PTX Tunneling Library v1.4.1 for Eclipse Maxim IDE

The PTX Tunneling library can be used to evaluate and optimize the performance (antenna matching, system/RF configuration, etc.) of any custom-made device using a PTX100x device via SPI serial interface.

Embedding this library into the device firmware enables the translation of communication from **UART to SPI**, so that the full functionality of the **PTX100x * Config Tool** can be used in a custom environment. This document provides also instructions on how to create a sample application using an MAX32558-KIT development board.

Contents

1. Requirements 2

2. Sample Firmware 2

2.1 Creating the Project 2

2.2 Importing the Library 6

2.2.1. Including Tunneling Source Code 6

2.2.2. Adding the Include Path 7

2.2.3. Adding the Library File 8

2.3 Implementing the HAL 8

2.4 Calling the Library Functions 15

2.5 Building the Firmware 15

3. Preparing the Hardware 16

3.1 Running the Application 17

4. Using the Tunneling Feature 18

5. Revision History 18

1. Requirements

The footprint of the library is ~13kB Flash and 10kB RAM. Moreover, a hardware abstraction layer must be implemented by the user for the particular uC/Board, which executes the low-level commands requested by the library. From the resource point of view, only the SysTick timer, UART, SPI, and the IRQ pin will be used.

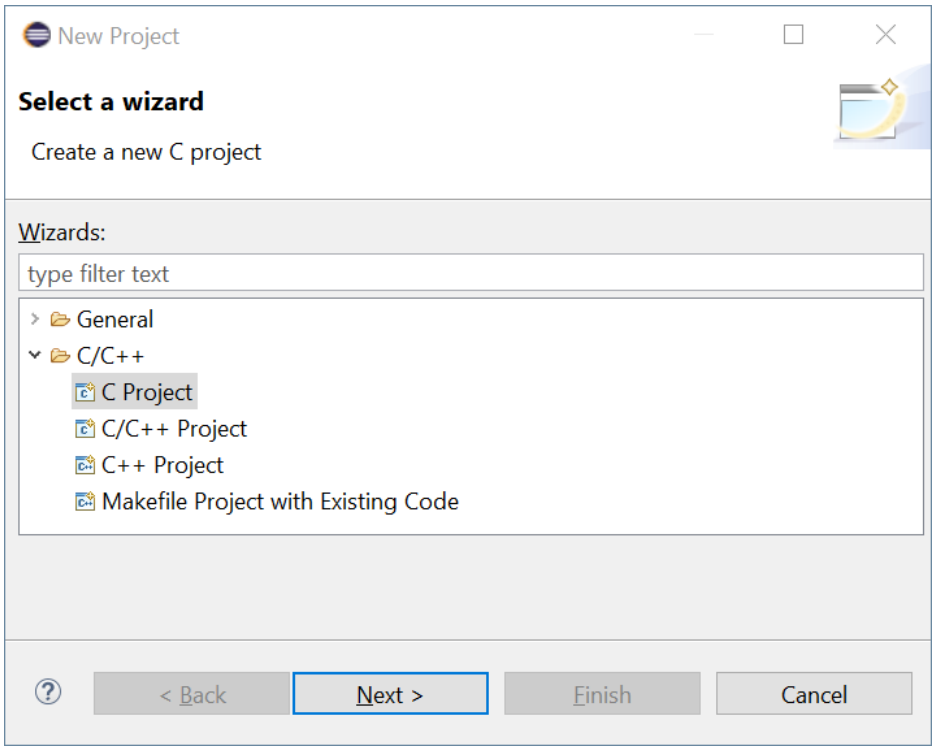
The library can be seamlessly integrated into a CMAKE project as well, but the MAX325xx is used in this document (for more information, see MAX325xx SDK 3.6.2 - Eclipse Maxim Integrated).

2. Sample Firmware

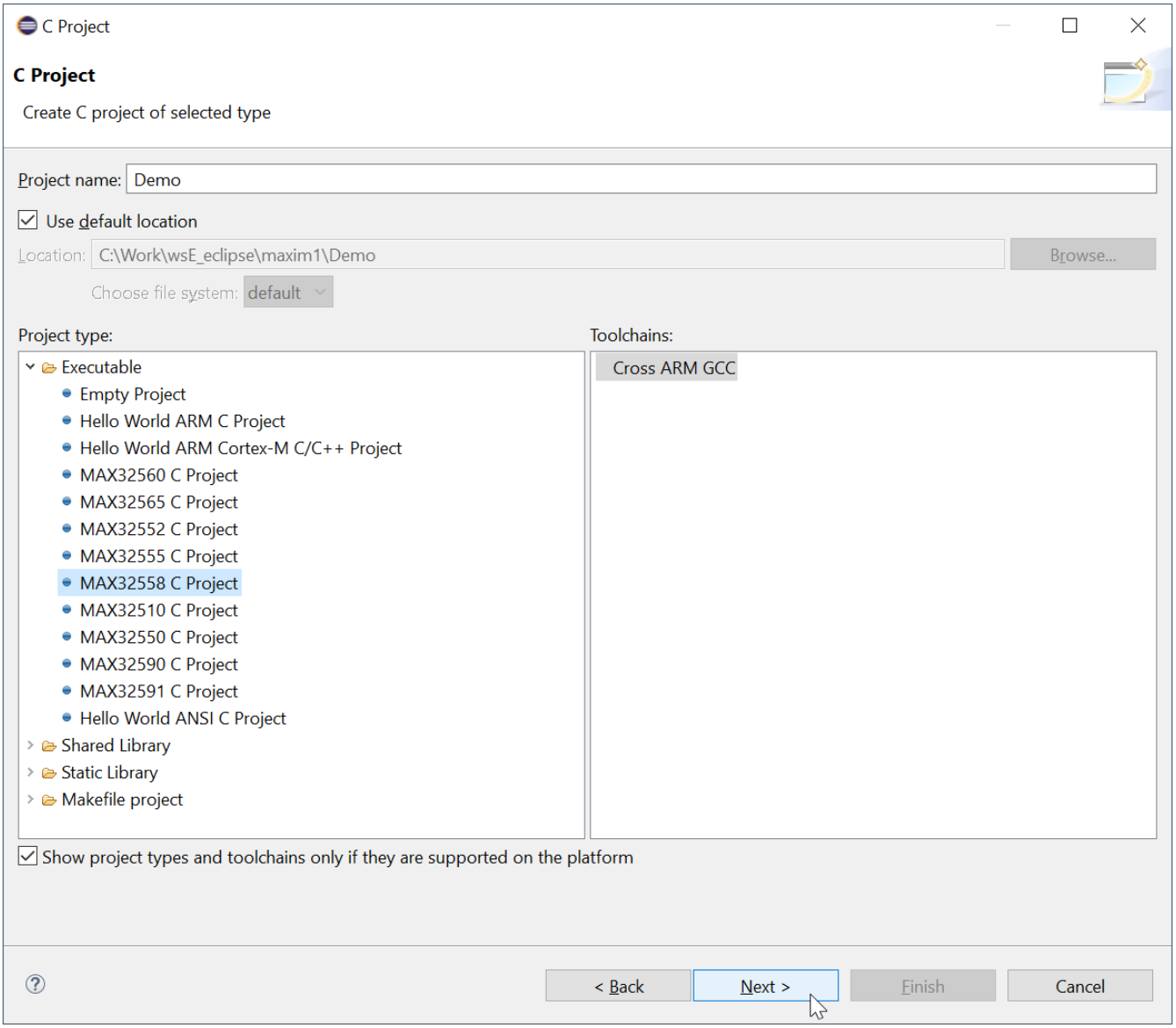
The sample application is used for creating and serving the tunnel between the host PC UART interface and the PTX100x chip connected by SPI. The library can be used either as a precompiled static library or as a source-library – most steps are the same for both cases.

2.1 Creating the Project

After selecting the **File menu > New > Project**, a wizard window will open to guide the user through the project creation process. Choosing the **C Project** from the list of templates and clicking the **Next** button will prepare the proper build environment.



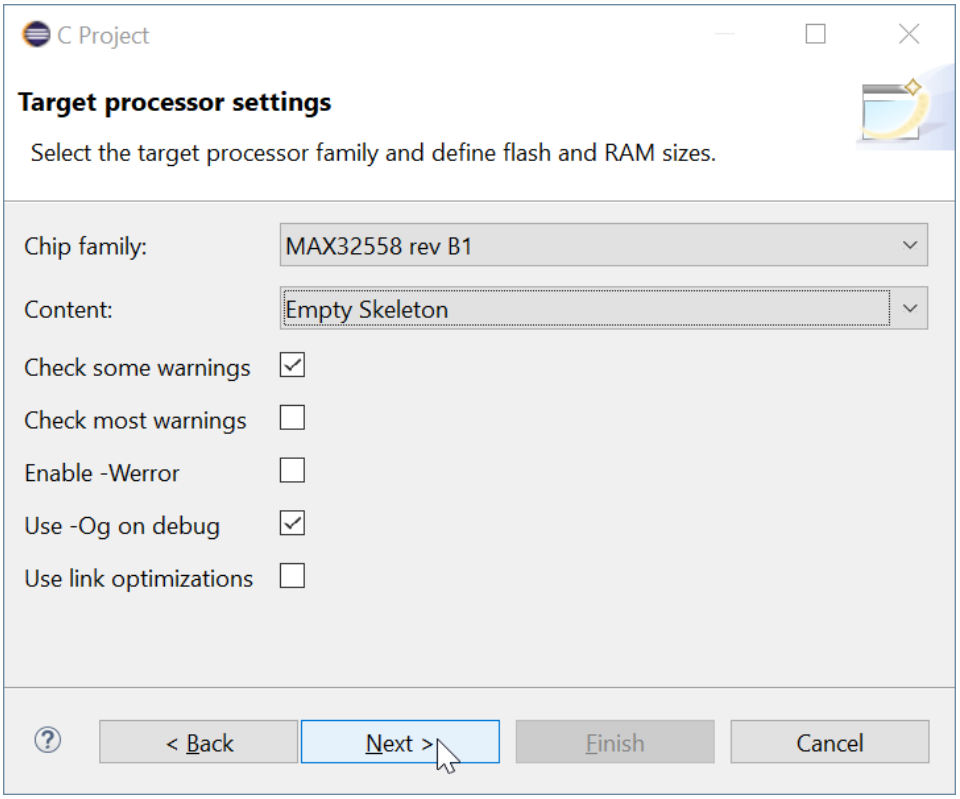
For convenience, the project will be called "Demo" and selecting the **MAX32558 C Project** tells the compiler which MCU will be the target platform.



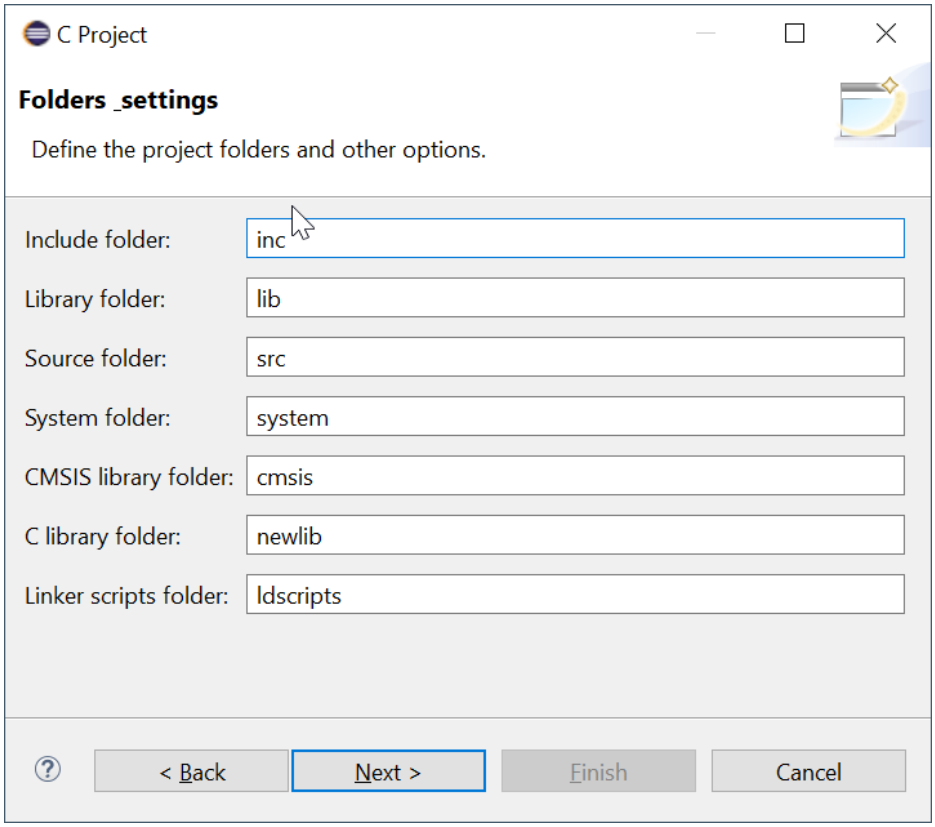
Next, the chip variant must be selected, which in this case is **MAX32558 rev B1**. This step will also configure the correct settings for the selected processor.

Note: The project will also work on other microcontrollers if the pre-built library architecture is compatible.

As a template content, **Empty skeleton** should be selected, which has no other content added to the project.



In the next step, the IDE offers to set up the folder structure. Because the library will be imported, it makes sense to prepare the folder names as they appear in the archive.

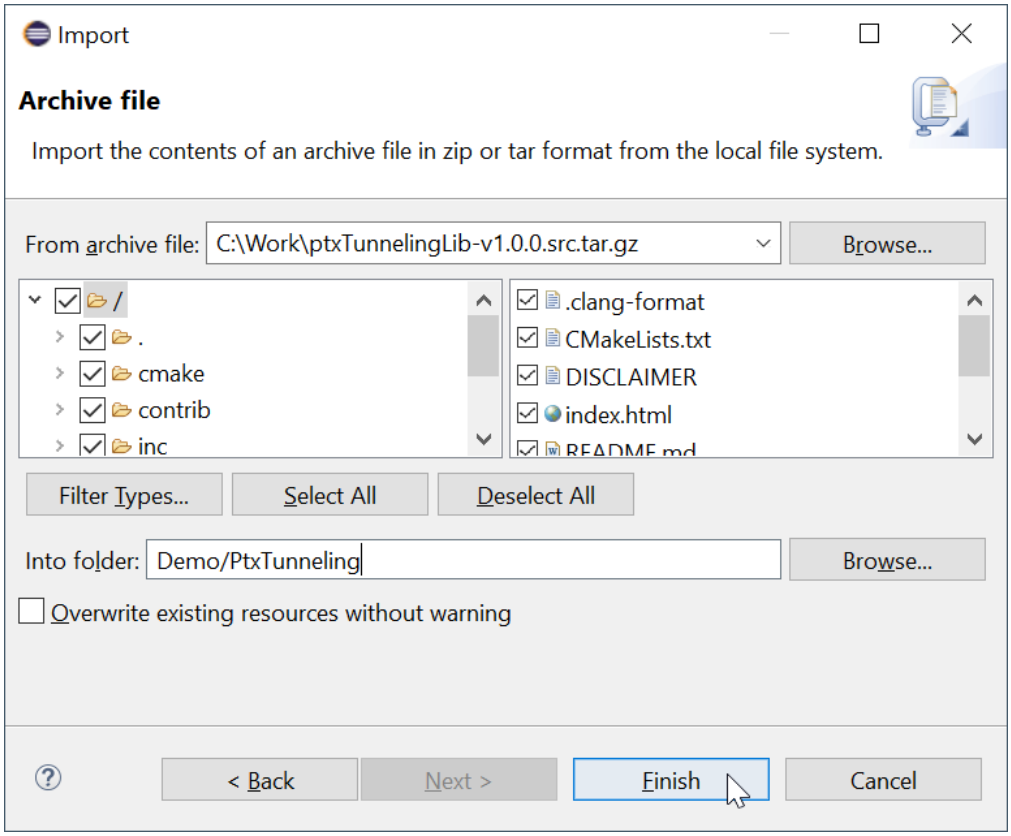


After pressing the **Next** button a few times and accepting the default target and compiler selection, the initial project gets created

2.2 Importing the Library

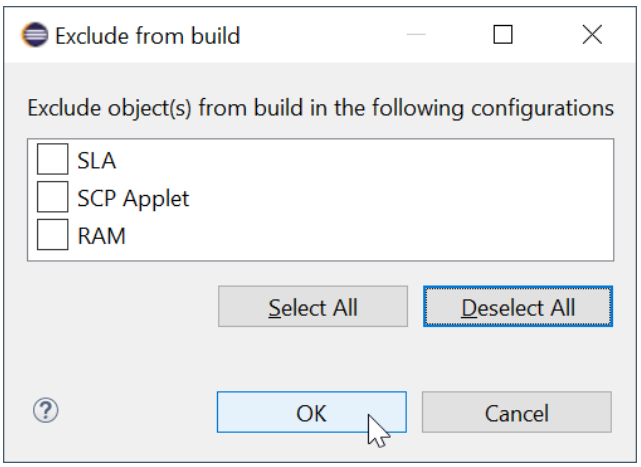
There is no difference whether the source or the precompiled package is being used: the library archive must be imported to the project using **File > Import > Archive File**.

To keep the folder structure clean, the library will be imported to the `PtxTunneling` subfolder by appending it to the default location shown in following figure.



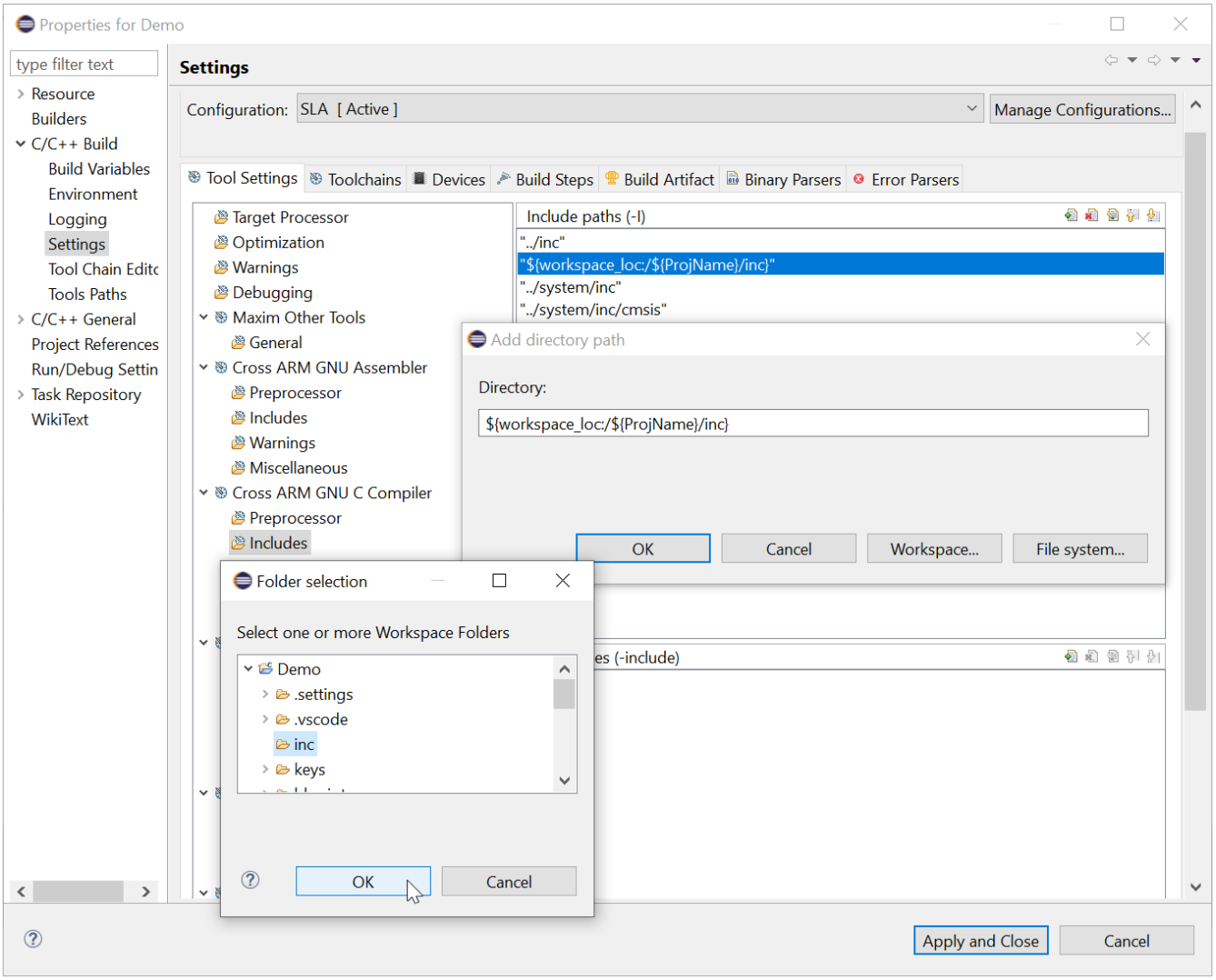
2.2.1. Including Tunneling Source Code

Should the library be used as source code, the subfolder `PtxTunneling` and `PtxTunneling/src` folders need to be included in the build. This can be done by opening the context menu with right mouse click on the folder name in the **Project Explorer** and selecting **Resource Configurations > Exclude from Build**. In the dialog window, for each (default) target the check can be removed, thus selecting the folder content for build.



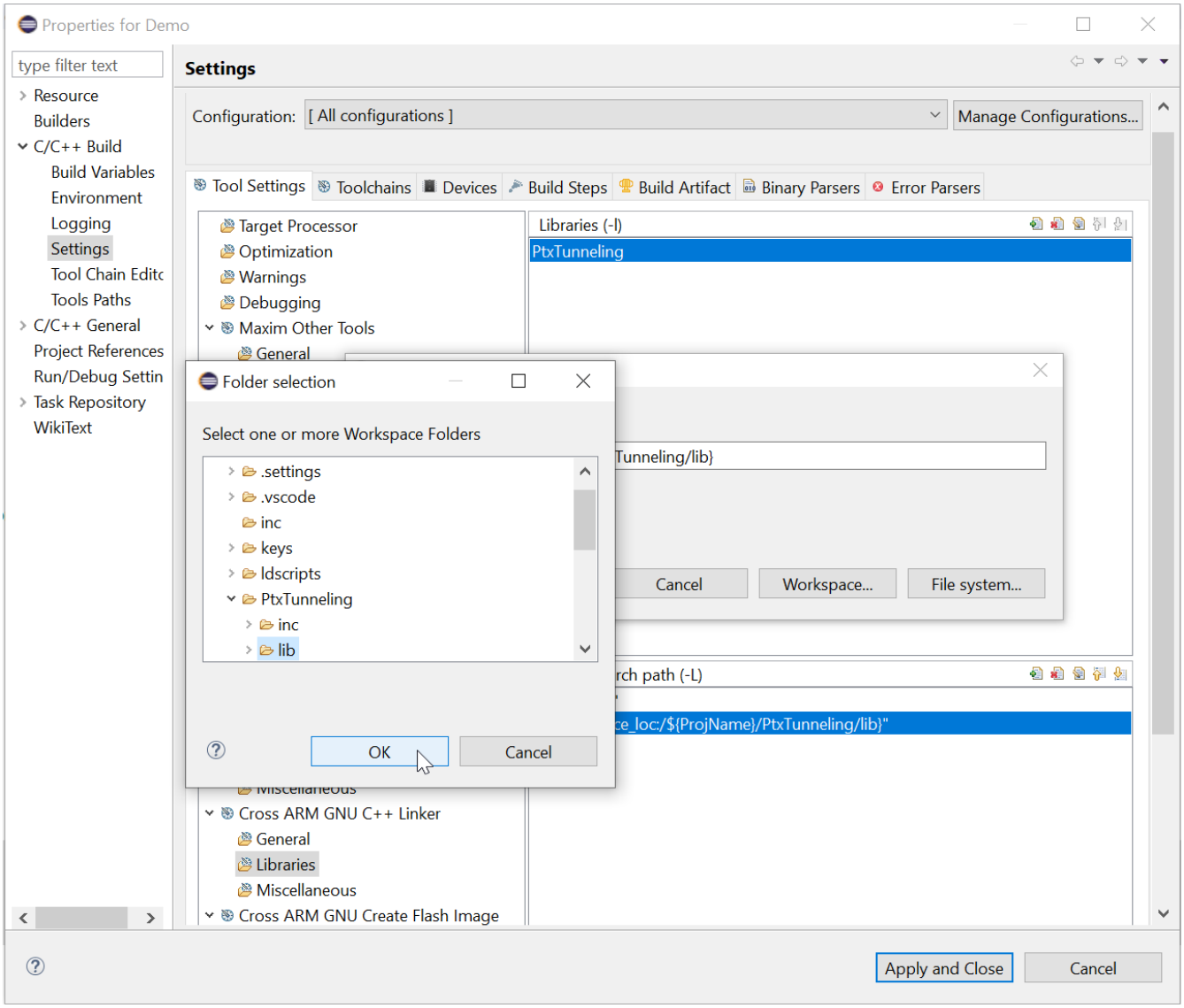
2.2.2. Adding the Include Path

In order that the compiler can find the header (.h) files containing the API functions, the library folder `inc` needs to be added to the list of user-defined include directories. This can be done by navigating to **Project > Properties > C/C++ Build > Settings > Tool Settings > Cross ARM GNU C Compiler > Includes**. Click on the **Add** button on the right side of the small toolbar and use the **Workspace** button in the popup window to locate the folder.



2.2.3. Adding the Library File

This step is required only if you are working with the precompiled binary package. Since there is no source code to be compiled, the linker must be able to find the functions in the library. In the same dialog window, changing to **Cross ARM GNU C++ Linker > Libraries** section, the `PtxTunneling/lib` folder can be added to the list of folders (lower pane) where the compiler is looking for external libraries. Additionally, the exact library needs also to be specified (upper pane) by its name `PtxTunneling`. From this the compiler will automatically find the static library file `libPtxTunneling.a`.



2.3 Implementing the HAL

If the library functions cannot access the underlying hardware or software resources, they require access to the Hardware Abstraction Layer (HAL), which then performs the requested action. Since this layer depends on the specific hardware configuration, it must be implemented for the exact setup.

The PtxTunneling library includes the header file ptx_tunneling_hal.h, which contains all the functions that must be provided by the host platform.

For the current case, there should be the file `ptx_tunneling_hal.c` created in the source code folder `Core/Src` with the following content.

```
/*
    ------------------------------------------------------------------
    SPDX-License-Identifier: BSD-3-Clause

    Copyright (c) 2024, Renesas Electronics Corporation and/or its affiliates


    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 provided with the distribution.

    3. Neither the name of Renesas 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 Renesas "AS IS" AND ANY EXPRESS
    OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
    OF MERCHANTABILITY, NONINFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE ARE
    DISCLAIMED. IN NO EVENT SHALL RENESAS 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.
    ------------------------------------------------------------------

    Project     : PtxTunneling
    Module      : HAL
    File        : ptx_tunneling_hal.c

    Description : Implementation of HAL for tunneling
*/

#include <MAX325xx.h>
#include <assert.h>
#include <cmsis_gcc.h>
#include <cmsis_nvic.h>
#include <mml_gpio.h>
#include <mml_spi.h>
#include <mml_tmr.h>
#include <mml_uart.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>

#include "ptx_tunneling_hal.h"

/*
 * Pin assignment for MAX 32588 EV KIT
 *
 * SPI MISO 0   P0.16
```

```
 * SPI MOSI 0   P0.17
 * SPI SCK  0   P0.18
 * SPI NSS  0_0 P0.19
 *
 * SEN          P0.10
 *
 * Ext. IRQ     P0.21
 */

#define EX_SPI_BAUD_RATE 5000000
#define PTX100_SPI_DEV MML_SPI_DEV0

#define PTX100_UART_DEV MML_UART_DEV0
#define PTX100_UART UART0
#define PTX_IRQ GPIO(0, 21)
#define PTX_CS GPIO(0, 19)

#define TIMER_INVALID -1
#define TMPBUFLEN 300

#ifndef UNUSED
#define UNUSED(var) (void)var
#endif

struct ptxHal
{
};

static volatile uint32_t g_timeFromStart = 0;
static uint8_t g_tmpBuf[TMPBUFLEN];

static void isrSysTick(); // System timer interrupt service routine
static int init_spi_master(void);

#define OFFSET_RSP_LENGTH_BYTE 0
#define OFFSET_CMD_LENGTH_BYTE 1
#define OFFSET_CMD_CODE_BYTE 0

#define COMMS_MAX_MESSAGE_LENGTH 280
#define COMMS_HEADER_SIZE 2

#define CMD_CODE_TUNNELING_MSG 0x55

static uint8_t uartRxBuf[2048]; // temporary uart buffer for received data
static volatile uint16_t readPos = 0;
static volatile uint16_t writePos = 0;
static uint8_t rx[COMMS_MAX_MESSAGE_LENGTH]; // for prefiltering received commands
static uint16_t rxi = 0;

void uartCallback()
{
    mml_uart_interrupt_clear(PTX100_UART_DEV, UINT32_MAX);
    // read received data from buffer
    while (PTX100_UART->STAT & UARTn_STAT_RXELT_Msk)
    {
        volatile uint8_t tmp;
        tmp = PTX100_UART->DATA;

        if (rxi == 0 && tmp != CMD_CODE_TUNNELING_MSG)
        { // if there are some invalid data received while waiting for sync byte, just
discard them
```

```
            continue;
        }
        rx[rxi++] = tmp;
    }

    if (rxi >= COMMS_HEADER_SIZE)
    {
        uint16_t packLen = COMMS_HEADER_SIZE +
                           (rx[OFFSET_CMD_LENGTH_BYTE] == 0 ? 256 :
rx[OFFSET_CMD_LENGTH_BYTE]);
        if (rxi >= packLen)
        {
            // whole packet has been received
            memcpy(uartRxBuf + writePos, rx, rxi);
            writePos += rxi;
            rxi = 0;
        }
    }
}

bool ptxTunneling_GPIO_IsIrqPinAsserted(ptxHal_t *context)
{
    UNUSED(context);
    unsigned int data;
    mml_gpio_read_bit_pattern(GPIO_DEV(PTX_IRQ), GPIO_NUM(PTX_IRQ), 1, &data);
    return data > 0;
}

int ptxTunneling_UART_rxLength(ptxHal_t *context)
{
    UNUSED(context);
    __disable_irq();
    const count = writePos - readPos;
    __enable_irq();
    return count;
}

int ptxTunneling_UART_read(ptxHal_t *context, uint8_t *buf, unsigned int len)
{
    UNUSED(context);
    assert(len < sizeof(uartRxBuf));
    __disable_irq();
    int readCount = writePos - readPos;
    if (readCount > len)
        readCount = len;

    memcpy(buf, uartRxBuf + readPos, readCount);

    readPos += readCount;
    if (readPos == writePos)
    {
        readPos = 0;
        writePos = 0;
    }
    __enable_irq();
    return readCount;
}

int ptxTunneling_UART_write(ptxHal_t *context, const uint8_t *buf, unsigned int len)
{
    UNUSED(context);
```

```
    int res = 0;
    while (len--)
    {
        res = mml_uart_write_char(PTX100_UART_DEV, *buf++);
        assert(!res);
    }

    return res;
}

void ptxTunneling_Timer_stopwatchStart(ptxHal_t *context, ptxTimeDiff_t *startVal)
{
    UNUSED(context);
    *startVal = g_timeFromStart * 1000;
}

void ptxTunneling_Timer_stopwatchStop(ptxHal_t *context, ptxTimeDiff_t *startStopVal)
{
    UNUSED(context);
    *startStopVal = g_timeFromStart * 1000 - *startStopVal;
}

void ptxTunneling_Timer_ThreadSleep(ptxHal_t *context, uint32_t msSleep)
{
    UNUSED(context);
    const uint32_t t0 = g_timeFromStart;
    while (g_timeFromStart - t0 < msSleep)
    {
    }
}

int ptxTunneling_SPI_trx(ptxHal_t *context, uint8_t *const txBuf[], const size_t
txLen[],
    size_t numBuffers, uint8_t *rxBuf, size_t *rxLen)
{
    UNUSED(context);
    size_t index;
    int st = COMMON_ERR_UNKNOWN;

    /* At this point the SPI transfer operation is triggered */
    mml_gpio_write_bit_pattern(GPIO_DEV(PTX_CS), GPIO_NUM(PTX_CS), 1, 0);

    // Tx operation is required always: to send and to receive anything on SPI. So, tx
buffers have
    // to be provided always.
    if ((NULL != txBuf) && (NULL != txLen))
    {
        /* Tx part of the overall transaction. */
        index = 0;
        while (index < numBuffers)
        {
            assert((txBuf[index] != NULL) && (txLen[index] > 0));
            const size_t len = txLen[index];
            assert(len <= TMPBUFLEN);
            // the transmit function also overwrites the input buffer, therefore it is
copied
            // beforehand
            memcpy(g_tmpBuf, txBuf[index], len);
            st = mml_spi_transmit(PTX100_SPI_DEV, g_tmpBuf, len);
            assert(!st);
            if (rxBuf)
```

```
                memcpy(rxBuf, g_tmpBuf, len);
            index++;
        }
    }
    else if ((NULL != rxBuf) && (NULL != rxLen) && (*rxLen > 0))
    /* Let's see if there is something to read. */
    {
        // the transmit function transmits buffer and replaces the content with received
bytes
        memset(rxBuf, 0, *rxLen);
        st = mml_spi_transmit(PTX100_SPI_DEV, rxBuf, *rxLen);
        assert(!st);
    }

    // de-assert cs
    mml_gpio_write_bit_pattern(GPIO_DEV(PTX_CS), GPIO_NUM(PTX_CS), 1, 1);

    return st;
}

void ptxTunneling_NVIC_disableInterrupts()
{
    __disable_irq();
}

void ptxTunneling_NVIC_enableInterrupts()
{
    __enable_irq();
}

void initPeripherals()
{
    int st;
    mml_gpio_pre_init();
    mml_tmr_pre_init();

    // System tick to 1ms resolution
    __NVIC_SetVector(SysTick_IRQn, (uint32_t)&isrSysTick);
    SysTick_Config(60000);

    init_spi_master();

    // configure PTX_IRQ pin
    mml_gpio_config_t gpioConfIrq = {.gpio_direction = MML_GPIO_DIR_IN,
        .gpio_function = MML_GPIO_NORMAL_FUNCTION,
        .gpio_intr_mode = MML_GPIO_INT_MODE_LEVEL_TRIGGERED,
        .gpio_intr_polarity = MML_GPIO_INT_POL_HIGH,
        .gpio_pad_config = MML_GPIO_PAD_PULLDOWN};

    mml_gpio_init(GPIO_DEV(PTX_IRQ), GPIO_NUM(PTX_IRQ), 1, gpioConfIrq);

    // Configure PTX_CS pin
    mml_gpio_config_t gpioConfCs = {.gpio_direction = MML_GPIO_DIR_OUT,
        .gpio_function = MML_GPIO_NORMAL_FUNCTION,
        .gpio_pad_config = MML_GPIO_PAD_NORMAL};

    mml_gpio_init(GPIO_DEV(PTX_CS), GPIO_NUM(PTX_CS), 1, gpioConfCs);
    mml_gpio_write_bit_pattern(GPIO_DEV(PTX_CS), GPIO_NUM(PTX_CS), 1, 1);

    rxi = 0;
    mml_uart_config_t uartConfig = {.baudrate = 115200,
```

```
        .data_bits = UARTn_CTRL_SIZE_bits8,
        .flwctrl = UARTn_CTRL_RTSCTSF_disable,
        .parity = MML_UART_PARITY_NONE,
        .stop_bits = UARTn_CTRL_STOP_stop1,
        .handler = &uartCallback,
        .rts_ctl = UARTn_PIN_RTS_hi,
        .parity_mode = MML_UART_PARITY_MODE_ONES};

    mml_uart_pre_init();
    st = mml_uart_init(PTX100_UART_DEV, uartConfig);
    assert(!st);
    mml_uart_interrupt_set(PTX100_UART_DEV, UARTn_INT_EN_FFRXIE_Msk);
    ptxTunneling_NVIC_enableInterrupts(NULL);
}

// Initialize SPI
static int init_spi_master(void)
{
    int result = NO_ERROR;
    mml_spi_params_t spiparams;

    spiparams.baudrate = EX_SPI_BAUD_RATE;
    spiparams.ssel = 0;
    spiparams.word_size = SPIn_MOD_NUMBITS_bits8;
    spiparams.mode = SPIn_CNTL_MMEN_master;
    spiparams.wor = SPIn_CNTL_WOR_disable;
    spiparams.clk_pol = SPIn_CNTL_CLKPOL_idleLo;
    spiparams.phase = SPIn_CNTL_PHASE_activeEdge;
    spiparams.brg_irq = SPIn_CNTL_BIRQ_disable;
    spiparams.ssv = SPIn_MOD_SSV_hi;
    spiparams.ssio = SPIn_MOD_SSIO_output;
    spiparams.tlj = SPIn_MOD_TX_LJ_disable;
    spiparams.dma_rx.active = SPIn_DMA_REG_DMA_EN_disable;
    spiparams.dma_tx.active = SPIn_DMA_REG_DMA_EN_disable;

    result = mml_spi_reset_interface();
    if (result)
        return result;

    result = mml_spi_init(PTX100_SPI_DEV, &spiparams);
    if (result)
        return result;

    M_MML_SPI_ENABLE(PTX100_SPI_DEV);
    return result;
}

// count milliseconds with SysTick
static void isrSysTick()
{
    g_timeFromStart++;
}
```

Note: This implementation is specific to the *MAX32558-KIT* board. It is not guaranteed to work on any other hardware.

2.4 Calling the Library Functions

After the initialization by `ptxTunneling_init()`, the main loop will provide the tunneling functionality by calling the library's superloop function, the `ptxTunneling_poll()`. This function performs the data processing and translation, and also the SPI communication. The **main()** function can be found in `src/main.c` file. Update this file with the following code:

```
#include <stdio.h>
#include <stdlib.h>
#include "ptx_tunneling.h"

extern void initPeripherals();

int main(int argc, char *argv[])
{
    initPeripherals();
    ptxTunneling_init();

    while (1)
    {
        ptxTunneling_poll(NULL);
    }

    return 0;
}
```

2.5 Building the Firmware

After the source files have been created, the project can be built with the **Project > Build Project**. When the build process has finished successfully, a table similar to the following will show with the footprint sizes.

```
arm-none-eabi-size --format=berkeley "demo.elf"
   text    data     bss     dec     hex filename
  14357     168   16656   31181    79cd demo.elf
```

3. Preparing the Hardware

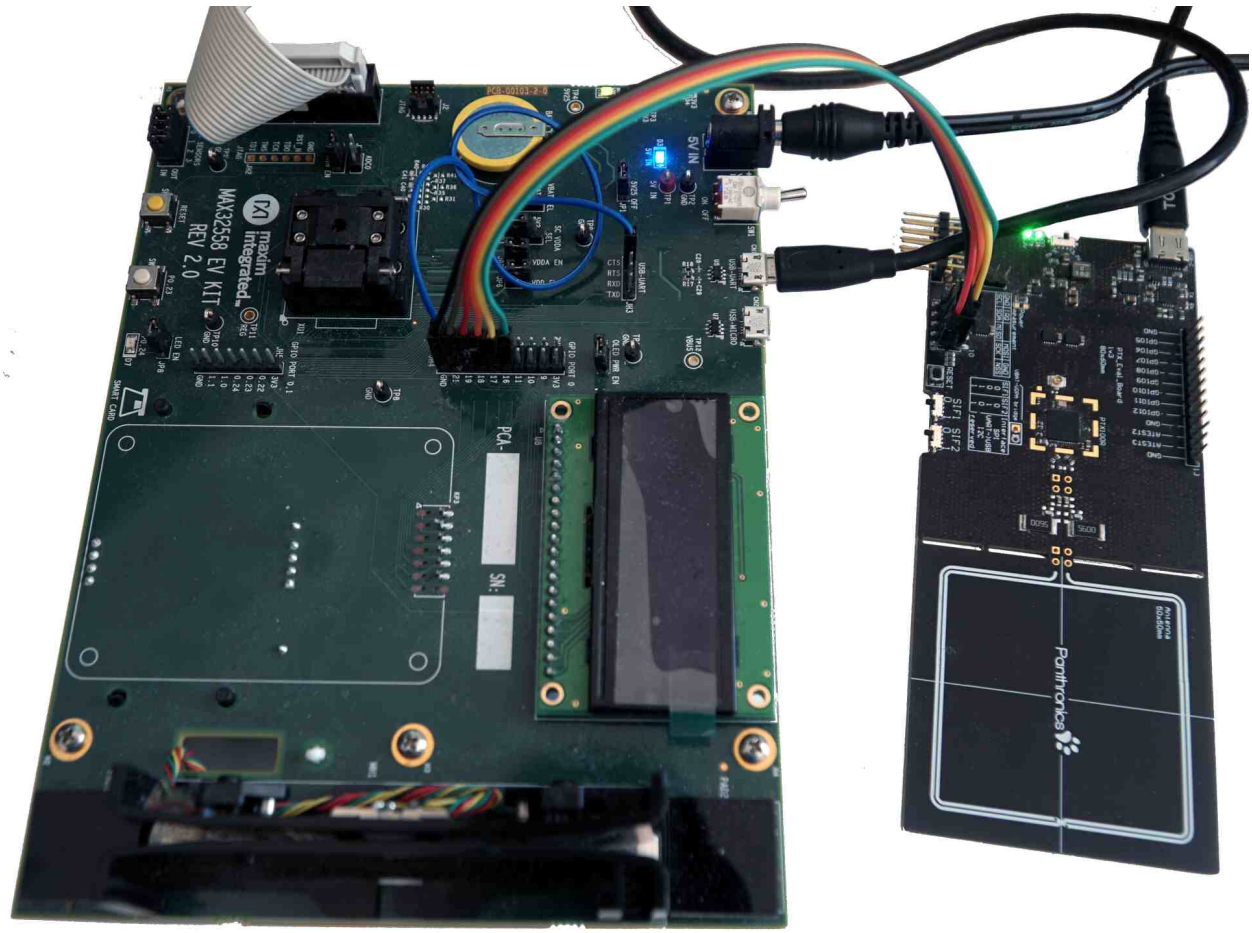
Configure the PTX evaluation board's serial interface to SPI by setting both interface configuration switches (**SIF1** and **SIF2**) to **0** and remove the jumper labeled **pmod 3v3** (located next to PMOD connector) to prevent supply conflicts between the two boards, since each will be powered separately.

Connect the PTX evaluation board to the MAX32558-KIT board with jumper cables. Using the vertical pin header on the PTX evaluation board is convenient because it shows the pin names on the silkscreen, therefore they are easy to identify. On the Maxim board, the pin header **JH4** with the **GPIO PORT 0** label will be used with the following interconnection.

Signal Name on PTX Evaluation Board	Maxim Board JH4 Pin Name
MISO	16
MOSI	17
SCK	18
SSN	19
IRQ	21
GND	GND

In addition, the **CTS** pin of the **JH3** connector must be pulled to **GND**. The boards can now be powered up.

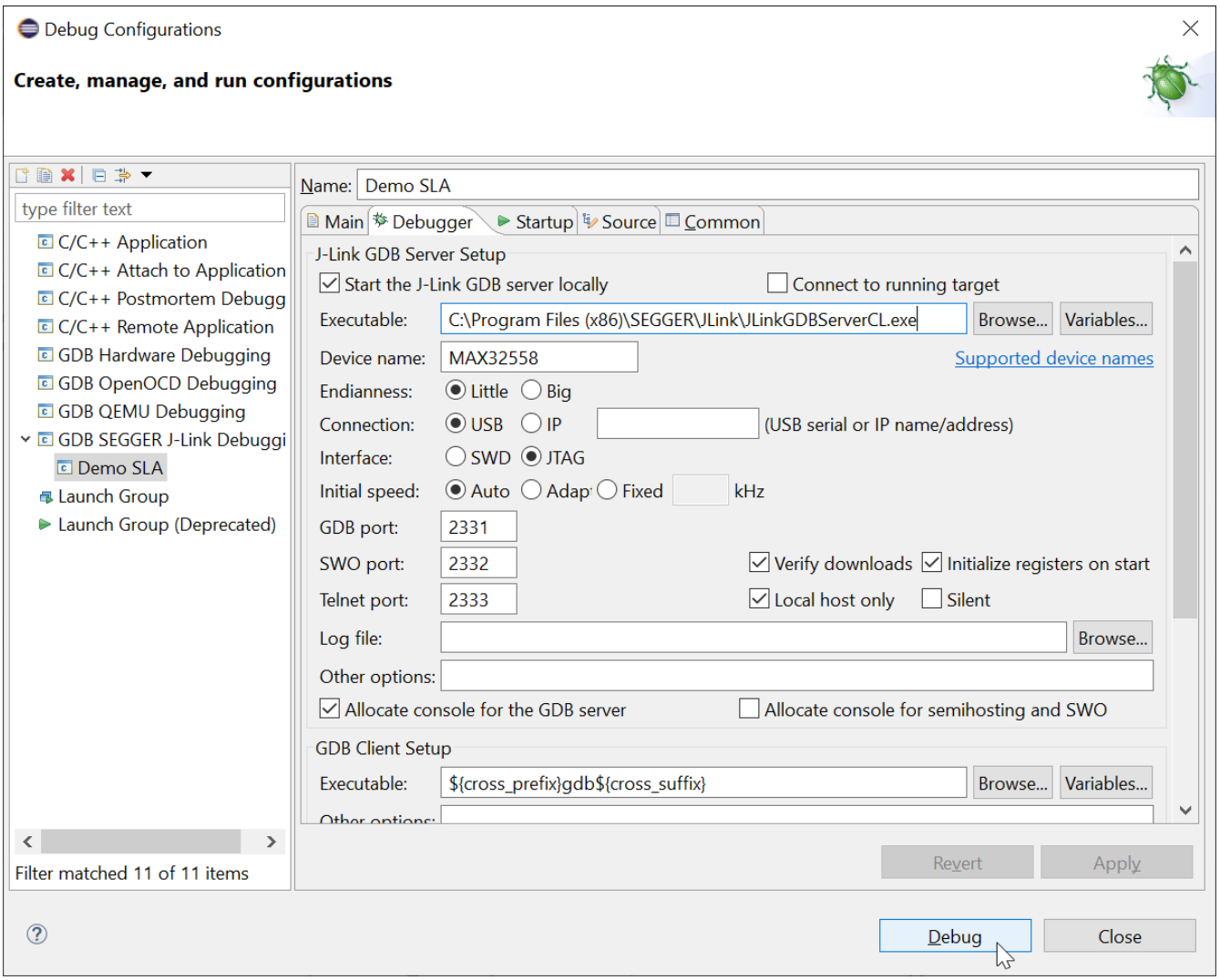
Note: For optimal RF performance, the **USB 3.0** port should be used to enable the PTX board to draw a current up to 900mA.



3.1 Running the Application

The last task is to flash and start the firmware on the Maxim evaluation board. This can be performed directly in the development environment of the Maxim SDK.

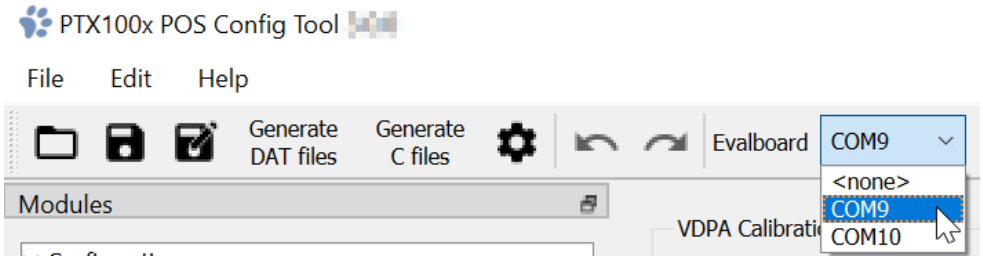
There is a J-Link debug probe in the MAX32558 evaluation kit for which a debug configuration is already generated automatically during project setup. It is possible that the internal variable `jlink_path` is set incorrectly during installation; therefore, debugging will not work out of the box. If this occurs, you must change the debug configuration in the **Run > Debug Configurations** menu. Selecting the **Demo SLA** and the **Debugger** tab, the GDB server **Executable** in the **J-Link GDB Server Setup** area can be set manually as shown below.



Finally, after clicking on the **Debug** button, the firmware will be uploaded to the Maxim board and the debug session will be started. After pressing **F8** to let the demo application run, the firmware will be ready for accepting communication frames from the PC.

4. Using the Tunneling Feature

To use of the tunneling functionality, the **PTX100x * Config Tool** must be started and configured to use the USB serial communication port (identified in Device manager previously) by selecting the correct entry in the dropdown list in toolbar.



The configuration is now ready. Any test started will communicate with the PTX100x via the tunneling firmware.

5. Revision History

Revision	Date	Description
1.01	Jun 18, 2024	Updated license text in ptx_tunneling_hal.c file.
1.00	Jan 16, 2024	Initial release.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.