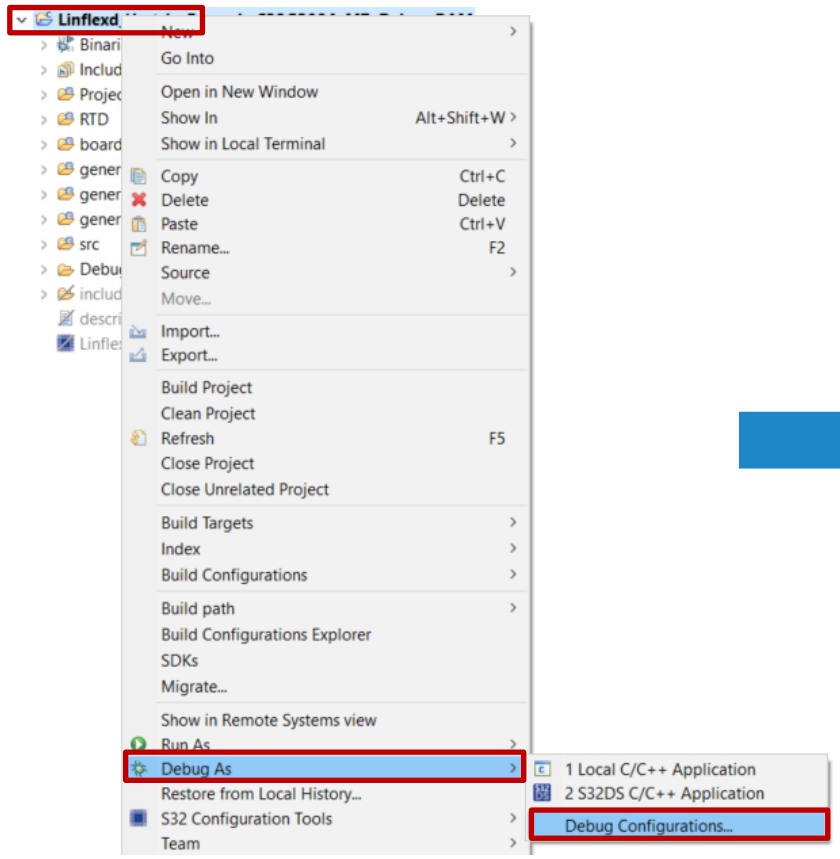
- Right-click the Project
- Select Build Project
- Print Build information on Console window
- Linflexd_Uart_Ip_Example_S32G399A_M7.elf is generated



HANDS ON UART: BUILD AND DEBUG 2

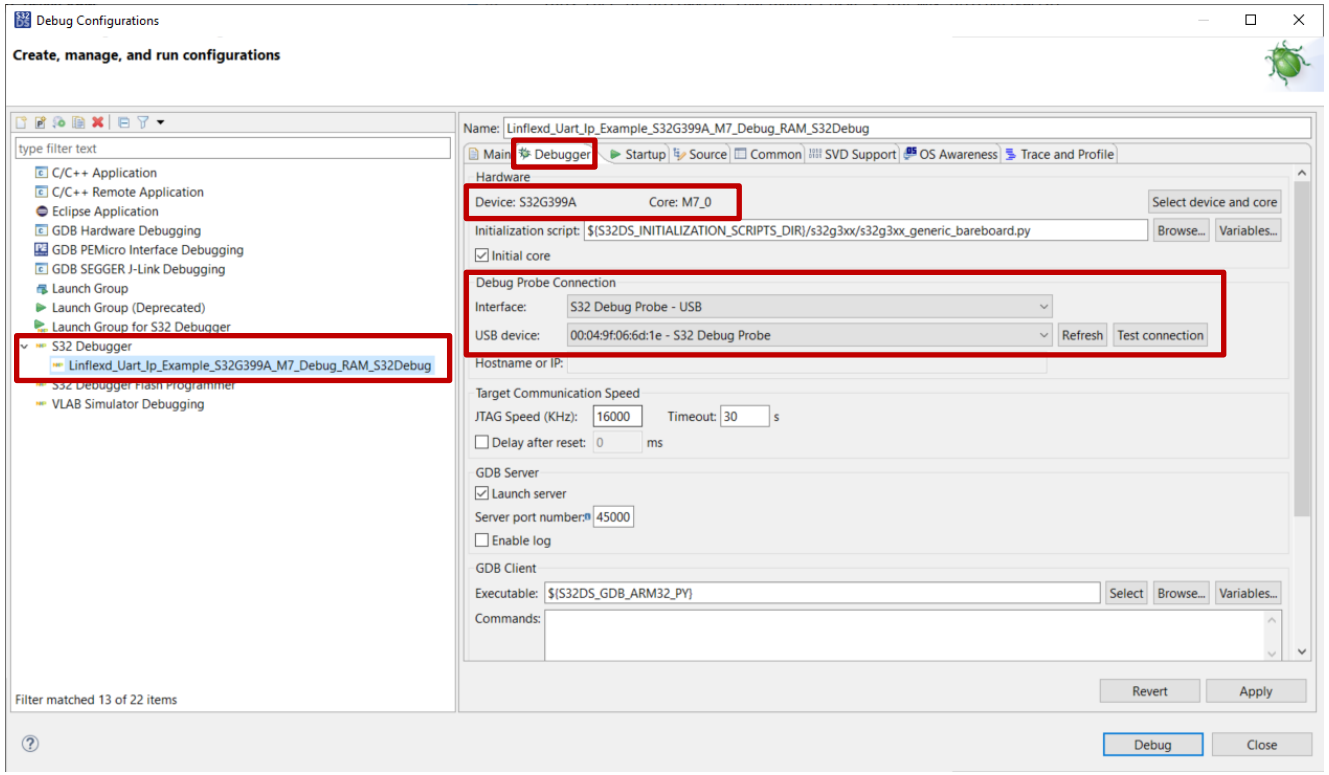
Go to debug configuration:

- Right-click the Project
- Select the Debug As
- Click on Debug Configurations



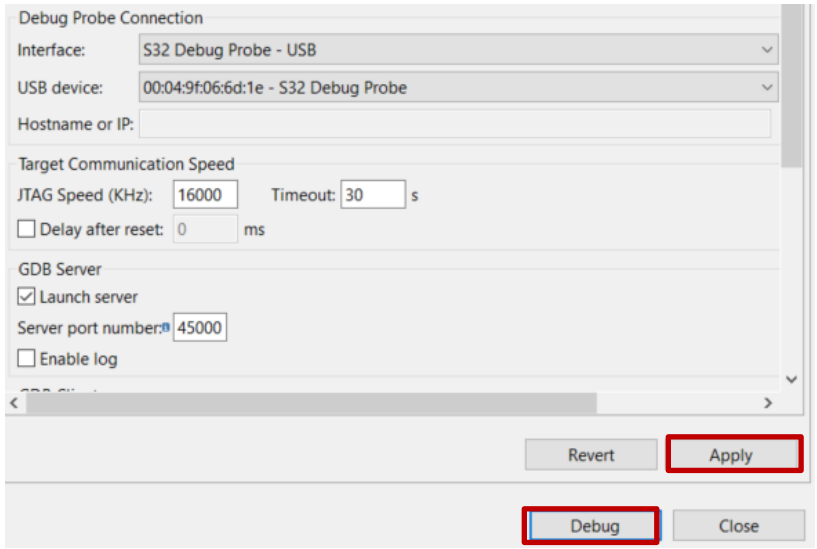
Debug configuration setting:

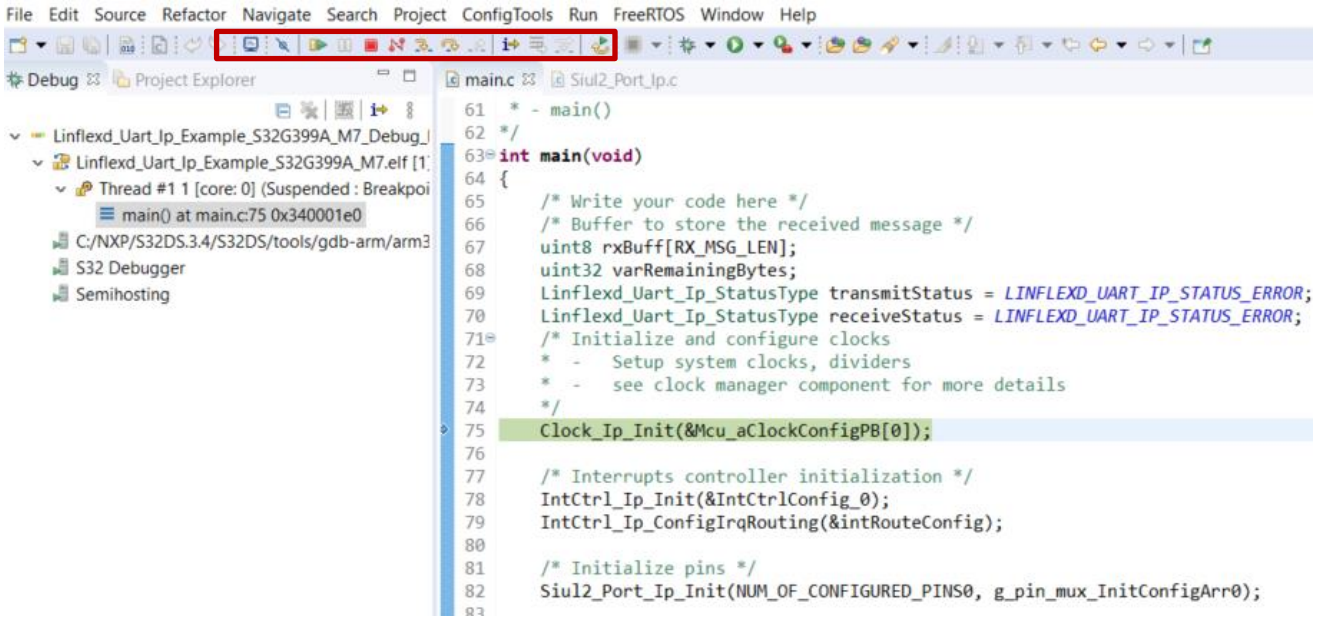
- Connect the S32 Debug probe with PC and RDB3
- Click on target project
- Select the target device and core as S32G399A_M7_0
- Select target S32 Debug Probe



HANDS ON UART : DEBUG AND RUN

Power on the RDB3, click on "Apply", then click on "Debug". The view will switch to the Debug Perspective, and you can use the controls to control the program flow.

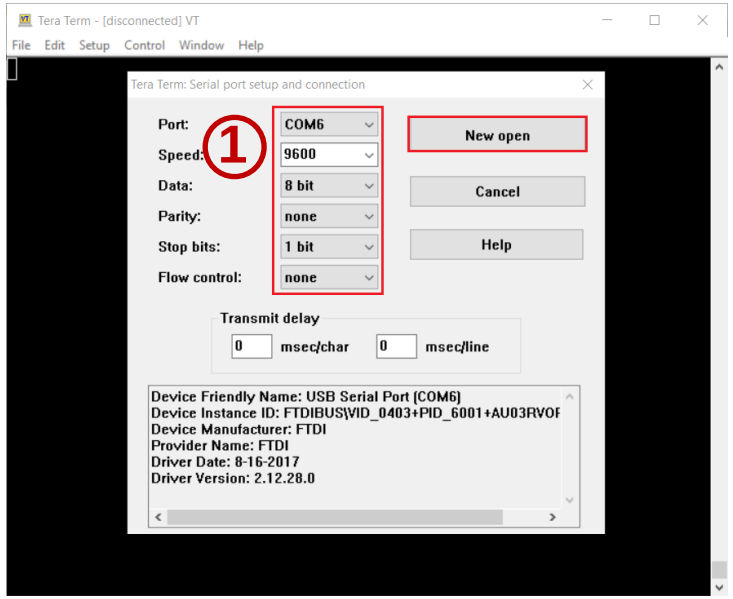


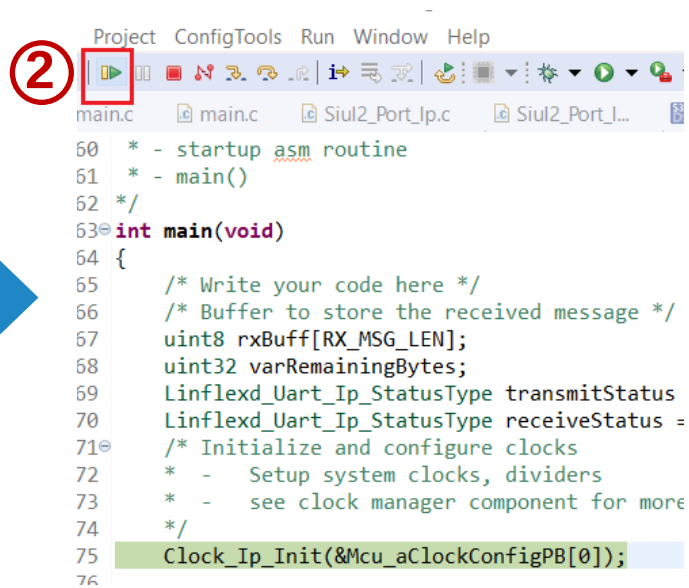


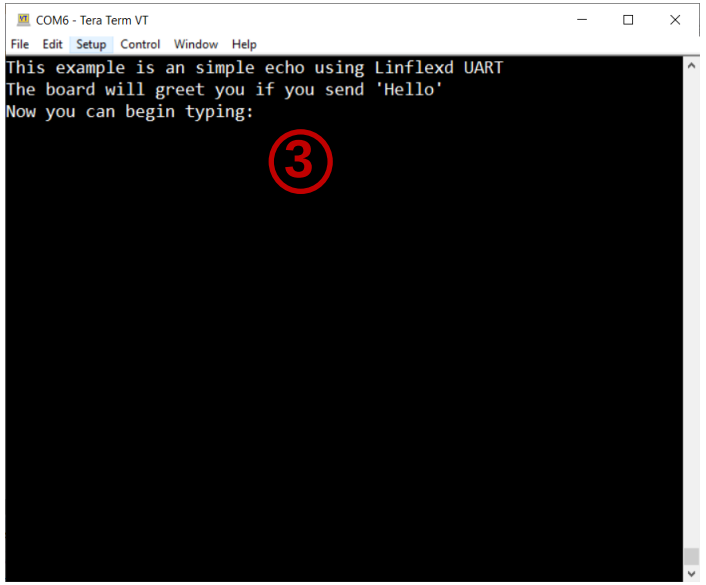
HANDS ON UART: TEST RESULT

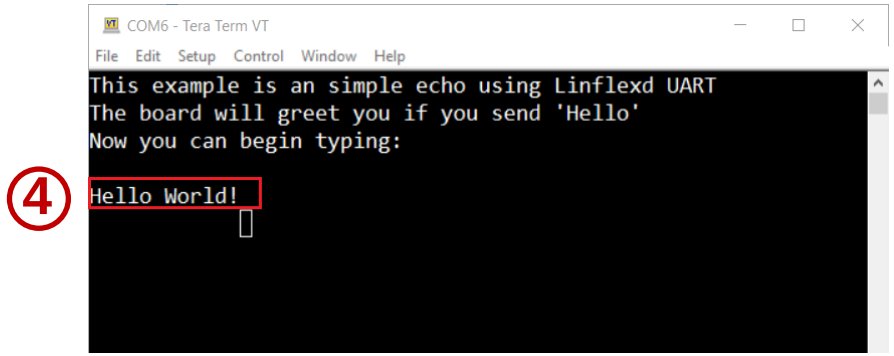
Check the test result:

- Connect UART1 with PC and RDB3
- Open Serial terminal like Tera Term and configure the serial port
- Click on the Resume option in Debug view
- The Serial terminal will print messages
- Then input "Hello" in step 3
- UART1 will output “Hello World!” back









Hands on ETH Example

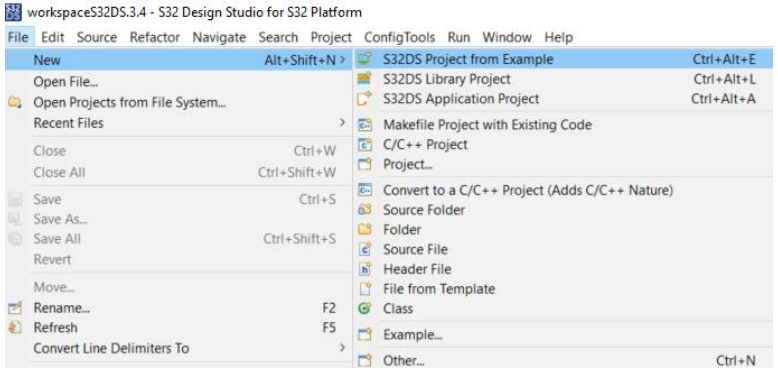
SECURE CONNECTIONS
FOR A SMARTER WORLD

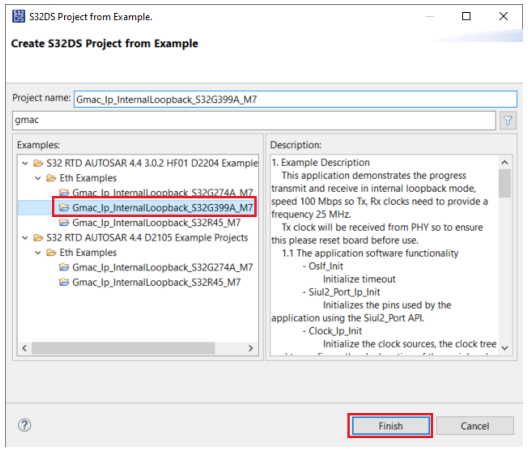
HANDS ON ETH – OBJECTIVE

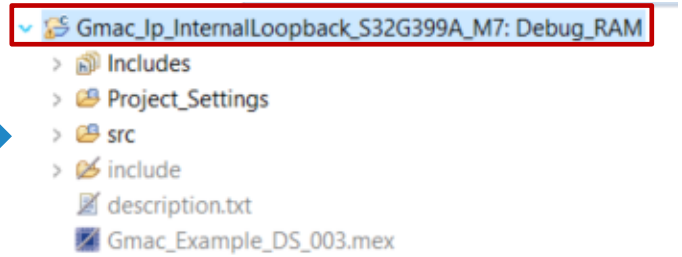
- How to import the ETH example into S32DS
- How to configure the clock of ETH via S32DS
- How to configure the port of ETH via S32DS
- How to use the ETH module to transmit/receive ETH frame
- How to debug the ETH example using S32 debug probe

HANDS ON ETH: IMPORT ETH EXAMPLE PROJECT

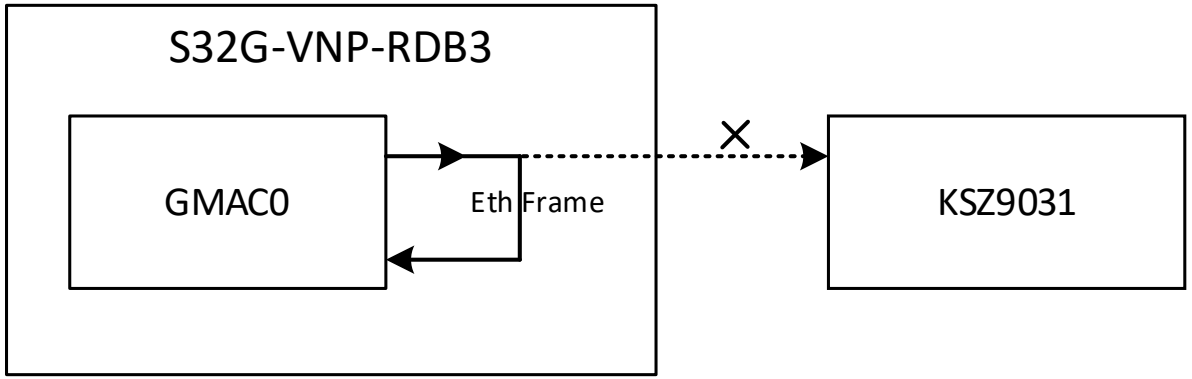
Open S32 Design Studio, go to "File -> New -> S32DS Project From Example". Select "**Gmac_Ip_InternalLoopback_S32G399A_M7** " example, then click on "Finish". The project is copied into the current workspace.







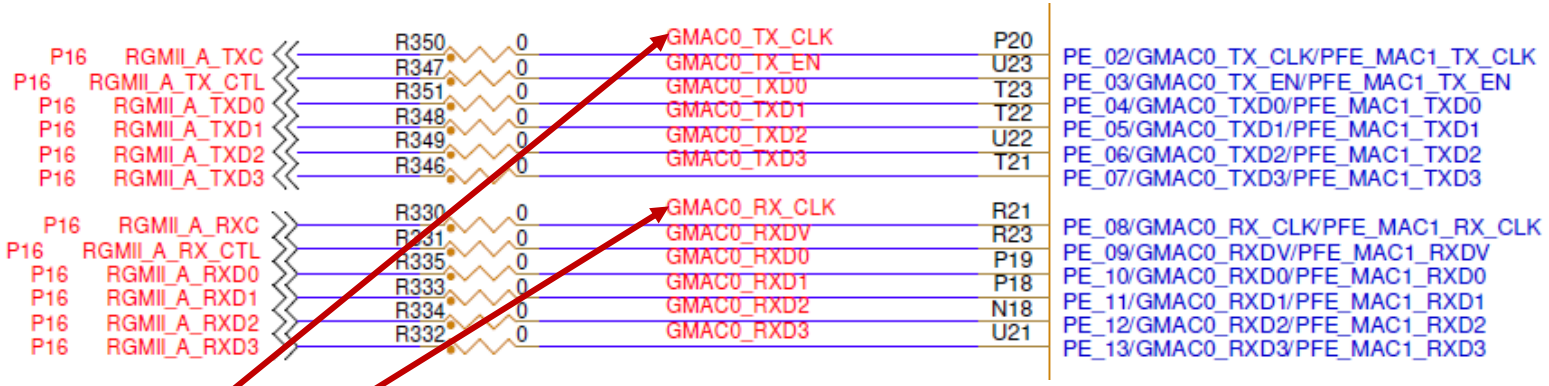
This “Gmac_Ip_InternalLoopback_S32G399A_M7” example demonstrates the GMAC transmission and reception in internal loopback mode. The ETH frame is transmitted back directly through GMAC, and the frame will not be transmitted to PHY.

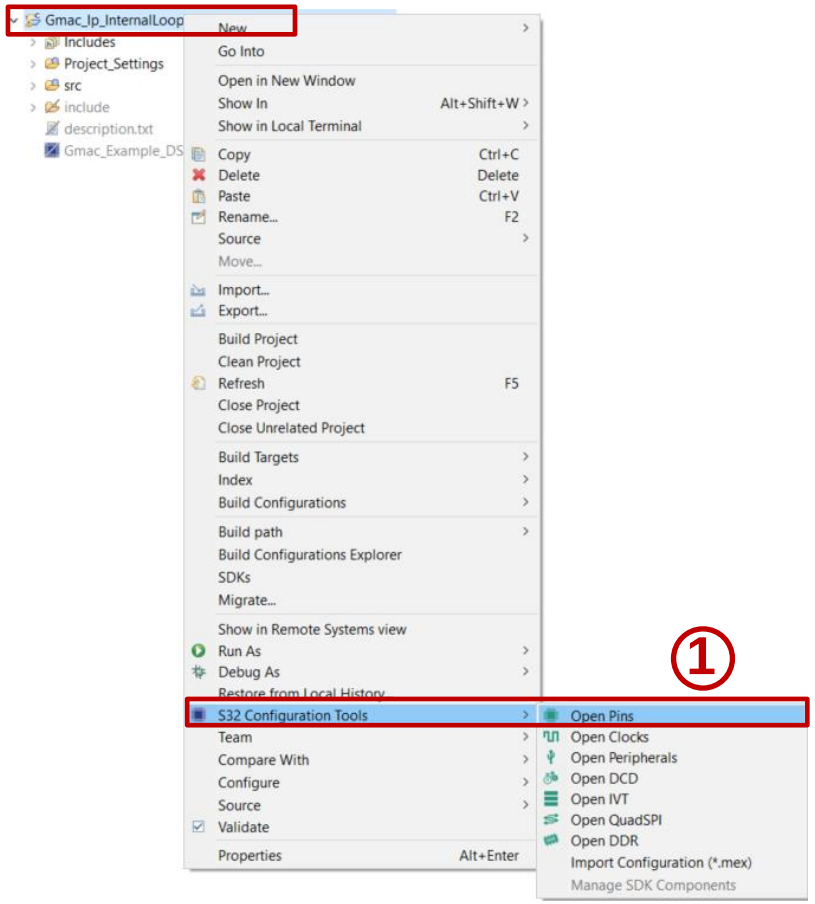


HANDS ON ETH : PORT CONFIGURATION

Pins configuration setting:

- Right-click the Project
- Select S32 Configuration Tools
- Select Open Pins
- Configure pins to provide the external clock to Tx, Rx signals





Routed Pins for BOARD... 3

#	Peripheral	Signal	Route to	Label	Identifier	Power group	Direction	Output Buffer	Open Drain	Input Buffer	Slew Rate Control	Pullup Enable
P20	GMAC_0	tx_clk_o	PE_02		n/a	VDD_IO_GMAC0 (0V)	Output	Enabled	Disabled	Disabled	SIUL2_0: FAST pad: 166MHz(1.8V), 150MHz(3.3V) - GPIO pad: 166MHz(1.8V), 50Mhz(3.3V) / SIUL2_1: 150MHz or lower	Disabled
R21	GMAC_0	rx_clk_i	PE_08		n/a	VDD_IO_GMAC0 (0V)	Input	Disabled	Disabled	Enabled	SIUL2_0: FAST pad: 166MHz(1.8V), 150MHz(3.3V) - GPIO pad: 166MHz(1.8V), 50Mhz(3.3V) / SIUL2_1: 150MHz or lower	Disabled

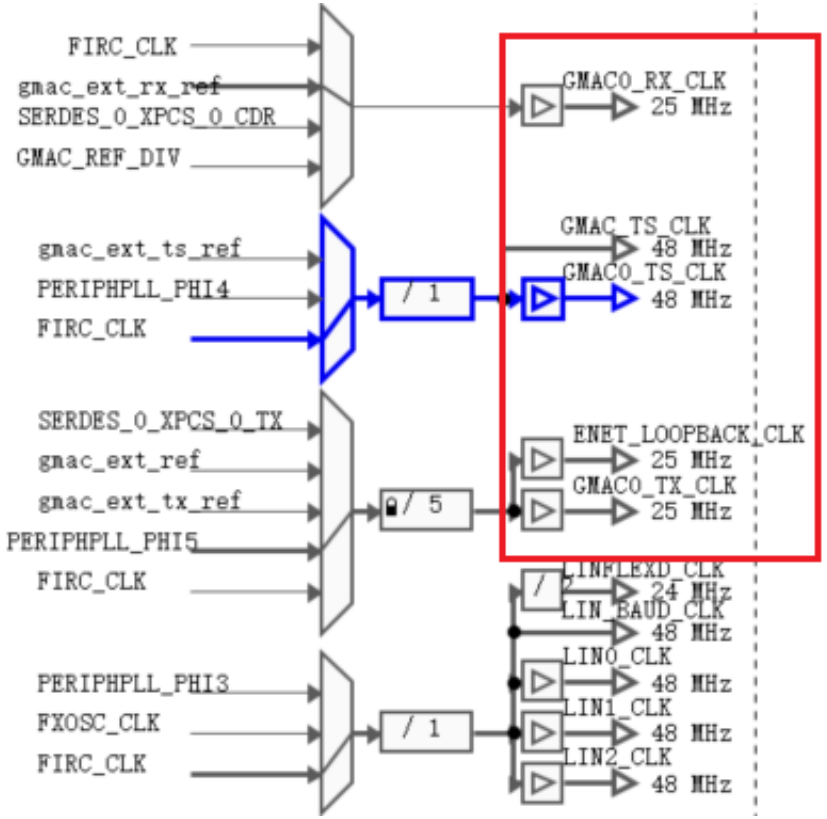
②

HANDS ON ETH : CLOCK CONFIGURATION

Open the **Peripheral Clock View**, and double-click the GMAC0 module. The **Clocks Diagram** shows the clock tree of GMAC module

Overview | Peripheral Clock View | Code Preview | Registers | Details

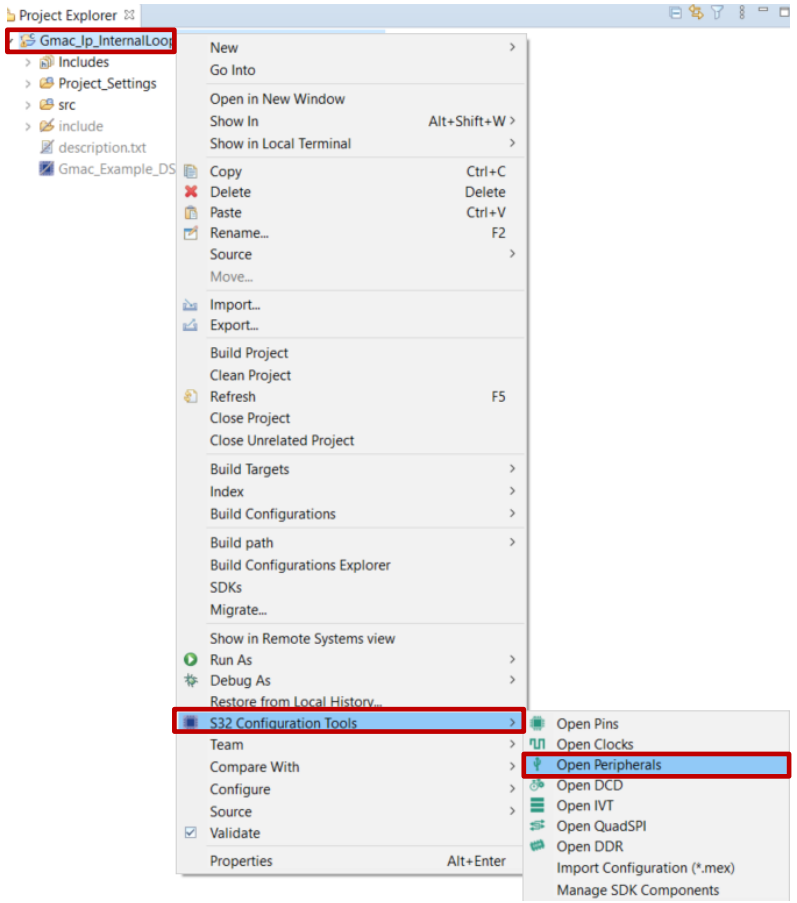
Clock Name	Enable	Source	Divider	Frequency
FLEXCAN3_CLK	☑	FIRC_CLK	/ 1	48 MHz
FRAY0_CLK	☑	FIRC_CLK	/ 2	24 MHz
FTIMER0_CLK	☑	FIRC_CLK	/ 1	48 MHz
FTIMER1_CLK	☑	FIRC_CLK	/ 1	48 MHz
GMAC0_RX_CLK	☑	GMAC_EXT_RX_REF	/ 1	25 MHz
GMAC0_TS_CLK	☑	FIRC_CLK	/ 1	48 MHz
GMAC0_TX_CLK	☑	PERIPH PHI5	/ 5	25 MHz
IIC0_CLK	☑	XBAR_DIV3_CLK	/ 1	8 MHz
IIC1_CLK	☑	XBAR_DIV3_CLK	/ 1	8 MHz
IIC2_CLK	☑	XBAR_DIV3_CLK	/ 1	8 MHz
IIC3_CLK	☑	XBAR_DIV3_CLK	/ 1	8 MHz
IIC4_CLK	☑	XBAR_DIV3_CLK	/ 1	8 MHz
LBIST0_CLK	☑	FIRC_CLK	/ 1	48 MHz
LBIST1_CLK	☑	FIRC_CLK	/ 1	48 MHz
LBIST2_CLK	☑	FIRC_CLK	/ 1	48 MHz
LBIST3_CLK	☑	FIRC_CLK	/ 1	48 MHz
LBIST4_CLK	☑	FIRC_CLK	/ 1	48 MHz
LBIST5_CLK	☑	FIRC_CLK	/ 1	48 MHz
LBIST6_CLK	☑	FIRC_CLK	/ 1	48 MHz
LBIST7_CLK	☑	FIRC_CLK	/ 1	48 MHz



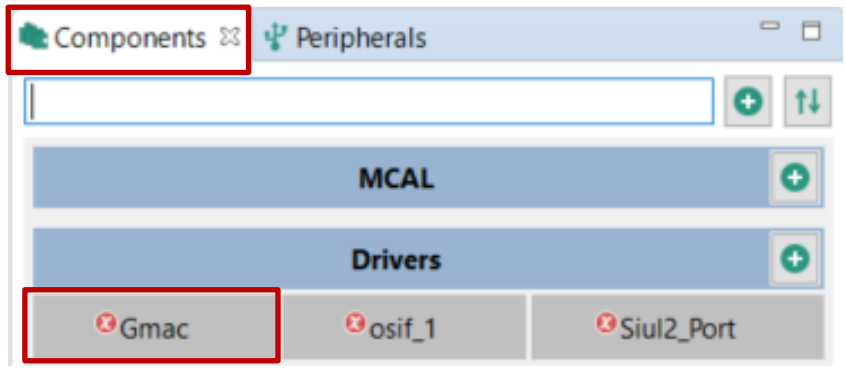
HANDS ON ETH: ETH CONFIGURATION

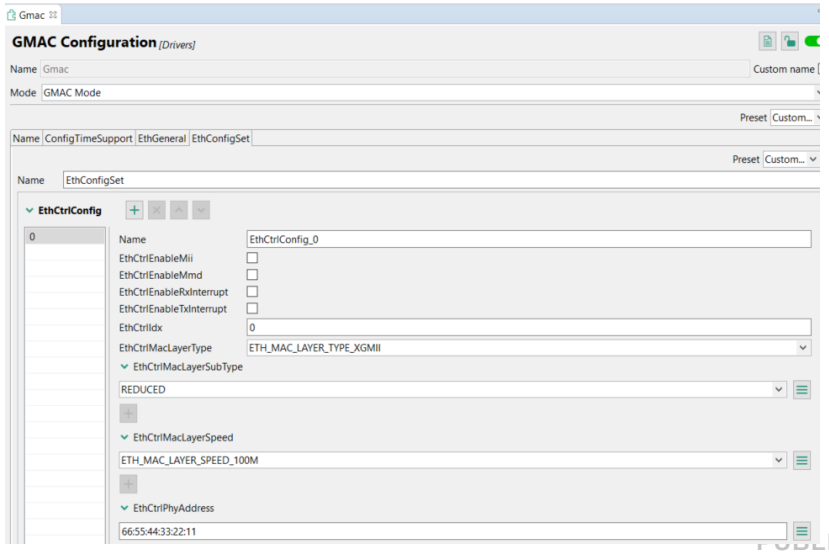
Open the peripheral configuration:

- Right-click the Project
- Select S32 Configuration Tools
- Select Open Peripherals



Select **Components** to find out **GMAC** Driver and double-click





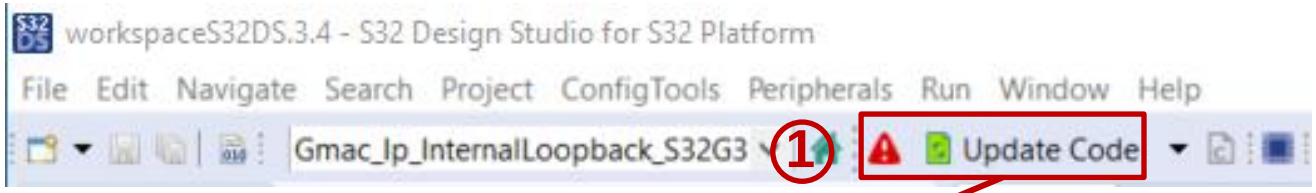
HANDS ON ETH: UPDATE CODE

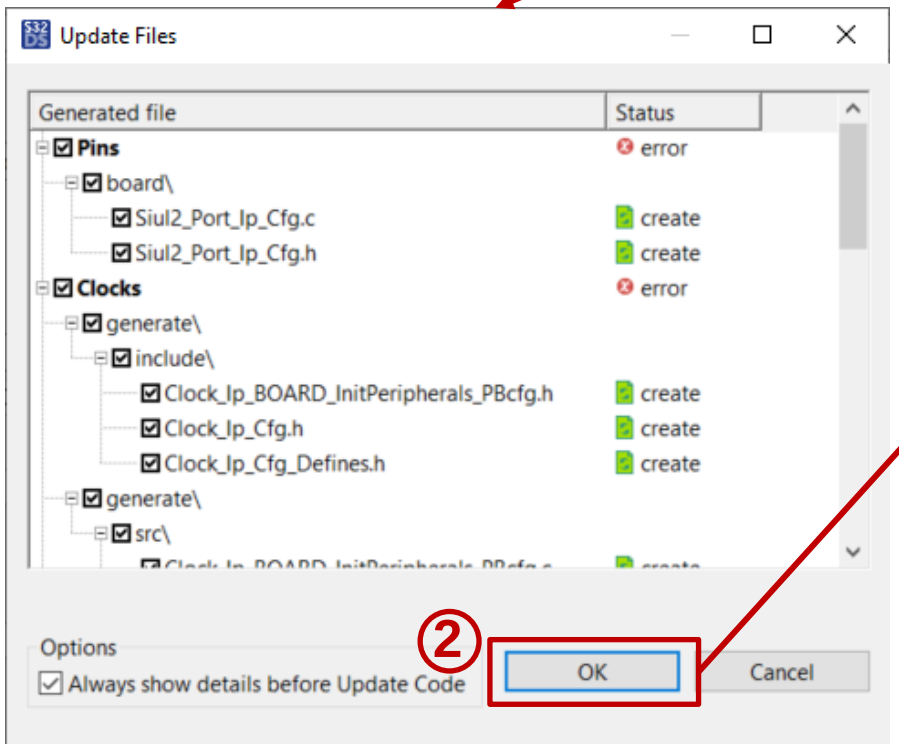
Generate code method:

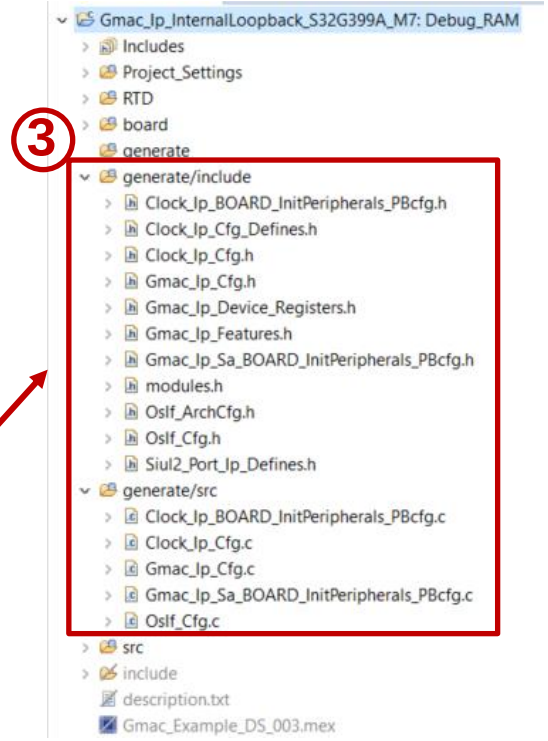
1. Open the view of any configuration tool, like Pins, then click **Update Code** (ensure desired project is selected)

2.The Update Files window pops up. It shows the detailed update information. Click **OK** button.

3.The configuration .c and .h files will be generated in "generate" folder.







HANDS ON ETH: APPLICATION CODE 1

Dissecting the main.c

```
int main(void)
{
    Gmac_Ip_TxOptionsType txOptions = {TRUE, GMAC_CRC_AND_PAD_INSERTION, GMAC_CHECKSUM_INSERTION_DISABLE};
    Gmac_Ip_BufferType txBuffer = {0};
    Gmac_Ip_BufferType rxBuffer = {0};
    Gmac_Ip_TxInfoType txInfo;
    Gmac_Ip_RxInfoType rxInfo;
    Gmac_Ip_StatusType status;
    uint8 macAddr[6U] = {0U};
    uint8 i;
    uint8 j = 0U;
    boolean result = TRUE;

    OsIf_Init(NULL_PTR);

    Siul2_Port_Ip_Init(NUM_OF_CONFIGURED_PINS0, g_pin_mux_InitConfigArr0);

    Clock_Ip_Init(&Mcu_aClockConfigPB[0]);

    Gmac_Ip_Init(INST_GMAC_0, &Gmac_0_ConfigPB_BOARD_INITPERIPHERALS);

    /* Setup the frame with the Mac address and size */
    Gmac_Ip_GetMacAddr(INST_GMAC_0, macAddr);

    /* Request a buffer of at least 64 bytes */
    txBuffer.length = 64U;
    if ((GMAC_STATUS_SUCCESS != Gmac_Ip_GetTxBuff(INST_GMAC_0, 0U, &txBuffer, NULL_PTR)) || (txBuffer.length < 64U))
    {
        result = FALSE;
    }

    for (i = 0U; i < 12U; i++)
    {
        *((uint8 *)(&txBuffer.data[0U] + i)) = macAddr[0 + j];
        if (j < 5U)
        {
            j++;
        }
        else
        {
            j = 0U;
        }
    }
```

Initialize pins to provide the external clock for GMAC

Enable GMAC controller, initialize Tx and Rx buffer via the function Gmac_Ip_Init

Apply for Txbuffer via the function Gmac_Ip_GetTxBuff and initialize transmission buffer

HANDS ON ETH: APPLICATION CODE 2

Dissecting the main.c

```
/* Payload = Frame - (DstAddr + SrcAddr + EtherType/Length + FCS) */
*((uint32 *)(txBuffer.data + 13U)) = 64U - (6U + 6U + 2U + 4U);

/* Send the ETH frame */
txBuffer.length = 64U - 4U;     /* Don't count FCS, because it is automatically inserted by the controller in this example */
if (GMAC_STATUS_SUCCESS != Gmac_Ip_SendFrame(INST_GMAC_0, 0U, &txBuffer, &txOptions))
{
    result = FALSE;
}
```

Transmit frame via Gmac_Ip_SendFrame

```
/* Wait for the frame to be transmitted */
do {
    status = Gmac_Ip_GetTransmitStatus(INST_GMAC_0, 0U, &txBuffer, &txInfo);
} while (status == GMAC_STATUS_BUSY);

/* Check the frame status */
if ((GMAC_STATUS_SUCCESS != status) || (0U != txInfo.errMask))
{
    result = FALSE;
}

/* Wait for the frame to be received */
do {
    status = Gmac_Ip_ReadFrame(INST_GMAC_0, 0U, &rxBuffer, &rxInfo);
} while (status == GMAC_STATUS_RX_QUEUE_EMPTY);

/* Check the frame status */
if ((GMAC_STATUS_SUCCESS != status) || (0U != rxInfo.errMask))
{
    result = FALSE;
}
```

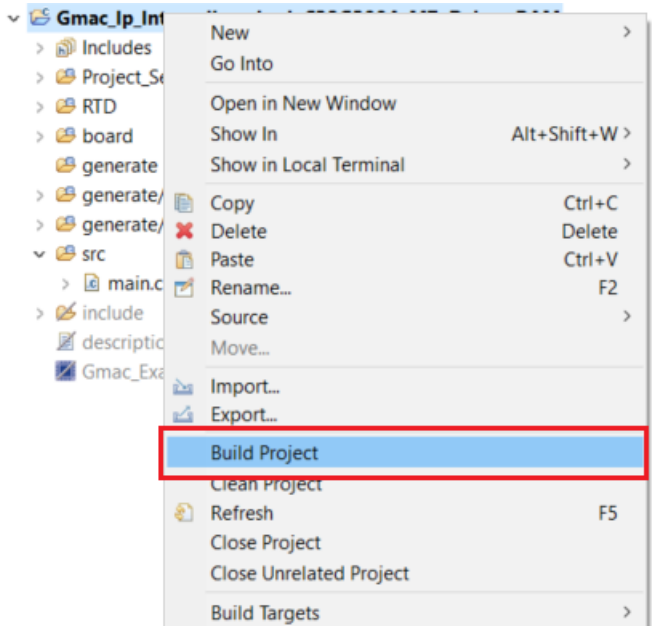
Verify frame is transmitted or received

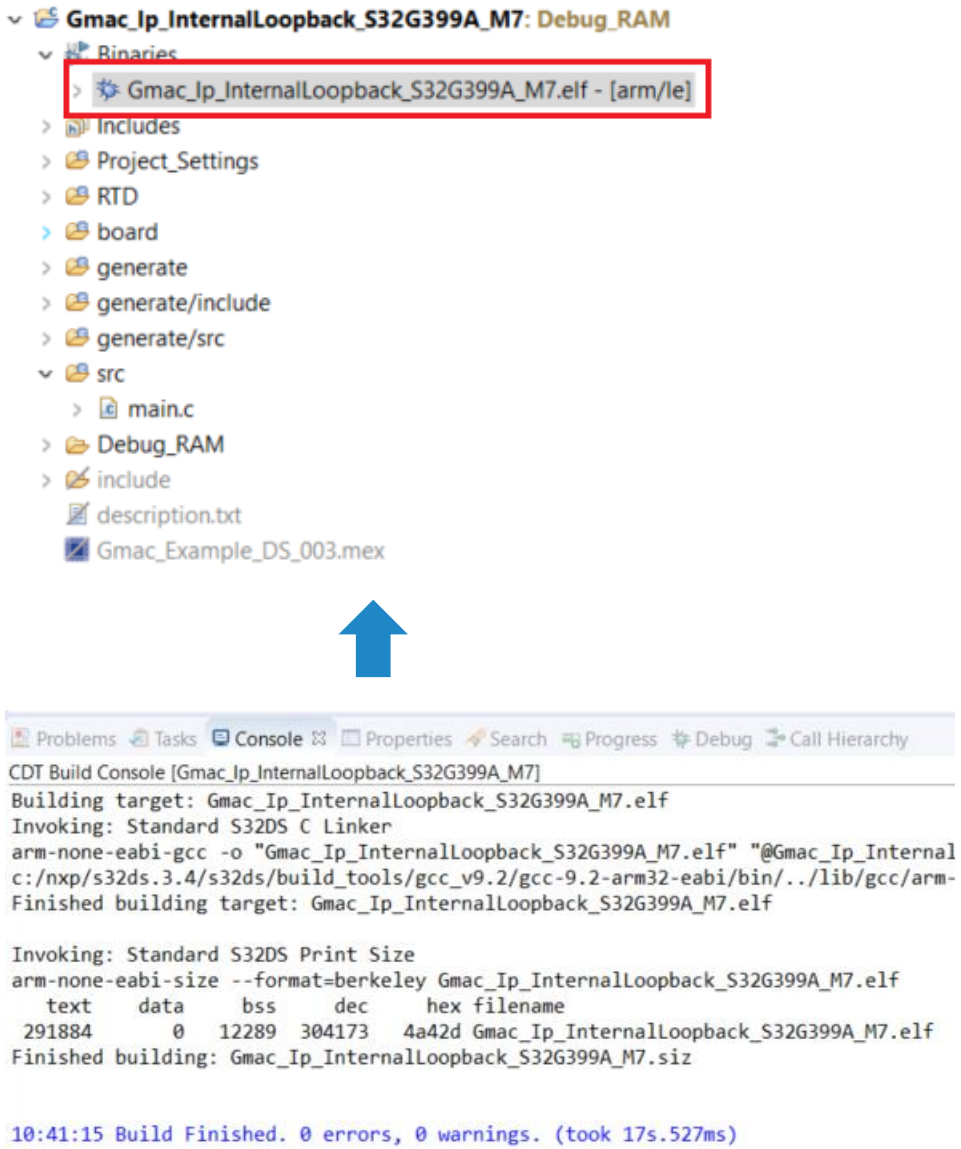
```
Gmac_Ip_ProvideRxBuff(INST_GMAC_0, 0U, &rxBuffer);

Gmac_Ip_DisableController(INST_GMAC_0);
```

HANDS ON ETH: BUILD AND DEBUG 1

Build target Project:

- Right-click the Project
- Build Project
- The console print build information
- Gmac_Ip_InternalLoopback_S32G399A_M7.elf is created

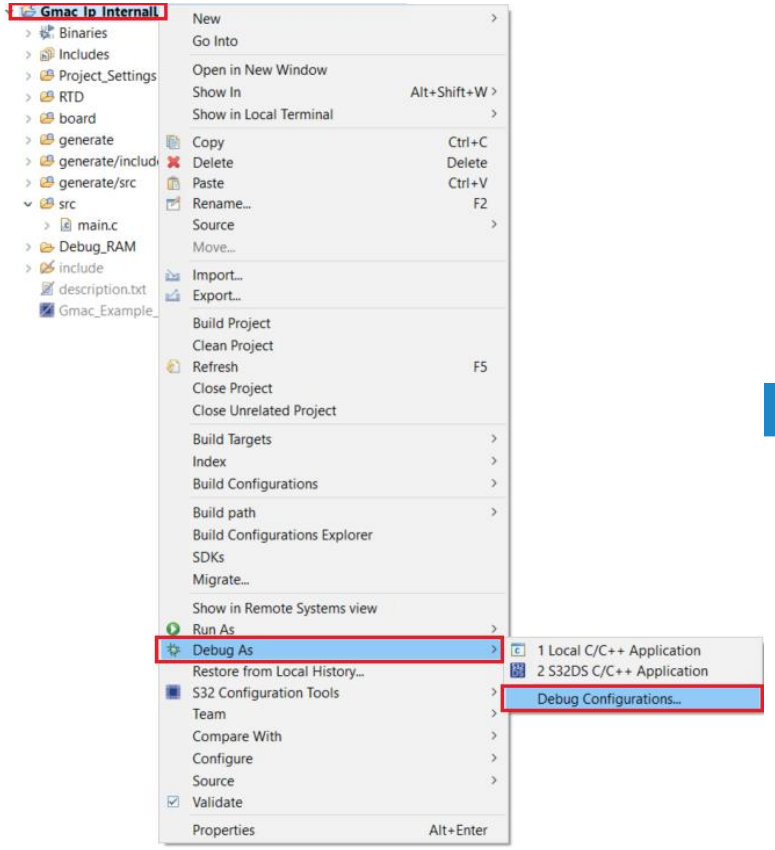




HANDS ON ETH: BUILD AND DEBUG 2

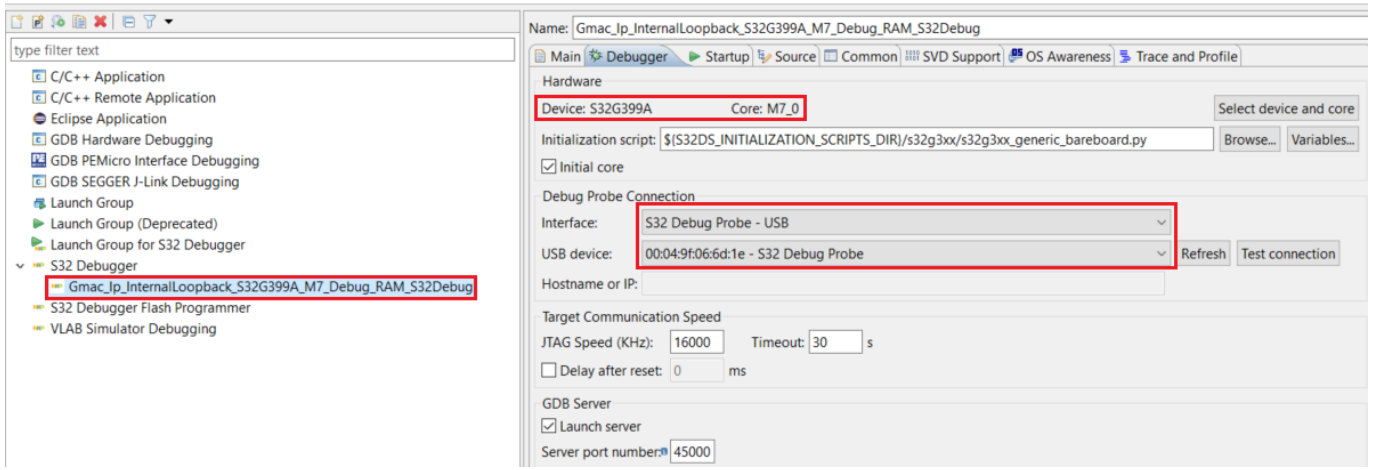
Go to debug configuration:

- Right-click the Project
- Select the Debug As
- Click on Debug Configurations



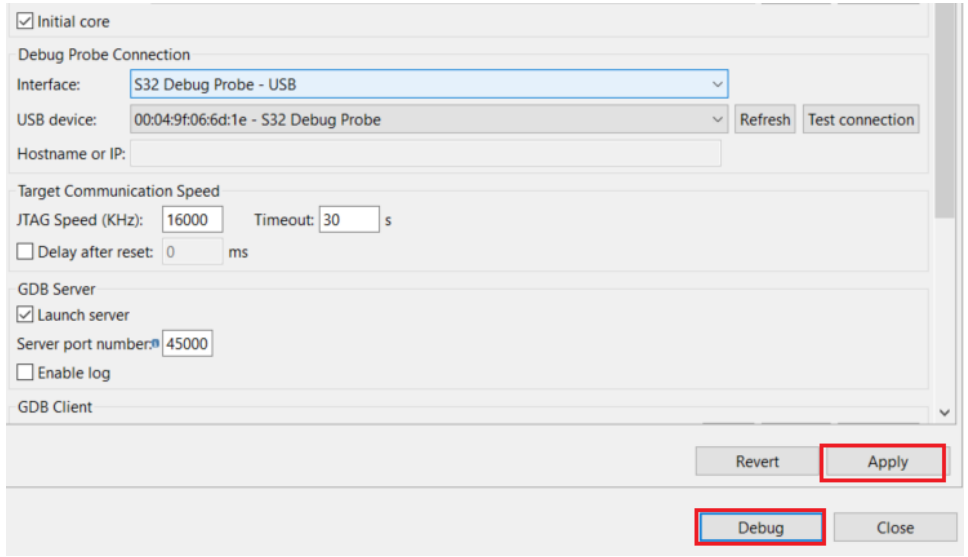
Debug configuration setting:

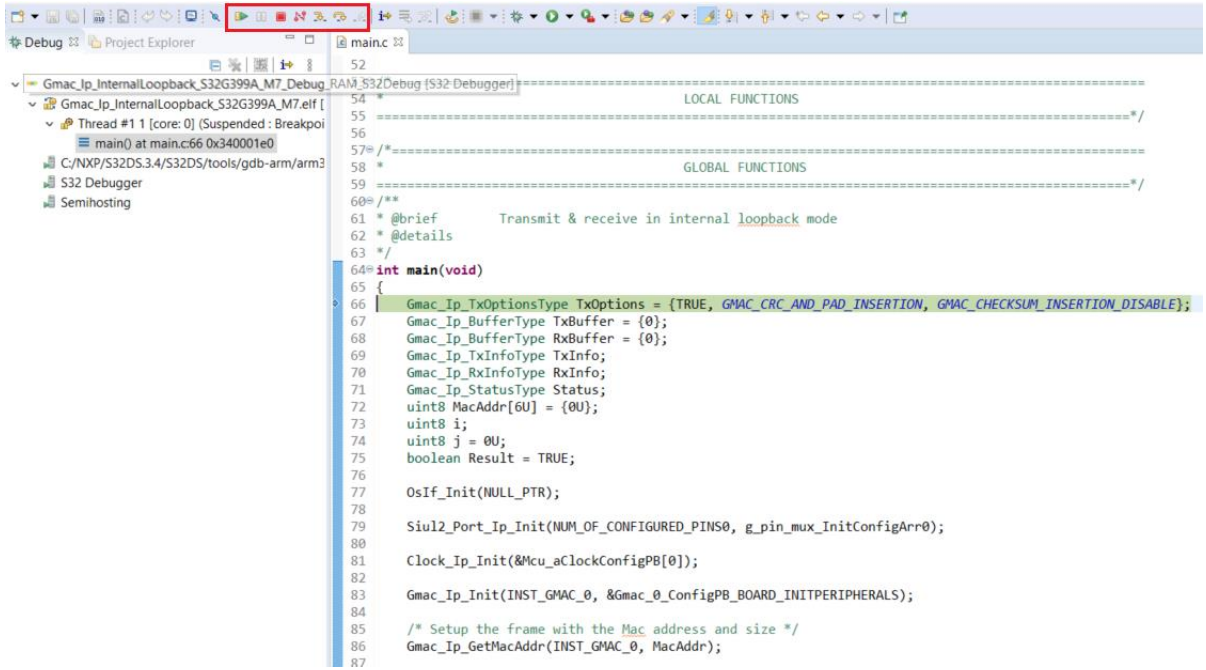
- Connect the S32 Debug probe with PC and RDB3
- Click on target project
- Select the target device and core as S32G399A_M7_0
- Select target S32 Debug Probe



HANDS ON ETH: DEBUG AND RUN

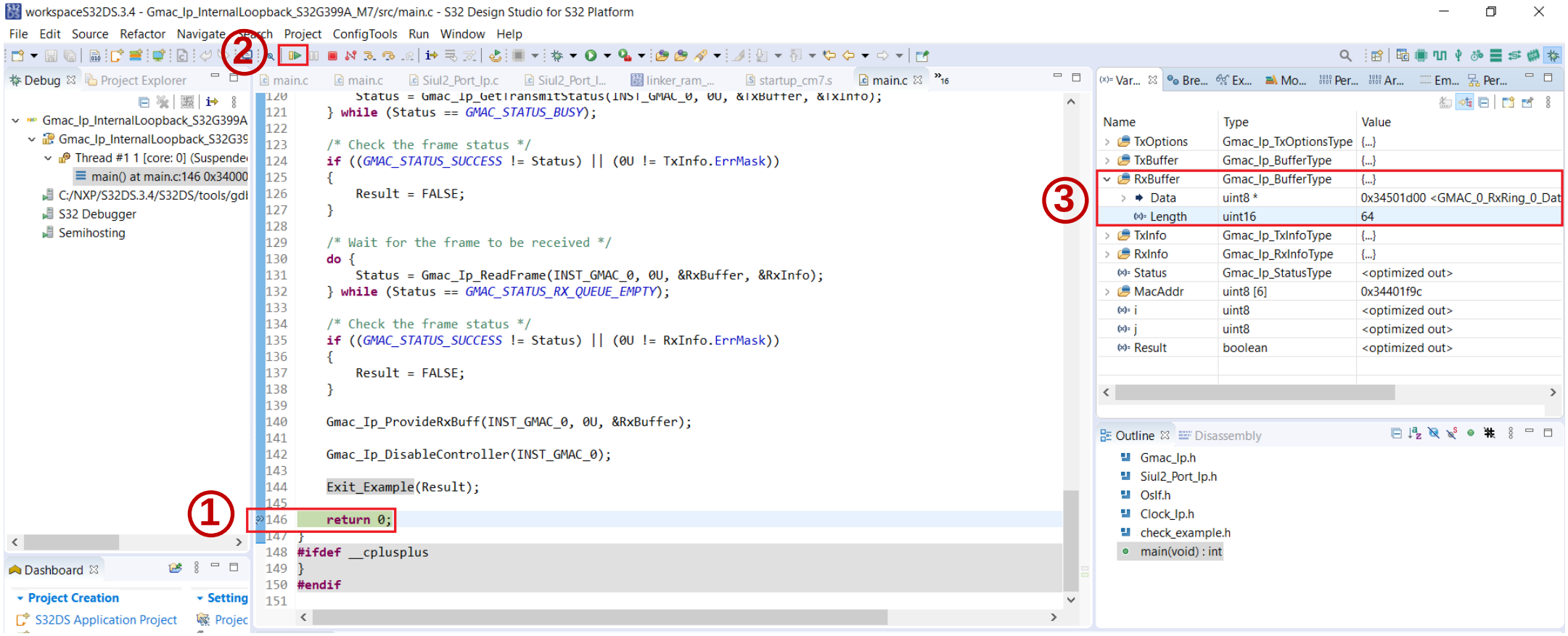
Power on the RDB3, click on "Apply", then click on "Debug". The view will switch to the Debug Perspective, and you can use the controls to control the program flow.





HANDS ON ETH: TEST RESULT 1

In this project, the Ethernet module works in internal loopback mode. Add a breakpoint to the last line of the main function, then click on the Resume option. The received ETH frame can be watched from rxBuffer.



HANDS ON ETH: TEST RESULT 2

The received frame can be watched in rxBuffer.

Variables | Breakpoints | Expressions | Modules | Peripheral Registers | Arm System Registers | EmbSys Registers | Peripherals

Name	Type	Value
TxOptions	Gmac_Ip_TxOptionsType	{...}
TxBuffer	Gmac_Ip_BufferType	{...}
RxBuffer	Gmac_Ip_BufferType	{...}
Data	uint8 *	0x34501d00 <GMAC_0_RxRing_0_DataBuffer> "fUD3\"\021fUD3\"\021"
Length	uint16	64
TxInfo	Gmac_Ip_TxInfoType	{...}

Console | Registers | Problems | Executables | Debug Shell | Memory Spaces | Watch registers | Debugger Console | Memory

Monitors

0x34501d00

0x34501d00 : 0x34501D00 <Hex> | New Renderings...

Address	0 - 3	4 - 7	8 - B	C - F
34501D00	66554433	22116655	44332211	002E0000
34501D10	00000000	00000000	00000000	00000000
34501D20	00000000	00000000	00000000	00000000
34501D30	00000000	00000000	00000000	65EA1543
34501D40	00000000	00000000	00000000	00000000
34501D50	00000000	00000000	00000000	00000000
34501D60	00000000	00000000	00000000	00000000
34501D70	00000000	00000000	00000000	00000000
34501D80	00000000	00000000	00000000	00000000

Hands on CAN Example

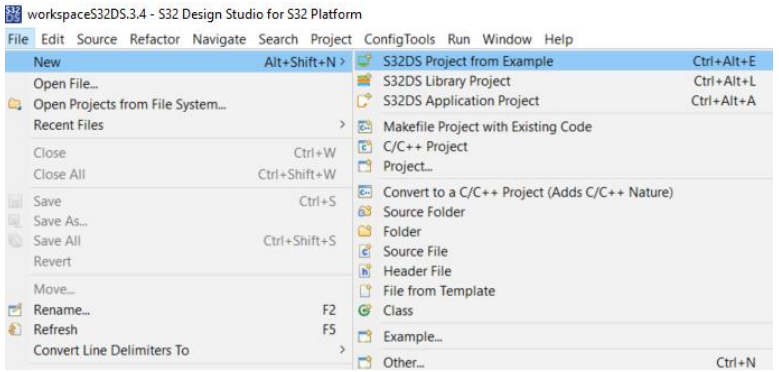
SECURE CONNECTIONS
FOR A SMARTER WORLD

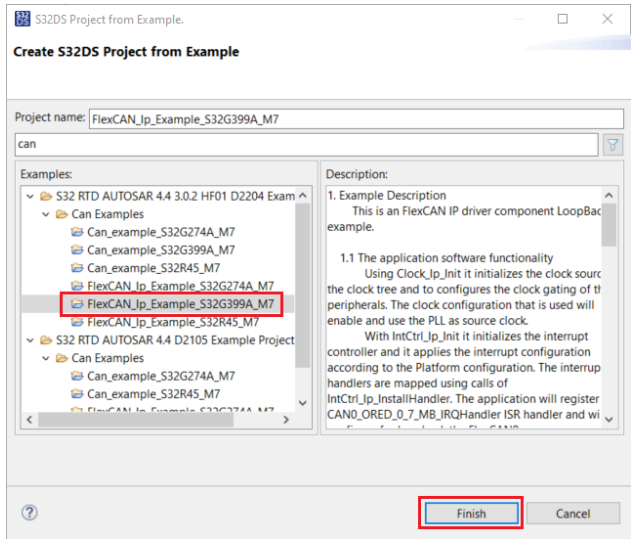
HANDS ON CAN – OBJECTIVE

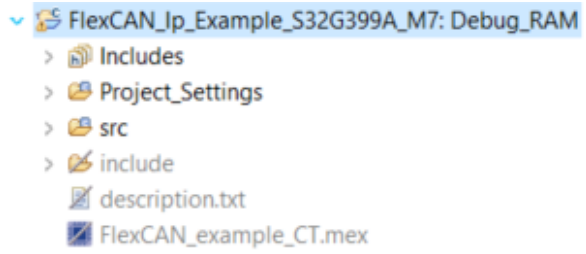
- How to import the CAN example into S32DS
- How to configure the clock of CAN via S32DS
- How to configure the port of CAN via S32DS
- How to modify the CAN loopback
- How to debug the CAN example with S32 debug probe

HANDS ON CAN : IMPORT CAN EXAMPLE PROJECT

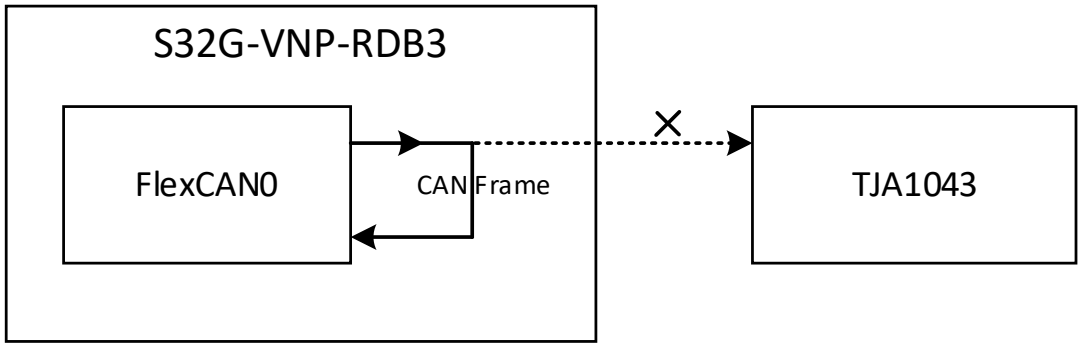
Open S32 Design Studio, go to "File -> New -> S32DS Project From Example". Select "**FlexCAN_Ip_Example_S32G399A_M7**" example, then click on “Finish”. The project is copied into current workspace.





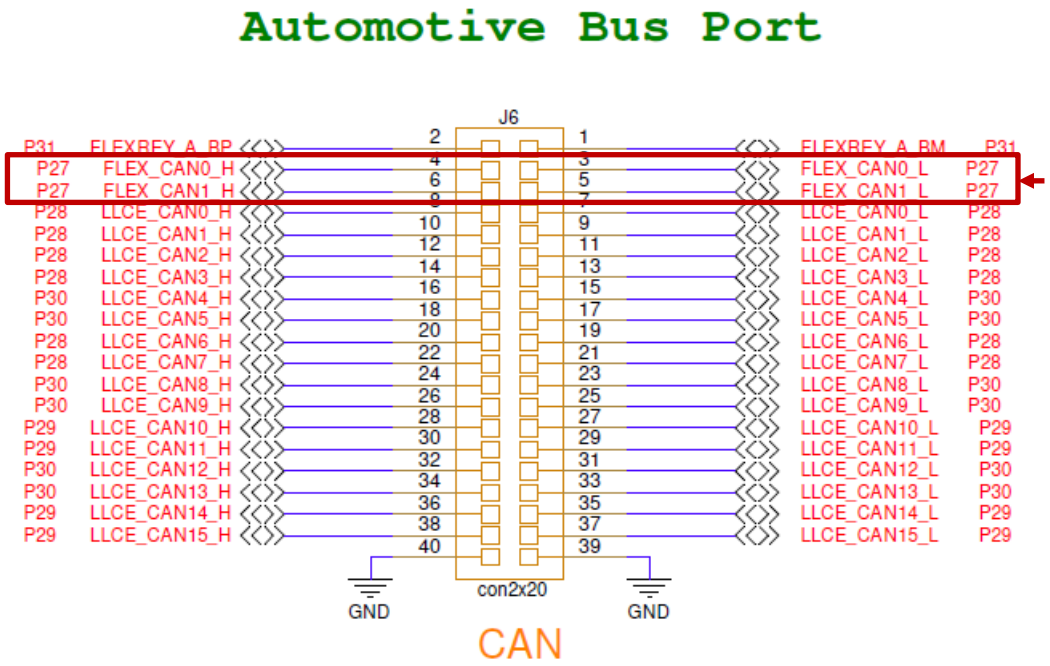


By default, "FlexCAN_Ip_Example_S32G399A_M7" project only shows the LoopBack function of FlexCAN.

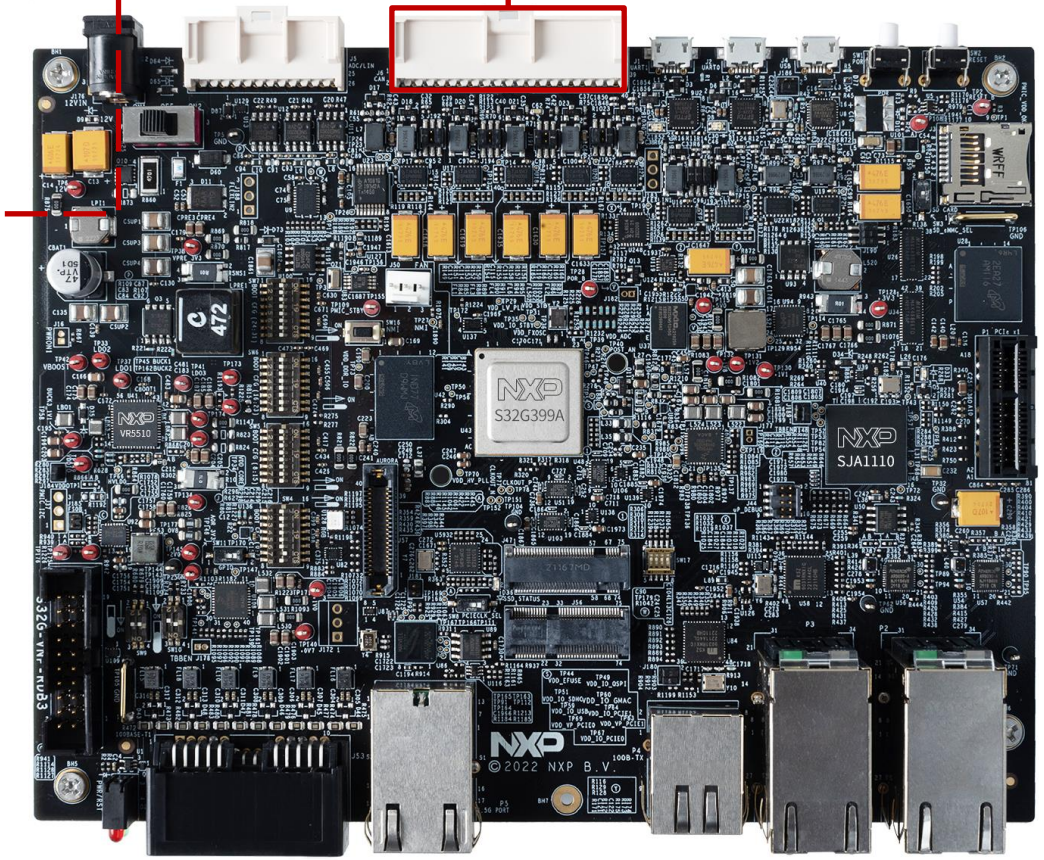


HANDS ON CAN: MODIFY THE EXAMPLE

The "FlexCAN_Ip_Example_S32G399A_M7" project only supports loopback mode. This guide will demonstrate how to modify the default configuration to transmit CAN frame from FlexCAN_0 to FlexCAN_1



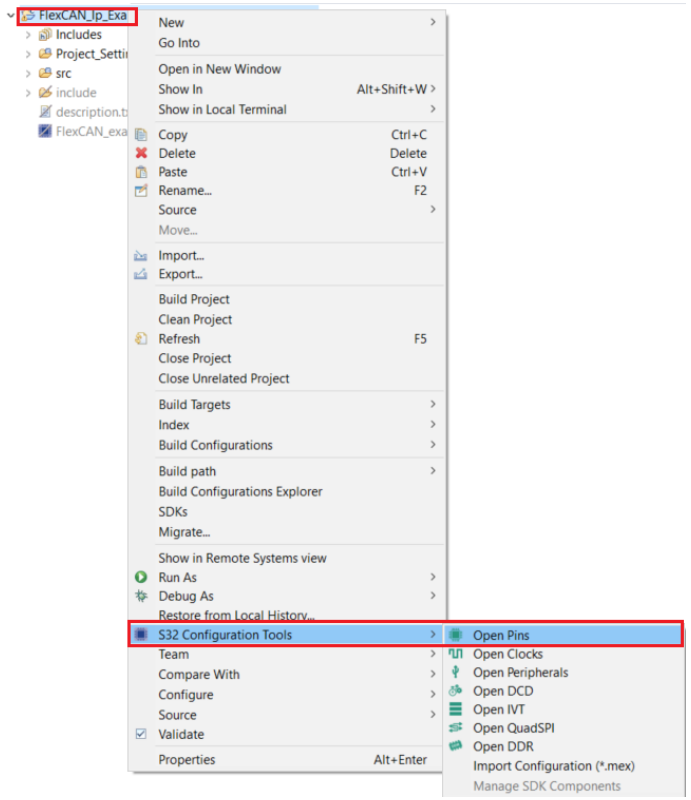
FlexCAN_0 connect to FlexCAN_1 via physical wiring

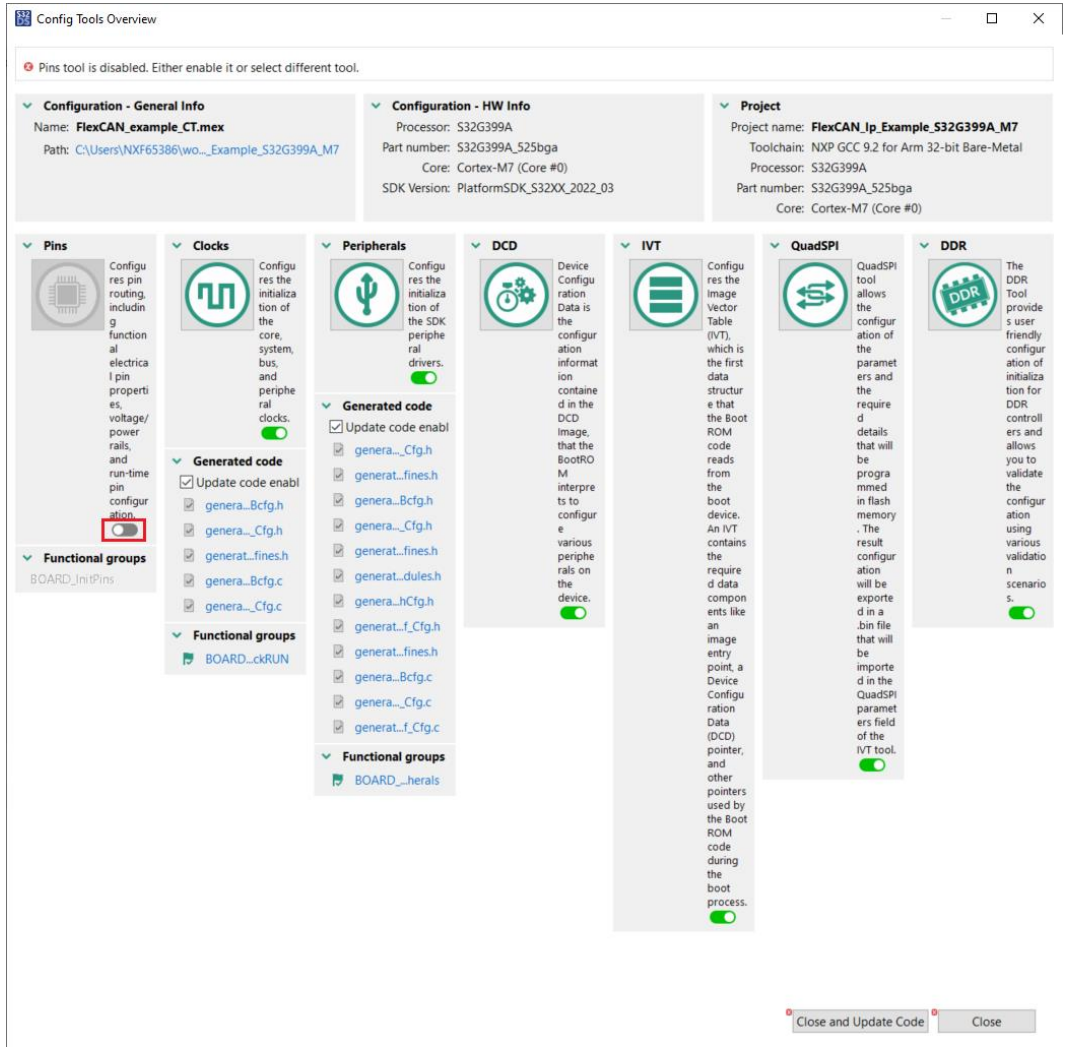


HANDS ON CAN: PORT CONFIGURATION 1

Go to Pin configuration view:

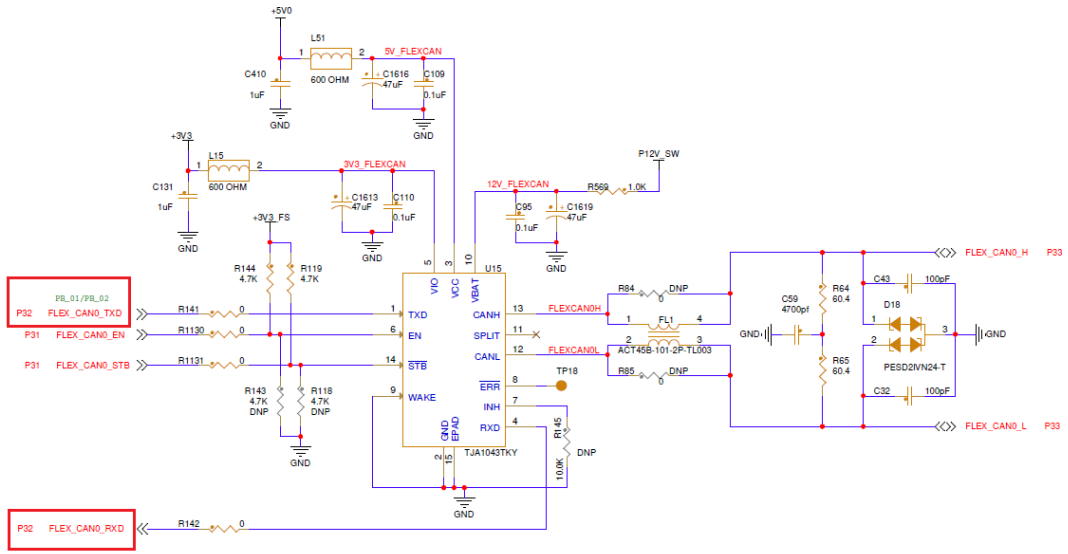
- Right-click the Project
- Select S32 Configuration Tools
- Select Open Pins
- Enable Pins

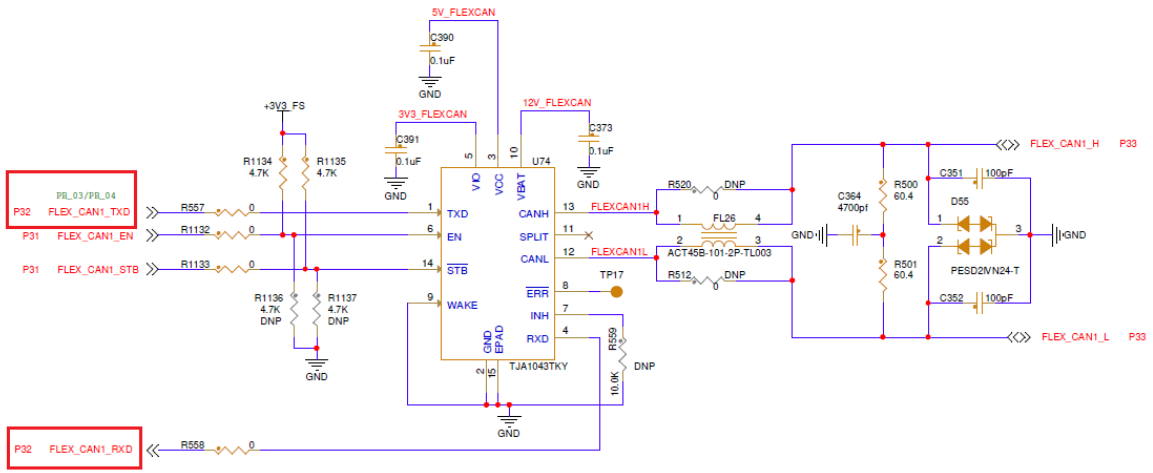




HANDS ON CAN: PORT CONFIGURATION 2

Add the Pins as the schematic of FlexCAN0 and FlexCAN1





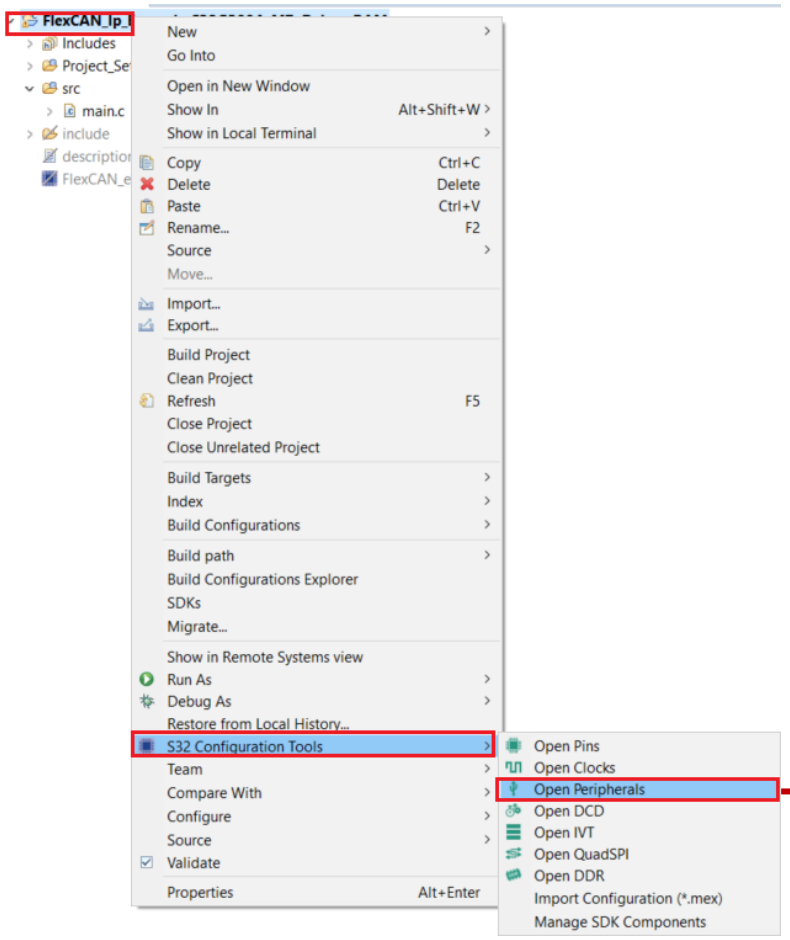
Routed Pins for BOARD... 4

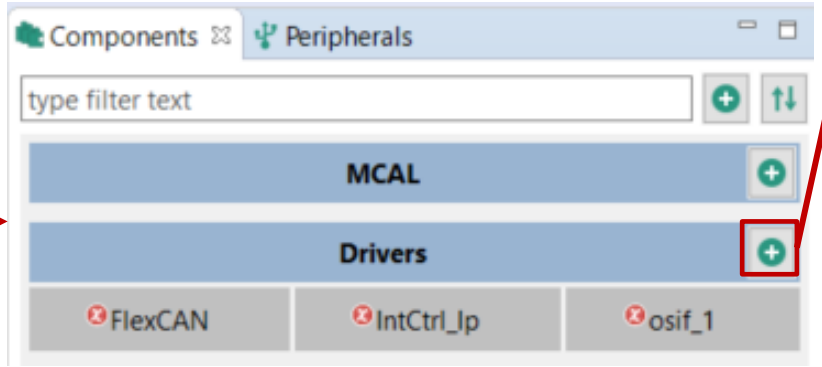
#	Peripheral	Signal	Route to	Label	Identifier	Power group	Direction	Output Buffer	Open Drain	Input Buffer
D7	CAN_0	rxd	PB_02		*n/a*	*VDD_IO_B (0V)*	*Input*	*Disabled*	*Disabled*	*Enabled*
E7	CAN_0	txd	PB_01		*n/a*	*VDD_IO_B (0V)*	*Output*	*Enabled*	*Disabled*	*Disabled*
E8	CAN_1	rxd	PB_04		*n/a*	*VDD_IO_B (0V)*	*Input*	*Disabled*	*Disabled*	*Enabled*
C6	CAN_1	txd	PB_03		*n/a*	*VDD_IO_B (0V)*	*Output*	*Enabled*	*Disabled*	*Disabled*

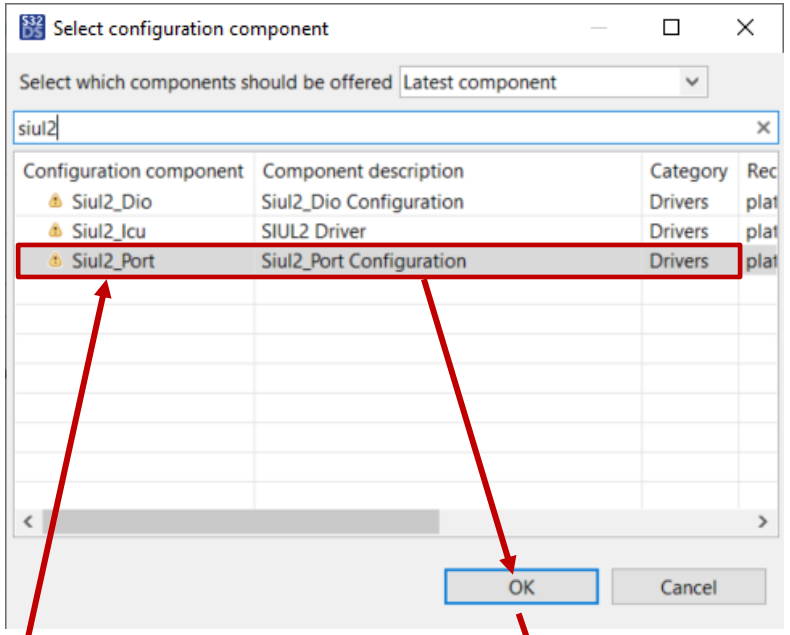
HANDS ON CAN: PORT CONFIGURATION 3

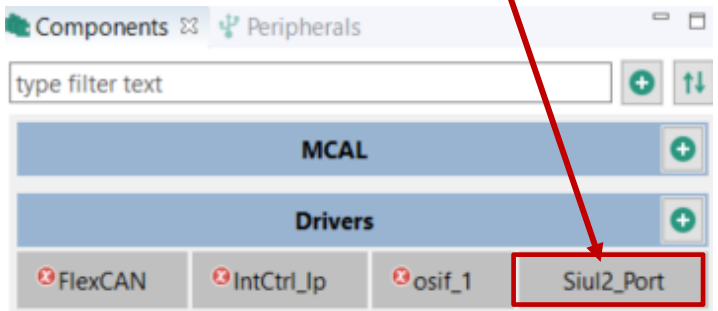
Add the Port configuration:

- Right-click the Project,
- Select S32 Configuration Tools
- Select Open Peripherals
- Click on the plus button
- Click on the Siul2_Port component
- Click on OK
- The Siul2_Port driver will be added





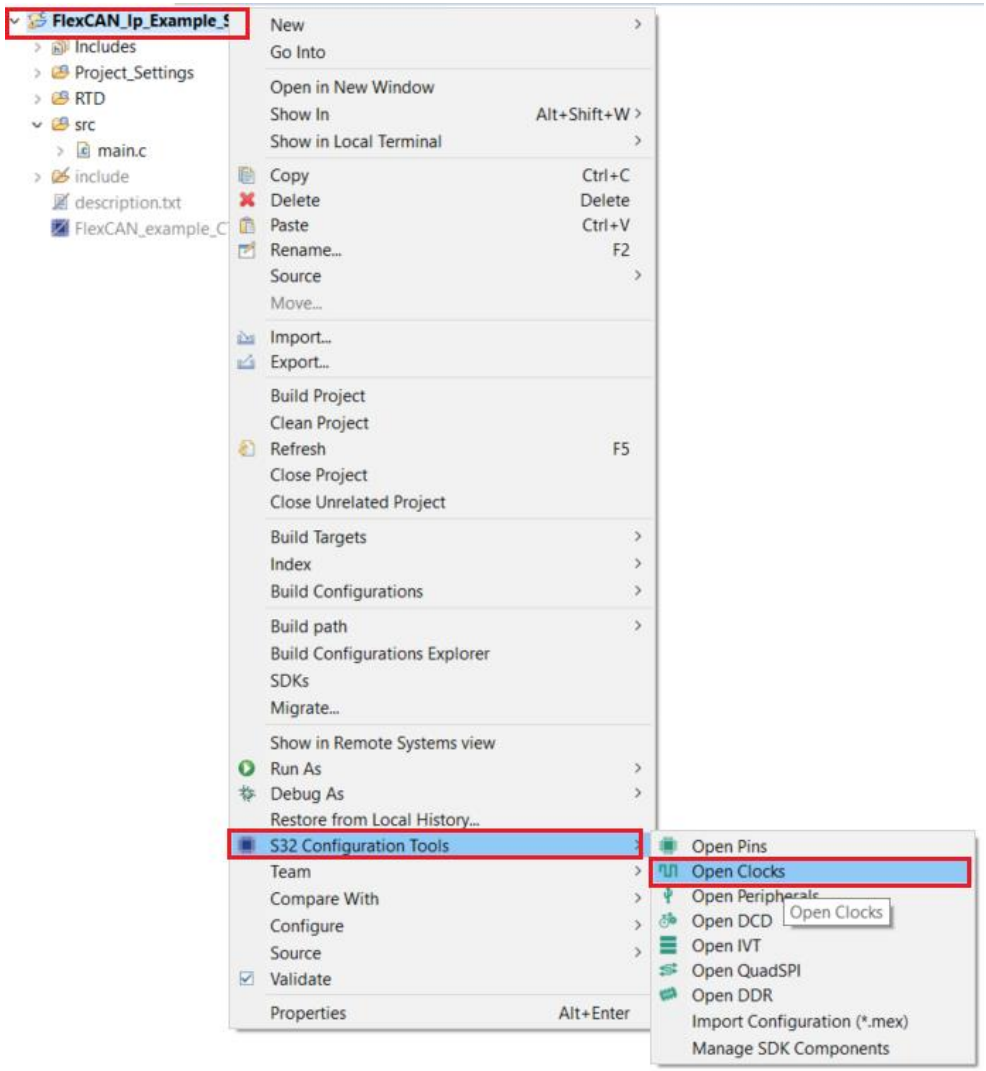




HANDS ON CAN: CLOCK CONFIGURATION 1

Switch to clocks configuration view:

- Right-click the Project,
- Select S32 Configuration Tools
- Select Open Clocks

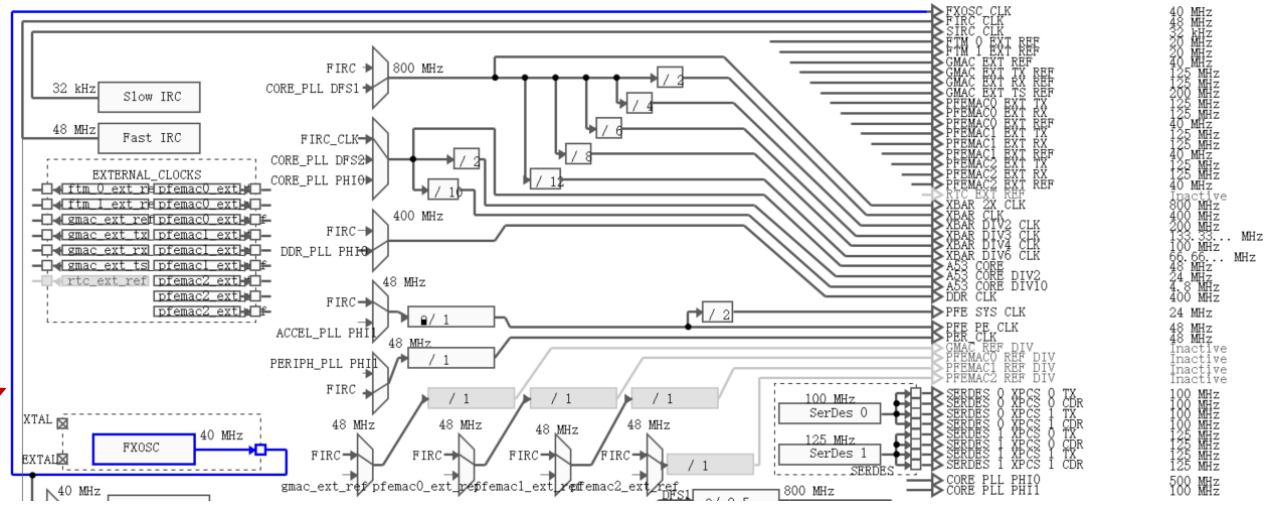


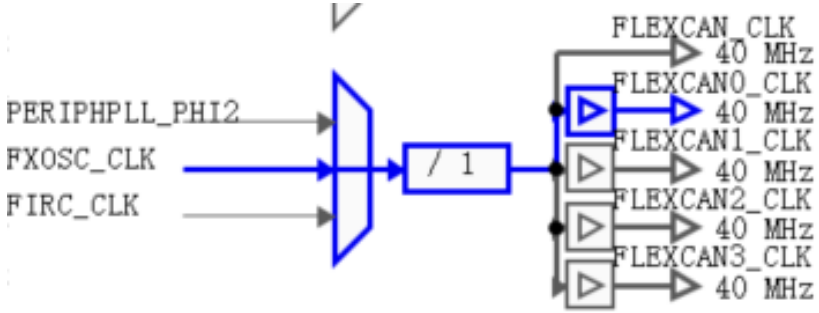
HANDS ON CAN: CLOCK CONFIGURATION 2

Open the **Peripheral Clock View**, double-click the **FLEXCAN0_CLK**. The **Clocks Diagram** will show the clock tree and the key node can be re-set. The default clock configuration of FlexCAN is 48 MHZ. Switch the clock which source from FXOSC(**40** MHZ).

Overview | Peripheral Clock View | Code Preview | Registers | Details

Clock Name	Enable	Source	Divider	Frequency
DMAMUX0_CLK	☑	XBAR_DIV3_CLK	/ 1	133.33... ...
DMAMUX1_CLK	☑	XBAR_DIV3_CLK	/ 1	133.33... ...
DMAMUX2_CLK	☑	XBAR_DIV3_CLK	/ 1	133.33... ...
DMAMUX3_CLK	☑	XBAR_DIV3_CLK	/ 1	133.33... ...
DMA_CRC0_CLK	☑	XBAR_CLK	/ 1	400 MHz
DMA_CRC1_CLK	☑	XBAR_CLK	/ 1	400 MHz
EIM0_CLK	☑	A53_CORE_DIV10_C...	/ 1	4.8 MHz
EIM1_CLK	☑	XBAR_DIV6_CLK	/ 1	66.66... ...
EIM2_CLK	☑	XBAR_DIV6_CLK	/ 1	66.66... ...
EIM3_CLK	☑	XBAR_DIV6_CLK	/ 1	66.66... ...
EIM_CLK	☑	XBAR_DIV6_CLK	/ 1	66.66... ...
ENET_LOOPBACK_CLK	☑	FIRC_CLK	/ 1	48 MHz
ERM0_CLK	☑	XBAR_DIV6_CLK	/ 1	66.66... ...
FLEXCAN0_CLK	☑	FXOSC_CLK	/ 1	40 MHz
FLEXCAN1_CLK	☑	FXOSC_CLK	/ 1	40 MHz
FLEXCAN2_CLK	☑	FXOSC_CLK	/ 1	40 MHz
FLEXCAN3_CLK	☑	FXOSC_CLK	/ 1	40 MHz
FRAY0_CLK	☑	FIRC_CLK	/ 2	24 MHz
FTIMER0_CLK	☑	FIRC_CLK	/ 1	48 MHz
FTIMER1_CLK	☑	FIRC_CLK	/ 1	48 MHz
GMAC0_RX_CLK	☑	FIRC_CLK	/ 1	48 MHz
GMAC0_TS_CLK	☑	FIRC_CLK	/ 1	48 MHz
GMAC0_TX_CLK	☑	FIRC_CLK	/ 1	48 MHz

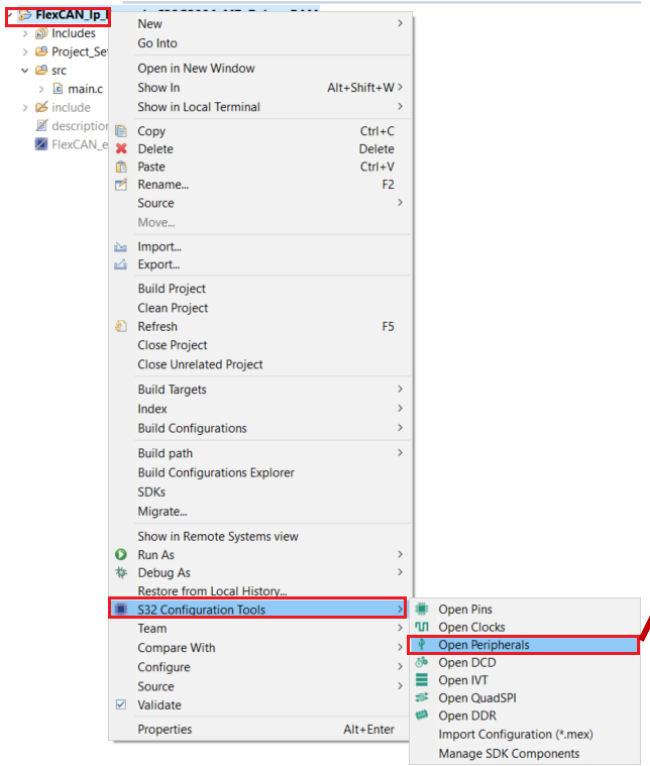


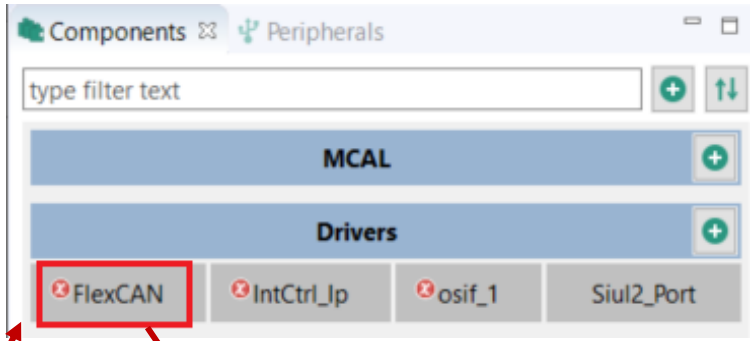


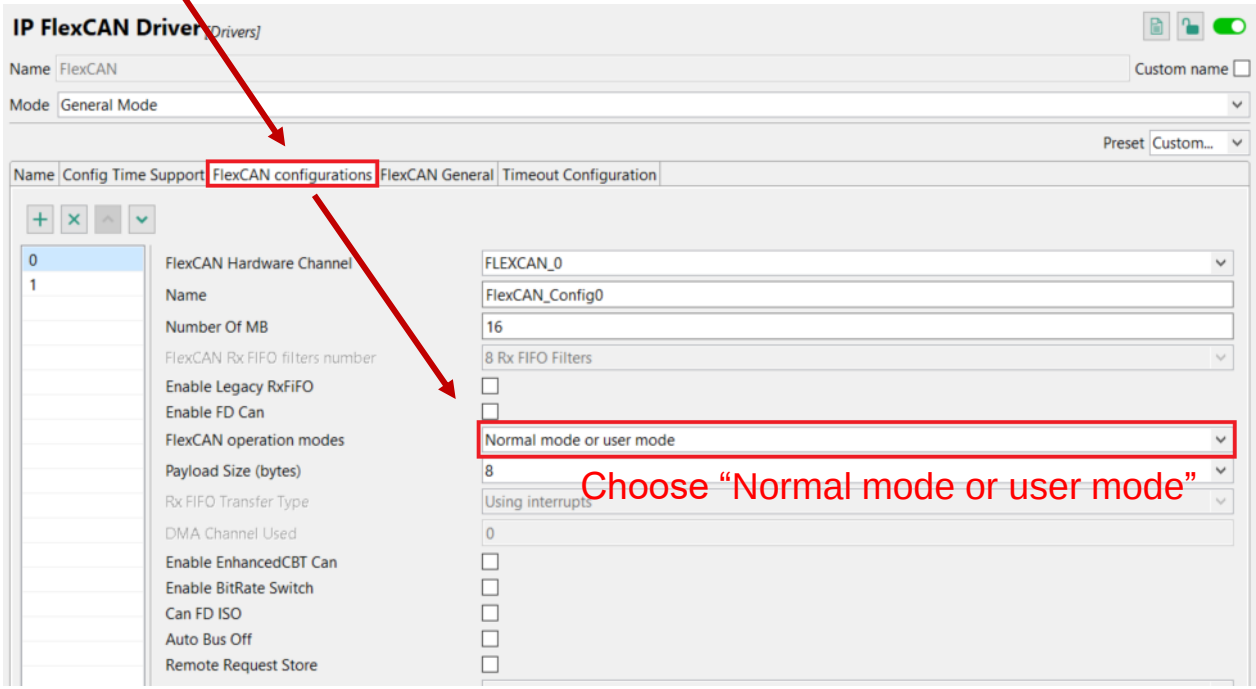
HANDS ON CAN: CAN CONFIGURATION 1

Mode setting for FlexCAN0:

- Right-click the Project,
- Select S32 Configuration Tools
- Select Open Peripherals
- Double-click FlexCAN component
- Set mode for FlexCAN0



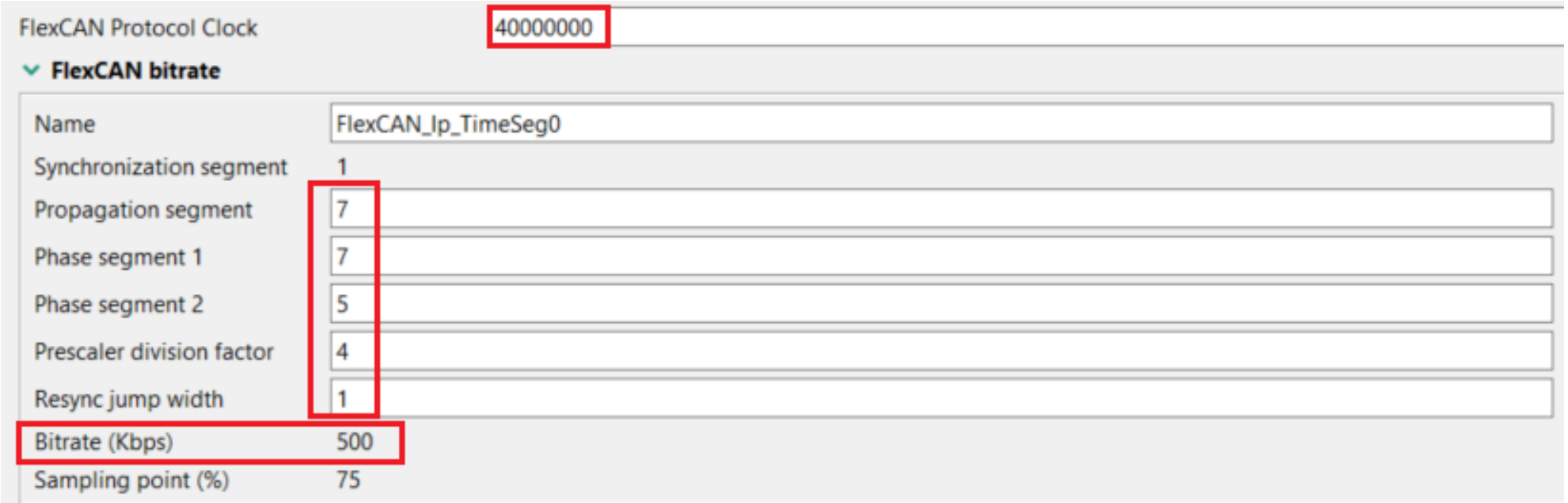




HANDS ON CAN: CAN CONFIGURATION 2

Configure the BaudRate as 500Kbps for FlexCAN0

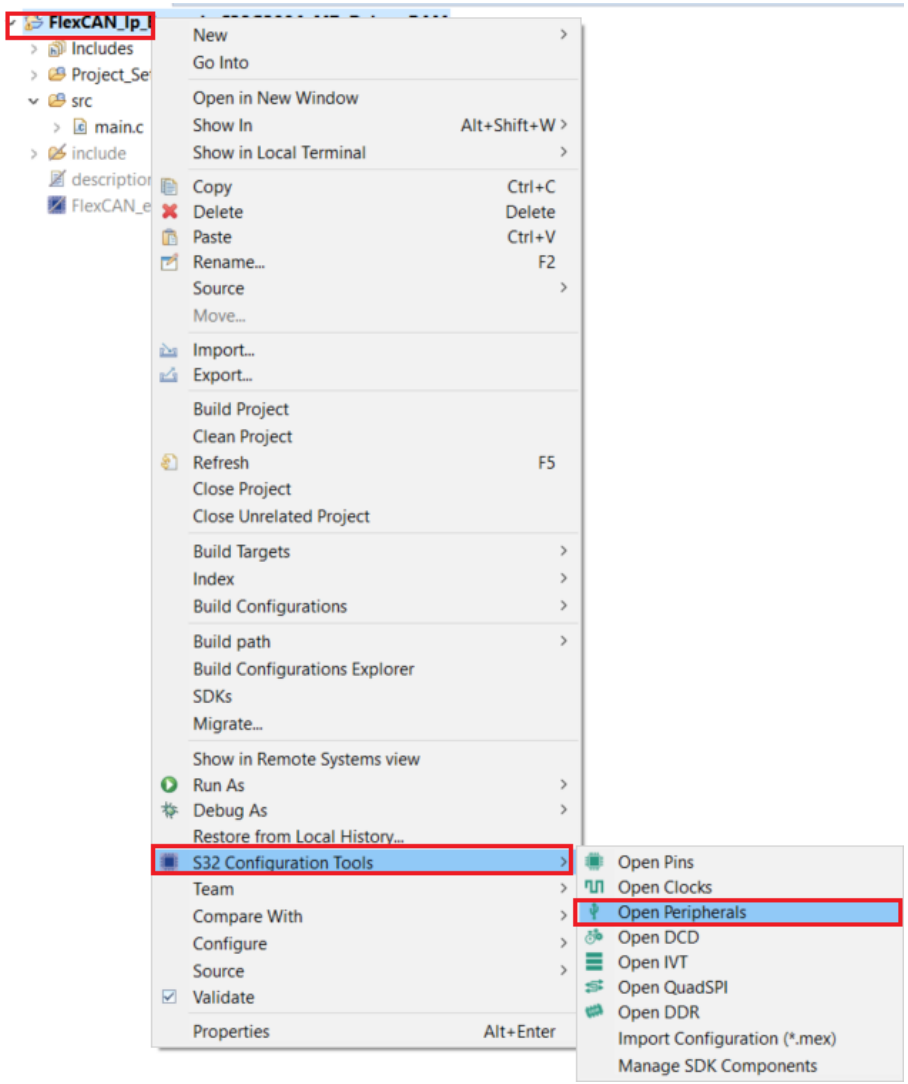
- TimeQuantum (seconds) = Prescaler / CanClockFrequency
- No. of CanTimeQuantas = (1 / CancontrollerBaudRate) / TimeQuantum
- No. of CanTimeQuantas = 1 + CanControllerPropSeg + CanControllerSeg1 + CanControllerSeg2



HANDS ON CAN: CAN CONFIGURATION 3

Add FlexCAN1:

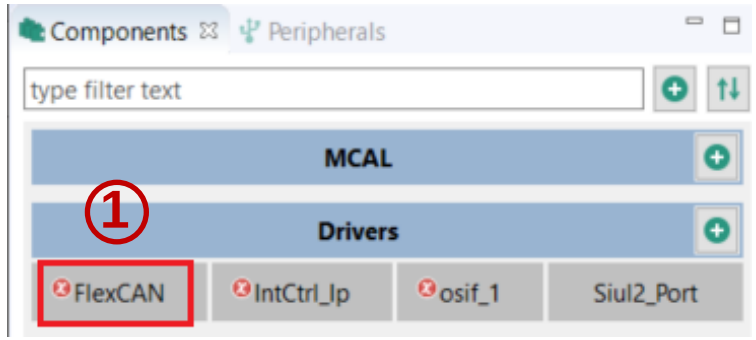
- Right-click the Project,
- Select S32 Configuration Tools
- Select Open Peripherals

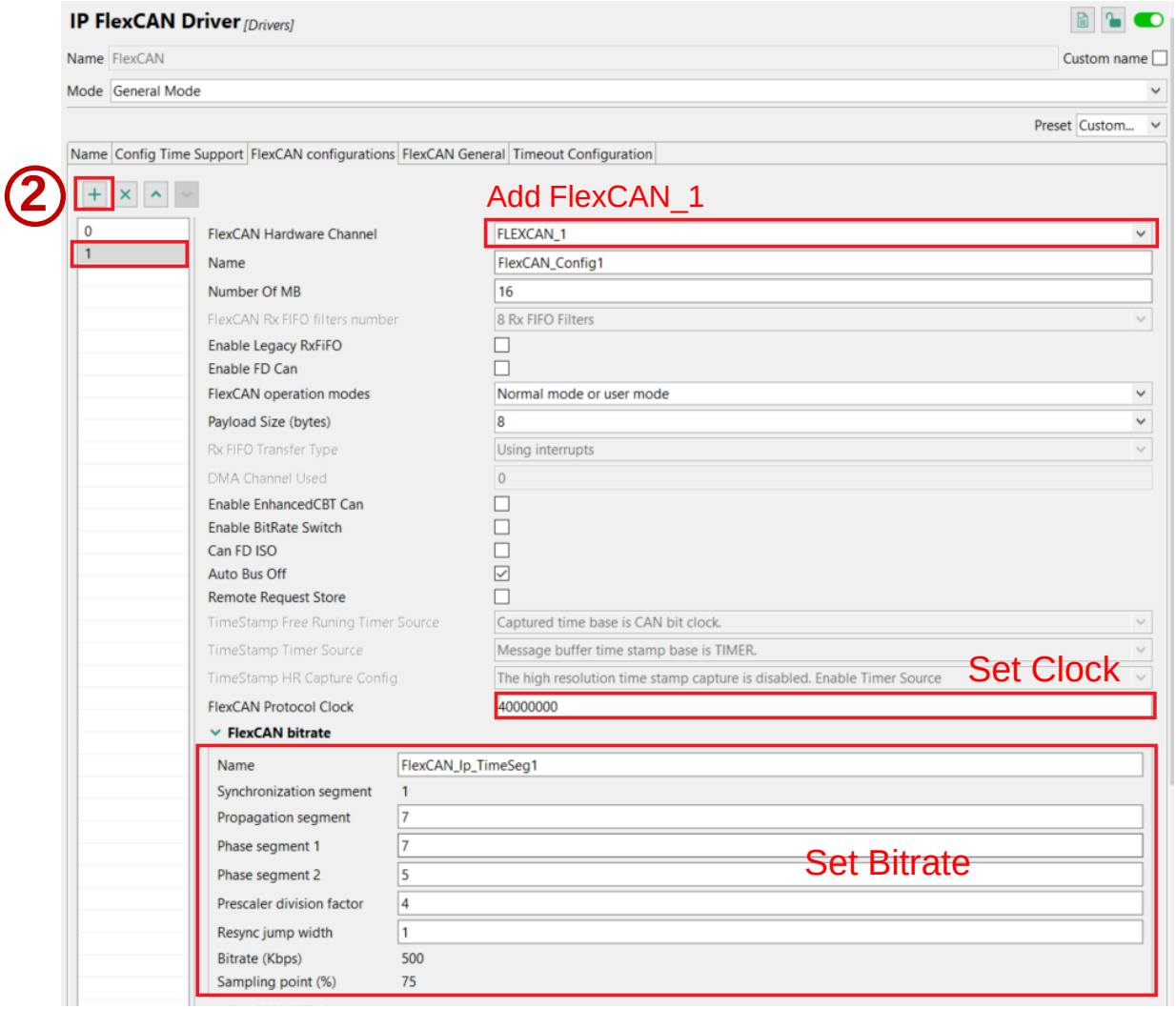


HANDS ON CAN: CAN CONFIGURATION 4

Add FlexCAN1:

- Double-click FlexCAN component
- Click on plus button to add FlexCAN_1
- Set Clock as 40MHz for FlexCAN_1
- Set Bitrate as 500kbps





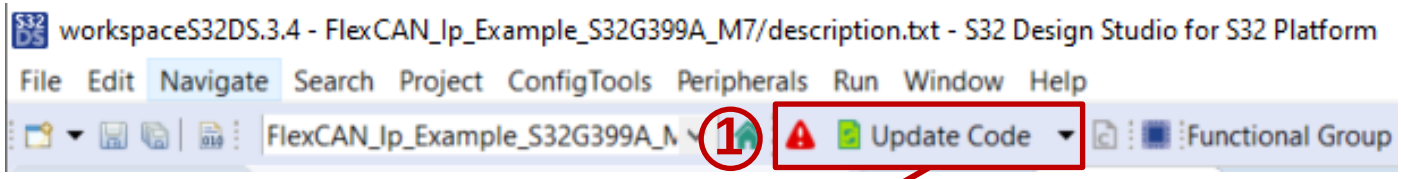
HANDS ON CAN: UPDATE CODE

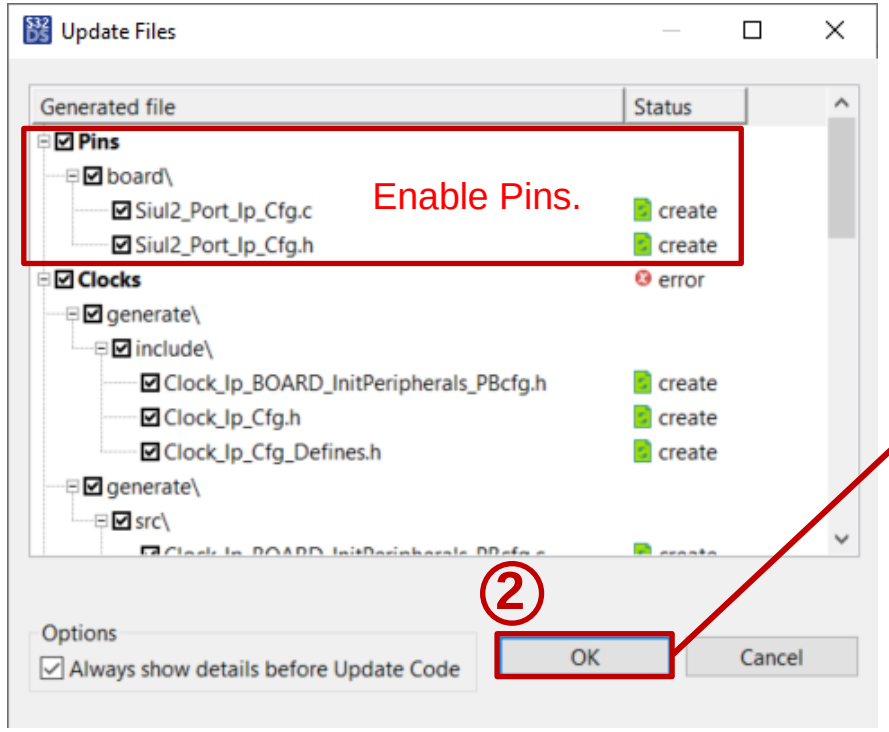
Generate code method:

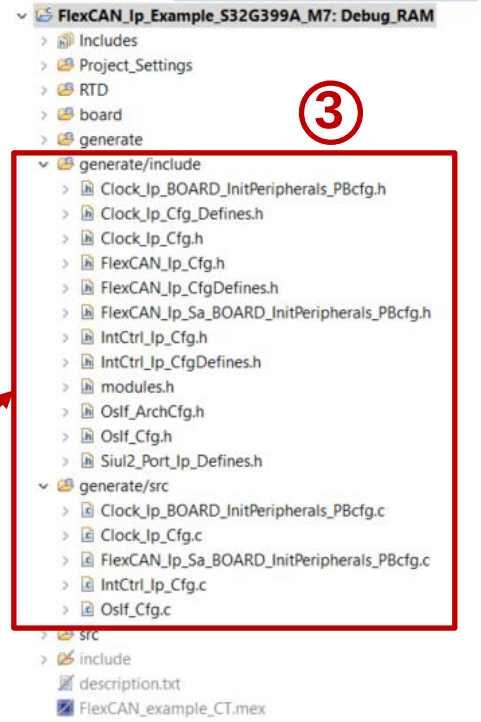
1. Open the view of any configuration tool, like Pins, then click **Update Code** (ensure desired project is selected)

2.The Update Files window pops up. It shows the detailed update information. Click **OK** button.

3.The configuration .c and .h files will be generated at “generate” folder.







HANDS ON CAN: APPLICATION CODE 1

Modify the main.c:

```c
/* Including necessary configuration files. */
#include "Mcal.h"
#include "Clock_Ip.h"
#include "FlexCAN_Ip.h"
#include "IntCtrl_Ip.h"
#include "check_example.h"

#include "Siul2_Port_Ip.h"

#define FLEXCAN0_INST 0U
#define FLEXCAN1_INST 1U
#define MSG_ID 20u
#define RX_MB_IDX 1U
#define TX_MB_IDX 0U
volatile int exit_code = 0;
/* User includes */
uint8 dummyData[8] = {1,2,3,4,5,6,7};
/*!
  \brief The main function for the project.
  \details The startup initialization sequence is the following:
 * - startup asm routine
 * - main()
*/
extern void CAN0_ORED_0_7_MB_IRQHandler(void);
extern void CAN1_ORED_0_7_MB_IRQHandler(void);

int main(void)
{
    /* Write your code here */
    Clock_Ip_Init(&Mcu_aClockConfigPB[0]);

    Siul2_Port_Ip_Init(NUM_OF_CONFIGURED_PINS0, g_pin_mux_InitConfigArr0);

    IntCtrl_Ip_EnableIrq(CAN0_ORED_0_7_MB_IRQn);
    IntCtrl_Ip_InstallHandler(CAN0_ORED_0_7_MB_IRQn, CAN0_ORED_0_7_MB_IRQHandler, NULL_PTR);

    IntCtrl_Ip_EnableIrq(CAN1_ORED_0_7_MB_IRQn);
    IntCtrl_Ip_InstallHandler(CAN1_ORED_0_7_MB_IRQn, CAN1_ORED_0_7_MB_IRQHandler, NULL_PTR);
```

Add the Port includes and Port initialization function

```c
/* Including necessary configuration files. */
#include "Mcal.h"
#include "Clock_Ip.h"
#include "FlexCAN_Ip.h"
#include "IntCtrl_Ip.h"
#include "check_example.h"

#include "Siul2_Port_Ip.h"

#define FLEXCAN0_INST 0U
#define FLEXCAN1_INST 1U
#define MSG_ID 20u
#define RX_MB_IDX 1U
#define TX_MB_IDX 0U
volatile int exit_code = 0;
/* User includes */
uint8 dummyData[8] = {1,2,3,4,5,6,7};
/*!
  \brief The main function for the project.
  \details The startup initialization sequence is the following:
 * - startup asm routine
 * - main()
*/
extern void CAN0_ORED_0_7_MB_IRQHandler(void);
extern void CAN1_ORED_0_7_MB_IRQHandler(void);

int main(void)
{
    /* Write your code here */
    Clock_Ip_Init(&Mcu_aClockConfigPB[0]);

    Siul2_Port_Ip_Init(NUM_OF_CONFIGURED_PINS0, g_pin_mux_InitConfigArr0);

    IntCtrl_Ip_EnableIrq(CAN0_ORED_0_7_MB_IRQn);
    IntCtrl_Ip_InstallHandler(CAN0_ORED_0_7_MB_IRQn, CAN0_ORED_0_7_MB_IRQHandler, NULL_PTR);

    IntCtrl_Ip_EnableIrq(CAN1_ORED_0_7_MB_IRQn);
    IntCtrl_Ip_InstallHandler(CAN1_ORED_0_7_MB_IRQn, CAN1_ORED_0_7_MB_IRQHandler, NULL_PTR);
```

Redefine the FlexCAN Instance

Enable Interrupt and Install Handler for FlexCAN_1

HANDS ON CAN: APPLICATION CODE 2

Modify the main.c:

```
Flexcan_Ip_DataInfoType rx_info = {
        .msg_id_type = FLEXCAN_MSG_ID_STD,
        .data_length = 8u,
        .is_polling = TRUE,
        .is_remote = FALSE
};
Flexcan_Ip_MsgBuffType rxData;
FlexCAN_Ip_Init(FLEXCAN0_INST, &FlexCAN_State0, &FlexCAN_Config0);
FlexCAN_Ip_SetStartMode(FLEXCAN0_INST);

FlexCAN_Ip_Init(FLEXCAN1_INST, &FlexCAN_State1, &FlexCAN_Config1);
FlexCAN_Ip_SetStartMode(FLEXCAN1_INST);

FlexCAN_Ip_ConfigRxMb(FLEXCAN1_INST, RX_MB_IDX, &rx_info, MSG_ID);
rx_info.is_polling = FALSE;

FlexCAN_Ip_Send(FLEXCAN0_INST, TX_MB_IDX, &rx_info, MSG_ID, (uint8 *)&dummyData);

FlexCAN_Ip_Receive(FLEXCAN1_INST, RX_MB_IDX, &rxData, TRUE);
while(FlexCAN_Ip_GetTransferStatus(FLEXCAN1_INST, RX_MB_IDX) != FLEXCAN_STATUS_SUCCESS)
{ FlexCAN_Ip_MainFunctionRead(FLEXCAN1_INST, RX_MB_IDX); }

FlexCAN_Ip_SetStopMode(FLEXCAN0_INST);
FlexCAN_Ip_SetStopMode(FLEXCAN1_INST);
FlexCAN_Ip_Deinit(FLEXCAN0_INST);
FlexCAN_Ip_Deinit(FLEXCAN1_INST);
Exit_Example(TRUE);
```

Initialize FlexCAN_1 Controller

Configure Rx MailBox for FlexCAN_1

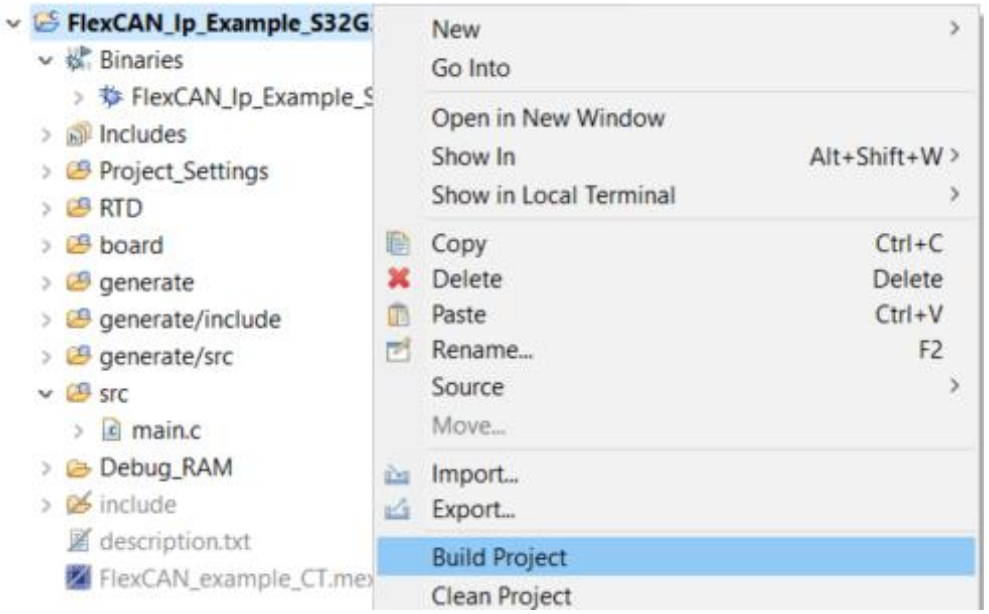
FlexCAN0 send the message

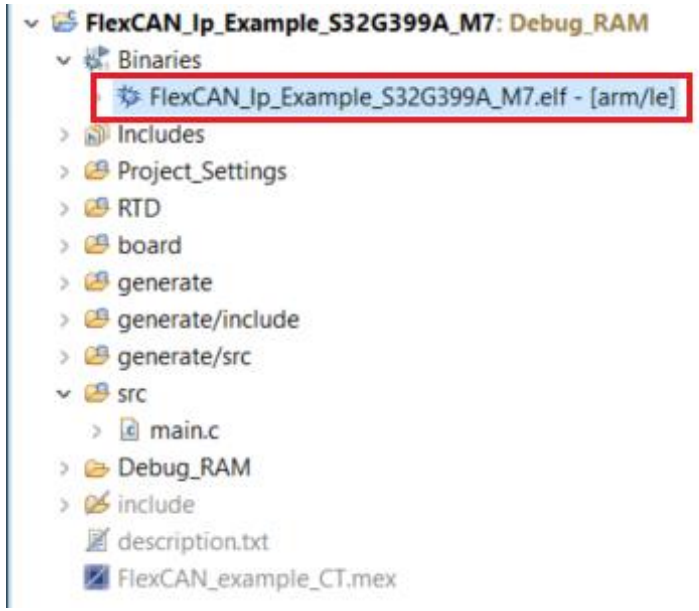
FlexCAN1 receive the message

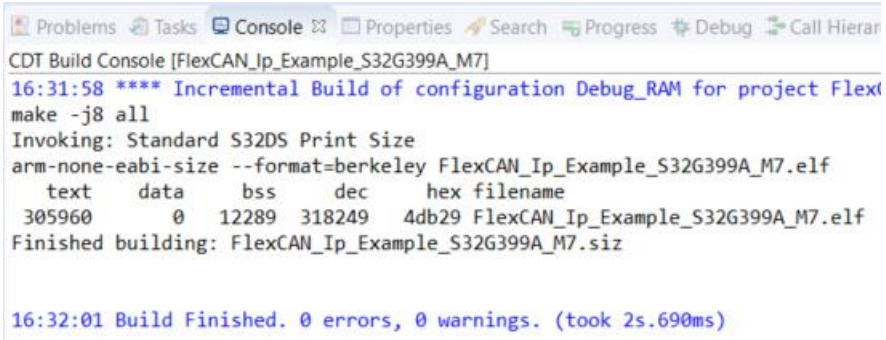
HANDS ON CAN: BUILD AND DEBUG 1

Build target Project:

- Right-click the Project
- Build Project
- The console will print build information
- FlexCAN_Ip_Example_S32G399A_M7.elf is created



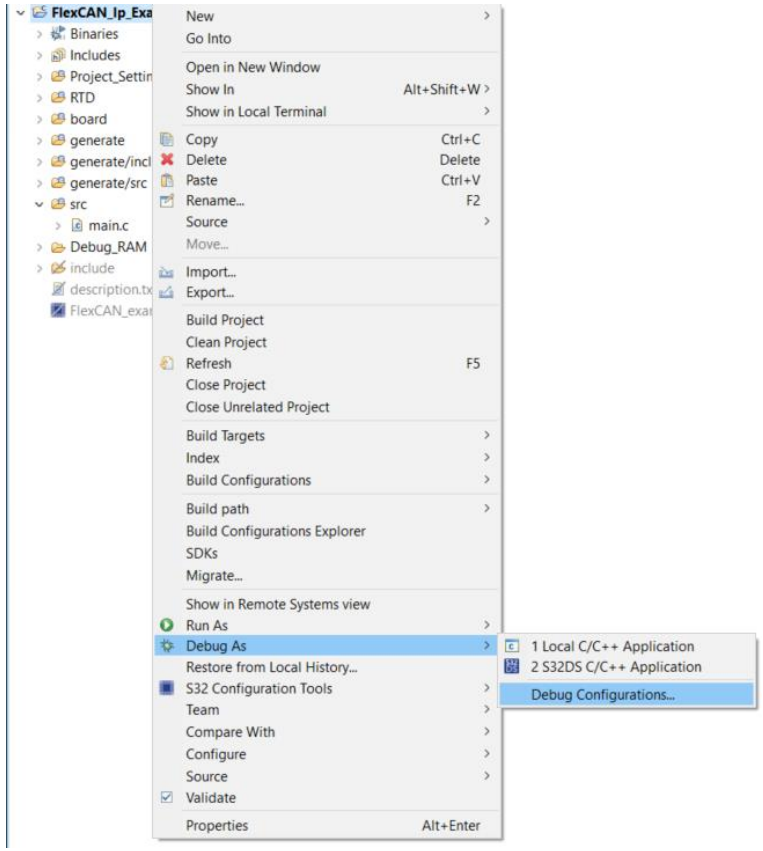




HANDS ON CAN: BUILD AND DEBUG 2

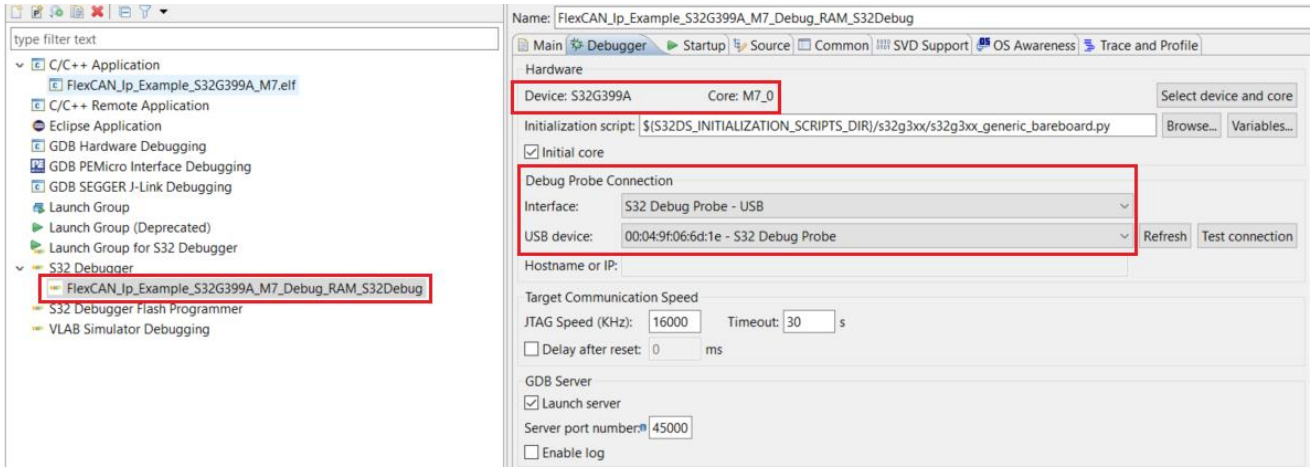
Go to debug configuration:

- Right-click the Project
- Select the Debug As
- Click on Debug Configurations



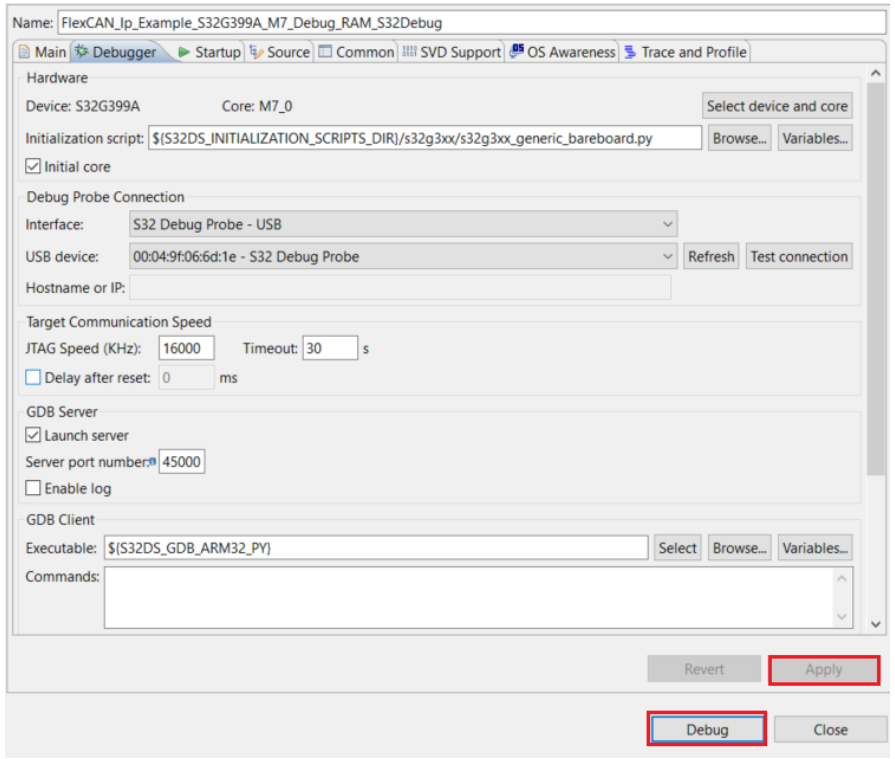
Debug configuration setting:

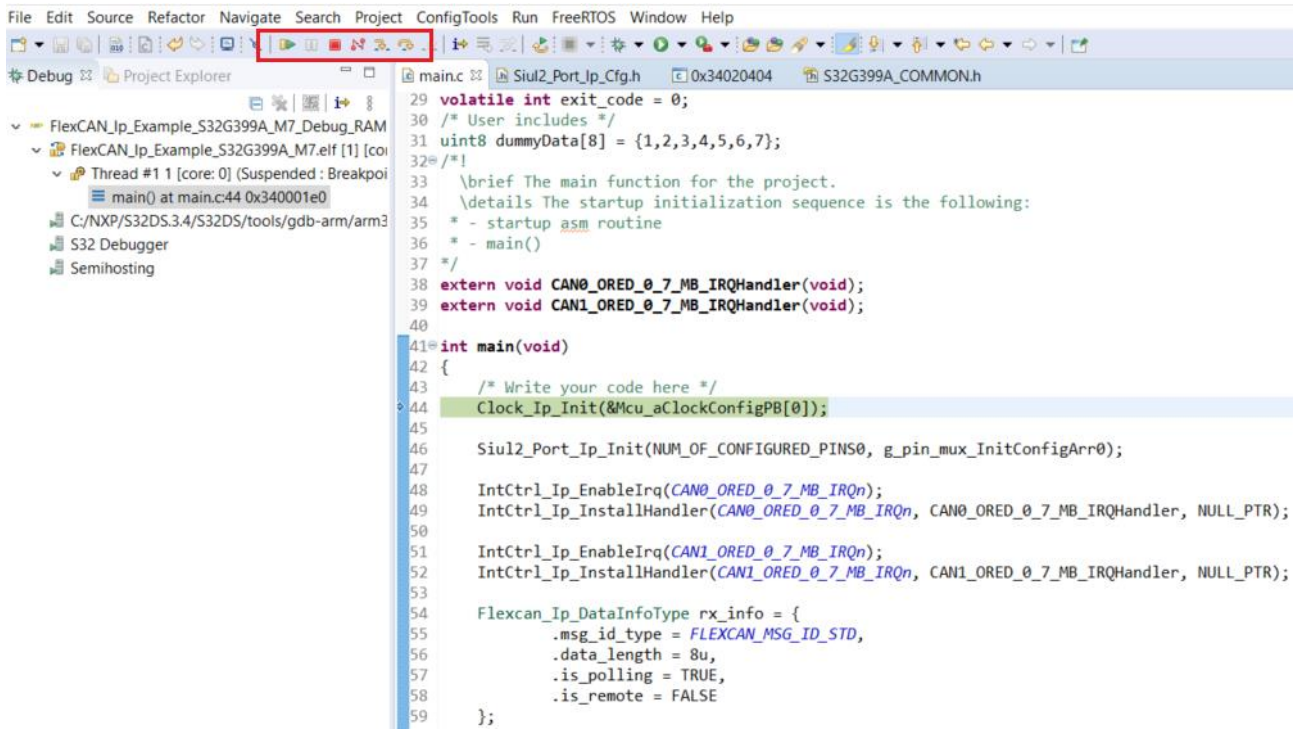
- Connect the S32 Debug probe with PC and RDB3
- Click on target project
- Select the target device and core as S32G399A_M7_0
- Select target S32 Debug Probe



HANDS ON CAN: DEBUG AND RUN

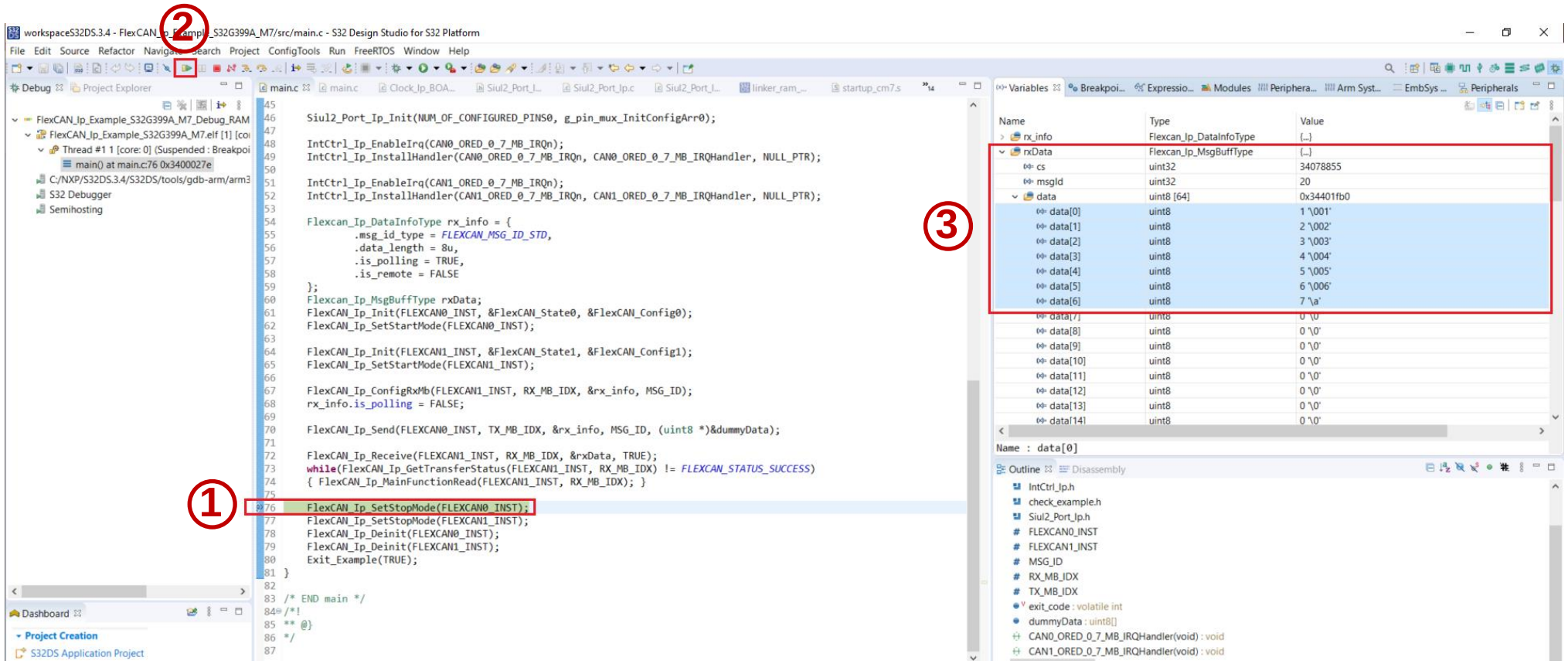
Power on the RDB3, click on “Apply”, then click on “Debug”. The view will switch to the Debug Perspective, and you can use the controls to control the program flow.





HANDS ON CAN: TEST RESULT

Add a breakpoint to the FlexCAN_Ip_SetStopMode function, then click on the Resume option. The received CAN frame can be watched from rxData.



Note: Make sure FlexCAN_0 connect to FlexCAN_1 via physical wiring

SECURE CONNECTIONS
FOR A SMARTER WORLD

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.

LEGAL INFORMATION

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 http://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.

Suitability for use in automotive applications — This NXP product has been qualified for use in automotive applications. If this product is used by customer in the development of, or for incorporation into, products or services (a) used in safety critical applications or (b) in which failure could lead to death, personal injury, or severe physical or environmental damage (such products and services hereinafter referred to as "Critical Applications"), then customer makes the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, safety, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. As such, customer assumes all risk related to use of any products in Critical Applications and NXP and its suppliers shall not be liable for any such use by customer. Accordingly, customer will indemnify and hold NXP harmless from any claims, liabilities, damages and associated costs and expenses (including attorneys' fees) that NXP may incur related to customer's incorporation of any product in a Critical Application.

LEGAL INFORMATION

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.

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.

- **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.