PTX IoTRD Library for Renesas Synergy™ Platform MCUs

Panthronics AG - IoT Reader Support

Exported on 05/03/2021

Table of Contents

1 Introduction

This document describes building the PTX IoT demo application using **Renesas e² studio** and running it on the Renesas Synergy TB-S3A1 Evaluation board.

To enable using the library on a **broad range of Synergy platform** devices, the demo package contains precompiled libraries for ARM Cortex-M0+ and Cortex-M4 architectures.

The following topics are covered in this document:

- Introduction to PTX_EvalBoard 1v3
- Overview of PTX IoT library
- PTX IoT library integration
- PTX IoT demo app

2 Requirements

The application has the following hardware requirements:

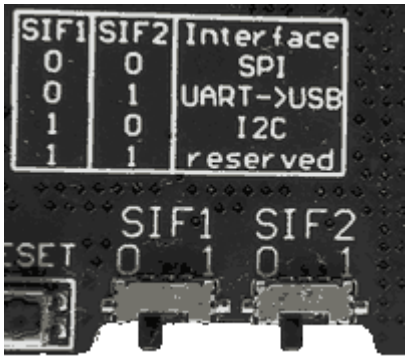
- Renesas Synergy TB-S3A1 kit
- PTX Evalboard 1v3
- USB type C cable
- Micro USB cables 2pcs

Building and running the application require the following tools and software to be pre-installed:

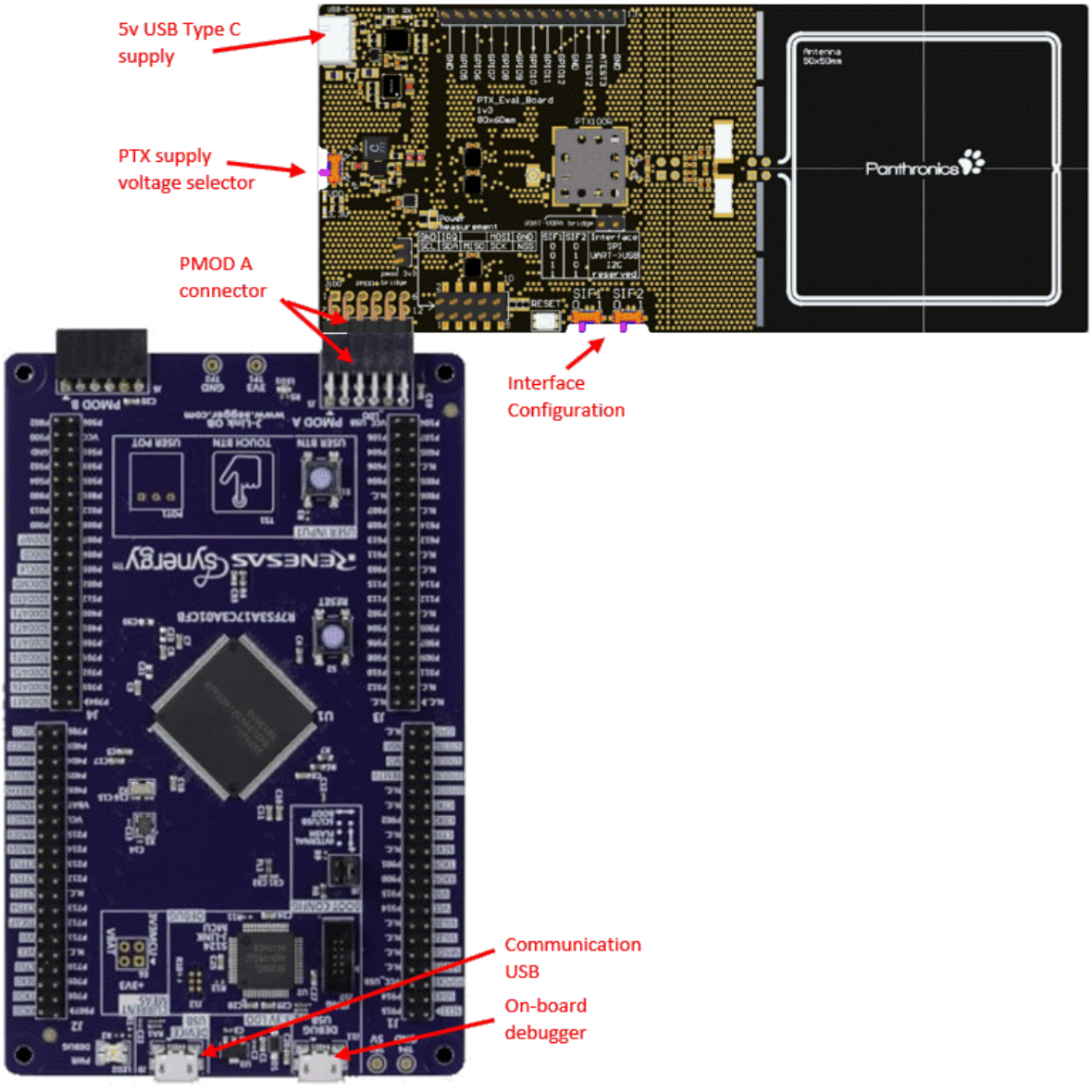
- e^2 studio (tested with version 7.8.0)
- SSP Distribution (tested with version 1.7.5)
- Toolchain *GCC ARM Embedded* Version 7.2.1.20170904
- PuTTY or similar terminal application

3 Hardware setup

1. Make sure that the SIF1 and SIF2 switches from PTX Evalboard are set to the interface type “SPI”



2. Connect the PTX Evalboard to TB-S3A1 PMOD A connector



3. Plugin both micro USB cables

4. Plugin the Type C USB cable to supply the boards
5. Flash the firmware using the onboard debugger
6. Use PuTTY to see the application output

NOTES:

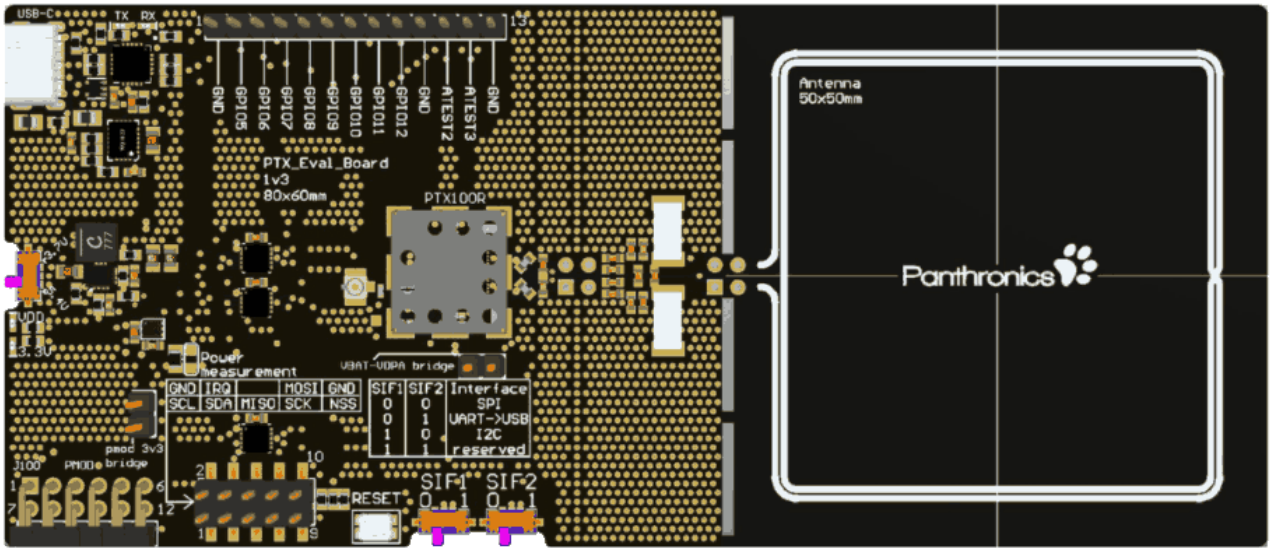
- If the USB port on the PC side can supply 900mA (i.e a USB-C or USB-3 port) the PTX Evaluation Board can achieve its maximum output power.
- The Type C USB cable will supply also the TB-S3A1 3v3 channel, if the pmod 3v3 bridge jumper is connected, no other power supply is needed.
- When the debugger cable is connected, the TB-S3A1 board will supply also the PTX Evalboard if the pmod 3v3 bridge jumper is connected, but it will have a low performance.

3.1 PTX Evalboard 1v3

The PTX Evalboard has the purpose of evaluating PTX100R performance.

3.1.1 Purpose

The PTX Evalboard shall serve to demonstrate the performance of PTX100R for IoT application: Full support for all technologies (A, B, F, V)



3.1.2 Power supply

The PTX Evalboard is powered via a USB-C connection.

If the USB port on the PC side can supply 900mA (i.e a USB-C or USB-3 port) the Evalboard can achieve its maximum output power.

The PTX100R can be operated in two voltage settings (3.7V & 5.4V) which are selected by the switch on the left side of the board.

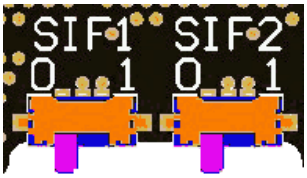
3.1.3 USB Interface

The USB Interface is handled by an FTDI FT231X USB/UART bridge.

Driver installation is automatic on Windows 10.

3.1.4 Interface switching

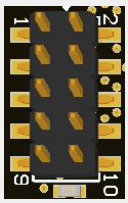
The evaluation board allows to switch between the three interfaces supported by the PTX100R IC using the SIF1 / SIF2 switches:

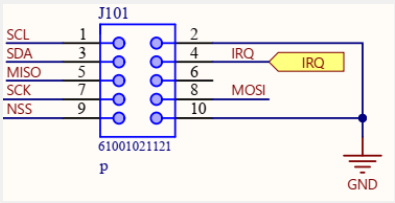


SIF1	SIF2	Interface
0	0	SPI
0	1	UART → USB
1	0	I2C

For the PTX IOT demo app, the SPI interface must be selected.

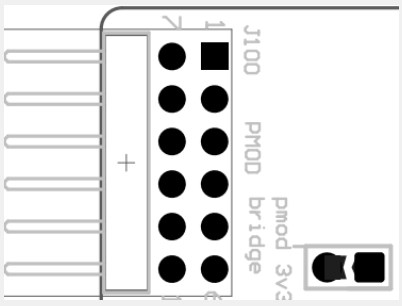
SPI and I2C are connected to a 2x5pin header on the bottom side of the board:

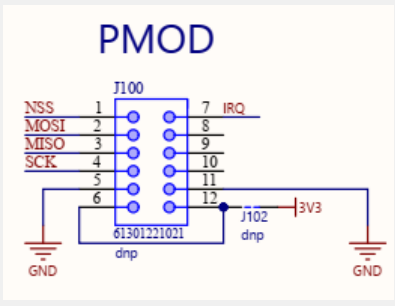




3.1.5 PMOD

SPI is also available via a PMOD 2x6pin connector. The PMOD connector allows connecting to a multitude of MCU demo boards.





The 3.3V supply for the host MCU can also be provided by the Demo Board if the jumper (PMOD 3v3 bridge) is placed

4 PTX IoT Reader library

PTX IoT Reader library provides an API for a set of functions:

- initialize the IOTRD API and NSC Stack
- initialize the PTX100R chip
- select a specific card in case multiple cards and/or protocols were discovered
- retrieve card details like technical and/or activation parameters etc.
- exchange RF-data and -bitstreams
- stop RF-communication

4.1 IoT-Reader API Description

This chapter contains an overview of the functions provided by the IoT-Reader API.

Note: A detailed description of all functions including parameters and types can be found in the file ptx_IOT_READER.h.

4.1.1 ptxIoTRd_Init

Declaration

```
1  ptxStatus_t ptxIoTRd_Init( ptxIoTRd_t iotRd, ptxIoTRd_InitPars_t
   initParams);
```

Description

Initialize software and hardware components for IoT-Reader operation. This function has to be called before any other API functions. It performs software initialization and configuration for PTX100R.

Input Parameters

Name	Description
iotRd	Pointer to stack component (to be allocated by user).
initParams	NSC initialization parameters.

Return Value

Status Success or failure, refer to ptxStatus_t for details.

4.1.2 ptxIoTRd_Update_ChipConfig

Declaration

```
1  ptxStatus_t ptxIoTRd_Update_ChipConfig ( ptxIoTRd_t iotRd, uint8_t
   nrConfigs, ptxIoTRd_ChipConfig_t configParams);
```

Description

Updates the RF- and System-Configuration parameters of the NFC hardware. This function allows to change RF- and System-Configuration parameters at runtime.

Input Parameters

Name	Description
iotRd	Pointer to stack component.
nrConfigs	Number of RF-/System-configurations to set
configParams	Pointer to n-configuration parameters sets

Return Value

Status Success or failure, refer to ptxStatus_t for details.

4.1.3 ptxIoTRd_Get_Revision_Info

Declaration

```
1  ptxStatus_t ptxIoTRd_Get_Revision_Info ( ptxIoTRd_t iotRd,
   ptxIoTRd_RevisionType_t revisionType, uint32_t revisionInfo);
```

Description

Get various revisions of system (C-Stack, DFY-Code/-Toochain, Chip-ID, Local changes etc.

Input Parameters

Name	Description
iotRd	Pointer to stack component (to be allocated by user).
revisionType	Revision type
revisionInfo	Pointer to variable holding revision information

Return Value

Status Success or failure, refer to ptxStatus_t for details.

4.1.4 ptxIoTRd_Initiate_Discovery

Declaration

```
1  ptxStatus_t ptxIoTRd_Initiate_Discovery ( ptxIoTRd_t iotRd,
   ptxIoTRd_DiscConfig_t discConfig);
```

Description

This function starts the RF-Discovery procedure as defined in the NFC-Forum.

Input Parameters

Name	Description
iotRd	Pointer to stack component.
discover	config Pointer to RF-Discovery structure (if set to NULL-default values will be used).

Return Value

Status Success or failure, refer to ptxStatus_t for details.

4.1.5 ptxIoTRd_Get_Card_Registry

Declaration

```
1  ptxStatus_t ptxIoTRd_Get_Card_Registry ( ptxIoTRd_t iotRd,
   ptxIoTRd_CardRegistry_t cardRegistry);
```

Description

Access the internal card registry. This function can be used to access the internal card registry to retrieve a cards detailed information.

Input Parameters

Name	Description
iotRd	Pointer to stack component.
cardRegistry	Pointer to pointer to keep reference to card registry

Return Values

Status Success or failure, refer to ptxStatus_t for details.

4.1.6 ptxIoTRd_Activate_Card

Declaration

```
1  ptxStatus_t ptxIoTRd_Activate_Card ( ptxIoTRd_t iotRd,
   ptxIoTRd_CardParams_t cardParams, ptxIoTRd_CardProtocol_t protocol);
```

Description

Selects / Activates a given card in case of multiple available cards

Input Parameters

Name	Description
iotRd	Pointer to stack component.
cardParams	Pointer to card (within registry) to select / activate

Name	Description
protocol	RF-protocol to activate

Return Value

Status Success or failure, refer to ptxStatus_t for details.

4.1.7 ptxIoTRd_Data_Exchange

Declaration

```
1  ptxStatus_t ptxIoTRd_Data_Exchange ( ptxIoTRd_t iotRd, uint8_t tx,
   uint32_t txLength, uint8_t rx, uint32_t rxLength, uint32_t msAppTimeout);
```

Description

Protocol-based or raw RF data exchange

Input Parameters

Name	Description
iotRd	Pointer to stack component.
tx	Pointer to buffer holding data to send
txLength	Length of data to send
rx	Pointer to buffer holding received data
rxLength	Size of buffer holding received data / length of received data
msAppTimeout	Application timeout

Return Values

Status Success or failure, refer to ptxStatus_t for details.

4.1.8 ptxIoTRd_Bits_Exchange_Mode

Declaration

```
1  ptxStatus_t ptxIoTRd_Bits_Exchange_Mode ( ptxIoTRd_t iotRd, uint8_t
   enable);
```

Description

Enables/Disables the bit-exchange mode required to call ptxIoTRd_Bits_Exchange.

Input Parameters

Name	Description
iotRd	Pointer to stack component.
enable	Enable/Disable flag

Return Values

Status Success or failure, refer to ptxStatus_t for details.

4.1.9 ptxIoTRd_Bits_Exchange

Declaration

```
1  ptxStatus_t ptxIoTRd_Bits_Exchange ( ptxIoTRd_t iotRd, uint8_t tx, uint8_t
   txPar, uint32_t txLength, uint8_t rx, uint8_t rxPar, uint32_t rxLength,
   uint32_t numTotBits, uint 32 _t msAppTimeout);
```

Description

Exchanges a bitstream based on NFC-A technology

Input Parameters

Name	Description
iotRd	Pointer to stack component.
tx	Pointer to buffer holding data bytes to send
txPar	Pointer to buffer holding parity bits to send
txLength	Length of tx- and txPar-buffers
rx	Pointer to buffer holding received bytes
rxPar	Pointer to buffer holding received parity bits
rxLength	Length of received bytes / parity bits
numTotBits	Total number of received bits
msAppTimeout	Application timeout

Return Value

Status of operation Success or failure, refer to ptxStatus_t for details.

4.1.10 ptxIoTRd_RF_PresenceCheck

Declaration

```
1  ptxStatus_t ptxIoTRd_RF_PresenceCheck ( ptxIoTRd_t iotRd,
   ptxIoTRd_CheckPresType_t presCheckType);
```

Description

Executes a presence check method on ISO-DEP cards or NFC-DEP targets

Input Parameters

Name	Description
iotRd	Pointer to stack component.
presCheckType	Presence-check method type.

Return Value

Status of operation

4.1.11 ptxIoTRd_T5T_IsolatedEoF

Declaration

```
1  ptxStatus_t ptxIoTRd_T5T_IsolatedEoF ( ptxIoTRd_t iotRd, uint8_t rx,
   uint32_t rxLength, uint32_t msAppTimeout);
```

Description

Sends an EoF-packet according to T5T protocol

Input Parameters

Name	Description
iotRd	Pointer to stack component.
rx	Pointer to buffer holding received bytes
rxLength	Size of buffer holding received data / length of received data
msAppTimeout	Application timeout

Return Value

Status of operation

4.1.12 ptxIoTRd_T3T_SENSFRequest

Declaration

```
1  ptxStatus_t ptxIoTRd_T5T_IsolatedEoF ( ptxIoTRd_t iotRd, uint16_t
   systemCode, uint8_t requestCode, uint8_t tsn, uint8_t rx, uint32_t
   rxLength, uint32_t msAppTimeout);
```

Description

Sends a SENSF_REQ-packet according to T3T protocol.

Input Parameters

Name	Description
iotRd	Pointer to stack component.
systemCode	T3T System-code
requestCode	T3T Request-code
tsn	T3T Number of timeslot(s)
rx	Pointer to buffer holding received bytes
rxLength	Size of buffer holding received data / length of received data
msAppTimeout	Application timeout

Return Value

Status of operation

4.1.13 ptxIoTRd_Reader_Deactivation

Declaration

```
1  ptxStatus_t ptxIoTRd_Reader_Deactivation ( ptxIoTRd_t iotRd, uint8_t
   deactivationType);
```

Description

Stops any finished RF-communication and deactivates the reader and / or the remove device.

Input Parameters

Name	Description
iotRd	Pointer to stack component.
deactivationType	Type of deactivation (IDLE, DISCOVERY, Sleep)

Return Value

Status of operation

4.1.14 ptxIoTRd_Update_ChipConfig

Declaration

```
1  ptxStatus_t ptxIoTRd_Update_ChipConfig ( ptxIoTRd_t iotRd, uint8_t
   nrConfigs, ptxIoTRd_ChipConfig_t configParams);
```

Description

Updates RF- and System-parameters at runtime.

Input Parameters

Name	Description
iotRd	Pointer to stack component.
nrConfigs	Number of RF-/System-configurations to set
configParams	Pointer to n-configuration parameters sets

Return Value

Status of operation

4.1.15 ptxIoTRd_Set_Power_Mode

Declaration

```
1  ptxStatus_t ptxIoTRd_Set_Power_Mode ( ptxIoTRd_t iotRd, uint8_t
   newPowerMode);
```

Description

Puts chip into stand-by or wakes it up from stand-by

Input Parameters

Name	Description
iotRd	Pointer to stack component.
newPowerMode	Type of stand-by operation

Return Value

Status of operation

4.1.16 ptxIoTRd_Get_System_Info

Declaration

```
1  ptxStatus_t ptxIoTRd_Get_System_Info ( ptxIoTRd_t iotRd,
   ptxIoTRd_SysInfoType_t infoType, uint8_t infoBuffer, uint8_t
   infoBufferLength);
```

Description

Enables / Disables immediate writing to a given log file

Input Parameters

Name	Description
iotRd	Pointer to stack component
infoType	Information identifier
infoBuffer	Buffer to store information
infoBufferLength	Length of information

Return Value

Status of operation

4.1.17 ptxIoTRd_SWReset

Declaration

```
1  ptxStatus_t ptxIoTRd_SWReset ( ptxIoTRd_t iotRd);
```

Description

Performs a soft-reset of the PTX100R

Input Parameters

Name	Description
iotRd	Pointer to stack component.

Return Value

Status Success or failure, refer to ptxStatus_t for details.

4.1.18 ptxIoTRd_Deinit

Declaration

```
1  ptxStatus_t ptxIoTRd_Deinit ( ptxIoTRd_t iotRd);
```

Description

Close the IoT-Reader Component. This function closes the IOT and releases the resources used. It must be called as the last function before the stop of the library usage.

Input Parameters

Name	Description
iotRd	Pointer to stack component.

Return Value

Status Success or failure, refer to ptxStatus_t for details.

4.1.19 ptxIoTRd_Get_Status_Info

Declaration

```
short ptxIoTRd_Status_Info ( ptxIoTRd_t iotRd, ptxIoTRd_StatusType_t
statusType uint8_t statusInfo);
```

Description

Retrieves current operating state of chip

Input Parameters

Name	Description
iotRd	Pointer to stack component
statusType	Status type identifier
systemState	Pointer to variable holding
system	state

Return Value

Status of operation

5 PTX IoT Reader library integration

The precompiled libraries for the supported architectures can be found in the `PtxIot/lib` subfolder. The appropriate one of them for the particular MCU must be fed to the linker during build time.

Even without using an IDE, it is very easy to work with the library. The following prerequisites must be met:

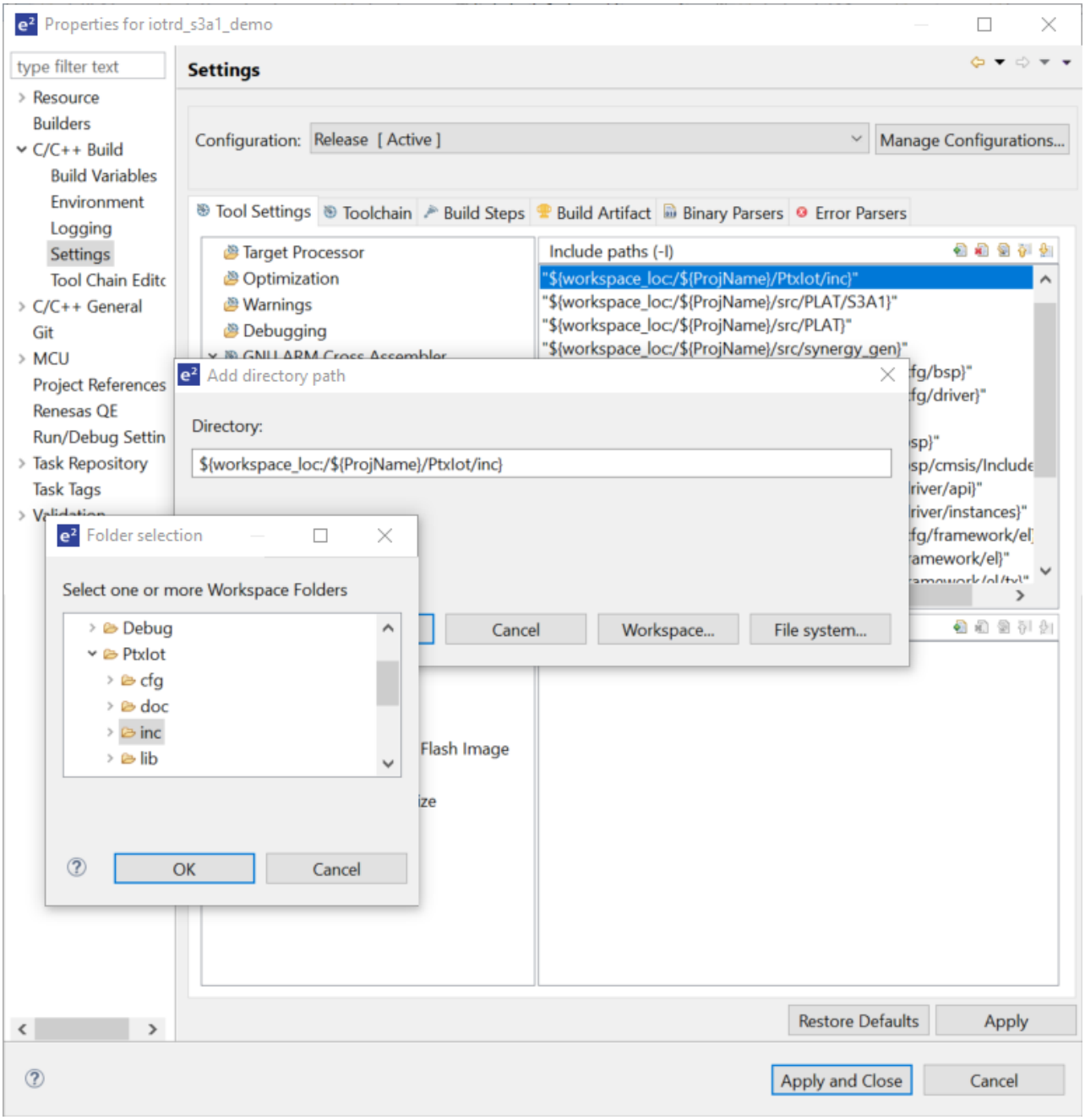
- the "C" or "C++" **compiler** shall be able to find the include files in the `inc` folder. (Use the *-I* option)
- the **linker** shall be able to find the library file. (Use the *-L* option)
- the **linker** needs to be told explicitly to use the library. (Use the *-l* option)

With these rules kept in mind, it is easy to employ the library in any codebase with any build system.

This chapter explains the steps to add the library to an **e² studio** project.

5.1 Add include path

For the compiler to find the header (.h) files containing the API functions provided by Panthronics AG, the folder `PtxIot/inc` needs to be added to the list of user defined include directories. This can be done by navigating to **Project menu ⟶ Properties ⟶ C/C++ Build ⟶ Settings ⟶ Tool Settings ⟶ GNU ARM Cross Compiler ⟶ Includes**. Clicking on the **Add...** button on the right side of the small toolbar and using **Workspace** button in the popup window, we can navigate to the folder and click **OK**.

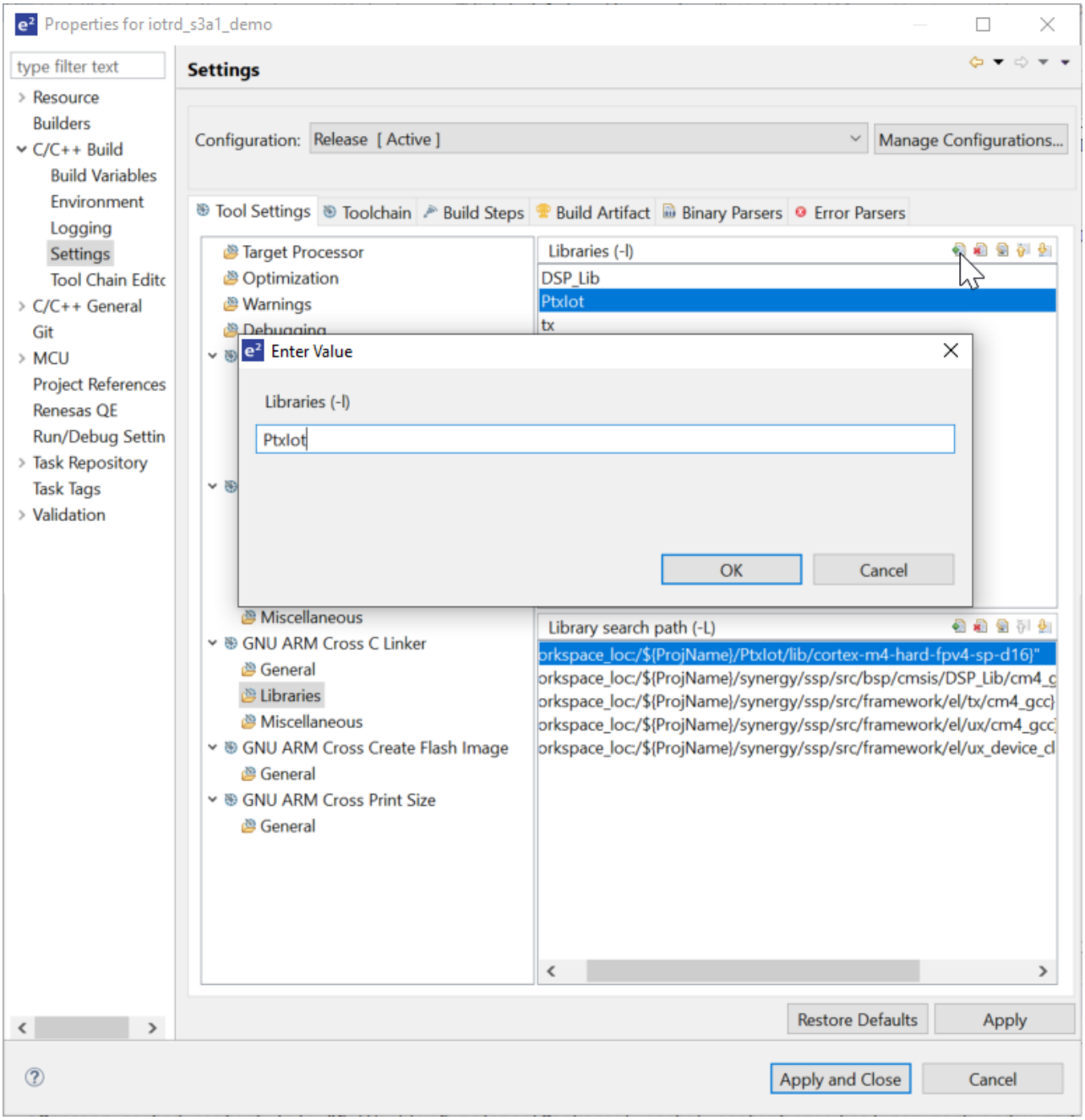


5.2 Adding library file

There is an individual subdirectory in the `PtxIot/lib` folder for each supported MCU architecture, which contains the library file `libPtxIot.a`.

The library file, like the header files, also needs to be found by the linker. Similarly to the include folders' settings, the proper library folder must be added to the *Library search path* list (see bottom right pane on image).

Moreover we need to add the **PtxIot** to the **Libraries** list. This will be rendered into *-lPtxIot* command line switch during linker invocation, which tells the linker to use the file `libPtxIot.a`.



5.3 Platform-specific implementation

As the library is common for all implementations, it must not use any hardware dependent resources. This is where the PLAT module comes into play by defining the interface functions to be implemented by the user. The required function declarations can be found in ptxPLAT.h. These functions must be individually implemented for the actual hardware.

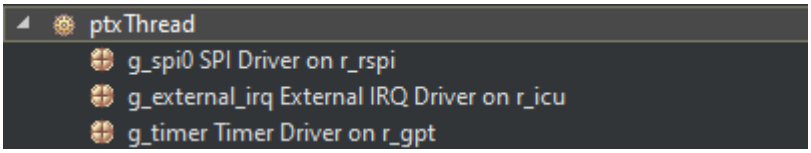
The ptxPLAT.c code contains the basic low-level functions, that the library will use for allocating memory area for objects, initializing and controlling timers, do communication across SPI, etc.

The **TB-S3A1** platform-specific implementation can be found in ptxPLAT.c, but thank to Renesas Synergy platform API, the implementation is compatible for any different Synergy MCU. In such case the device pin assignment might be the only configuration to be adjusted.

5.4 Project configuration in e² studio

To be able to implement PTX communication over the PMOD B connector, the following peripherals should be configured:

- SPI
- External IRQ
- Timer IRQ



5.4.1 SPI

- SPI channel 0
- SPI bitrate 10000000
- SPI callback NULL

Please note, the current implementation does not use IRQ for SPI.

◢ Common	
Parameter Checking	Default (BSP)
◢ Module g_spi0 SPI Driver on r_rspi	
Name	g_spi0
Channel	0
Operating Mode	Master
Clock Phase	Data sampling on odd edge, data variation on even edge(CPHA=0)
Clock Polarity	Low when idle
Mode Fault Error	Disable
Bit Order	MSB First
Bitrate	10000000
Callback	NULL
SPI Mode	SPI Operation
Slave Select Polarity(SSL)	Active Low
Select Loopback1	Normal
Select Loopback2	Normal
Enable MOSI Idle	Disable
MOSI Idle State	MOSI Low
Enable Parity	Disable
Parity Mode	Parity Odd
Select SSL Level After Transfer	SSL Level Do Not Keep
Clock Delay Enable	Clock Delay Disable
Clock Delay Count	Clock Delay 1 RSPCK
SSL Negation Delay Enable	Negation Delay Disable
Negation Delay Count	Negation Delay 1 RSPCK
Next Access Delay Enable	Next Access Delay Disable
Next Access Delay Count	Next Access Delay 1 RSPCK
Receive Interrupt Priority	Priority 5
Transmit Interrupt Priority	Priority 5
Transmit End Interrupt Priority	Priority 5
Error Interrupt Priority	Priority 5
Byte Swap(Only for S5 series MCU's)	Disable

SPI channel should also be enabled from PINS/Peripherals/Connectivity:SPI/ SPI0

5.4.2 External IRQ

- IRQ trigger - Rising
- IRQ callback ptx_IRQ

Property	Value
◢ Common	
Parameter Checking	Default (BSP)
◢ Module g_external_irq External IRQ Driver on r_icu	
Name	g_external_irq
Channel	1
Trigger	Rising
Digital Filtering	Disabled
Digital Filtering Sample Clock (Only valid when Digital Filte	PCLK / 64
Interrupt enabled after initialization	True
Callback	ptxIRQ
Pin Interrupt Priority	Priority 12

5.4.3 Timer IRQ

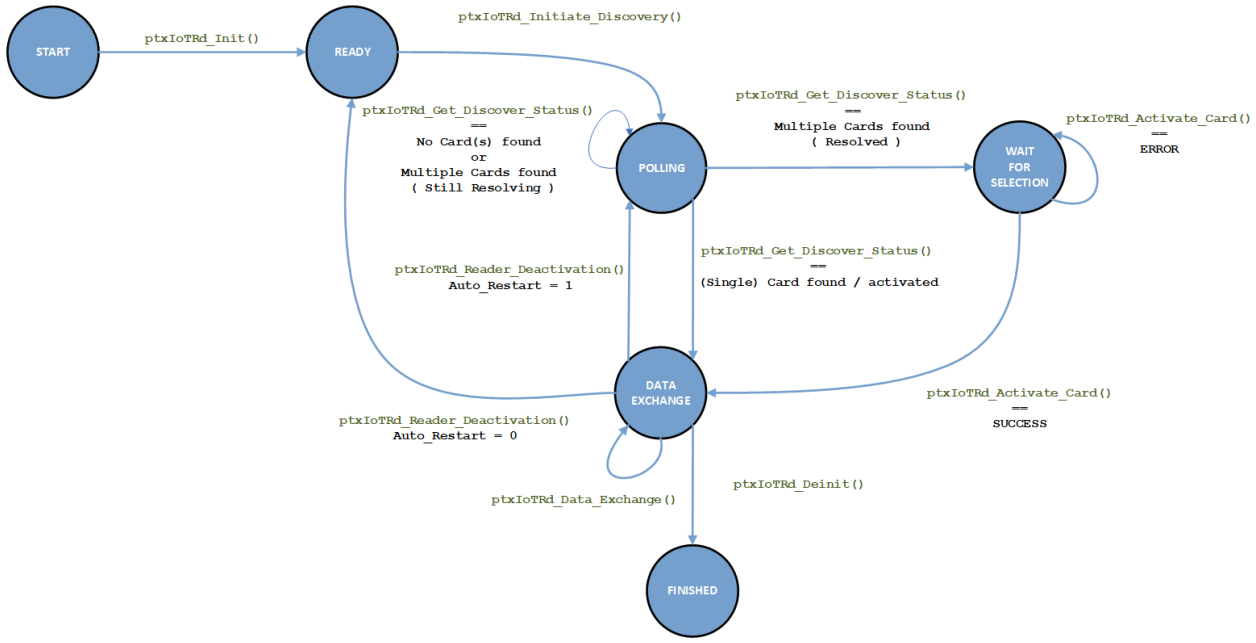
- Timer period 10 ms
- Timer callback ptxTimerIRQ

Property	Value
◢ Common	
Parameter Checking	Default (BSP)
◢ Module g_timer Timer Driver on r_gpt	
Name	g_timer
Channel	0
Mode	Periodic
Duty Cycle Range (only applicable in PWM mode)	Shortest: 2 PCLK, Longest: (Period - 1) PCLK
Period Value	10
Period Unit	Milliseconds
Duty Cycle Value	50
Duty Cycle Unit	Unit Raw Counts
Auto Start	True
GTIOCA Output Enabled	False
GTIOCA Stop Level	Pin Level Low
GTIOCB Output Enabled	False
GTIOCB Stop Level	Pin Level Low
Callback	ptxTimerIRQ
Overflow Interrupt Priority	Priority 12

6 PTX IoT demo application

PTX IoT demo application is meant to demonstrate the performance of the PTX100R for IoT applications, offering full support for all technologies (A, B, F, V).

This application implements all steps required to initialize the PTX100R, to discover, activate, communicate and to deactivate a tag as described in the chart below:



6.1 Environment

In this document we are going to use the **e² studio 7.8** as our code editor and our target is the **TB-S3A1** evaluation board with an ARM cortex-m4 microcontroller having a hardware floating point unit.

The current demo package contains precompiled libraries for ARM Cortex-M0+ and Cortex-M4 architectures. To use an MCU of a currently unsupported architecture, another precompiled library is required, which is built for exactly the particular hardware. For more information, please contact our support.

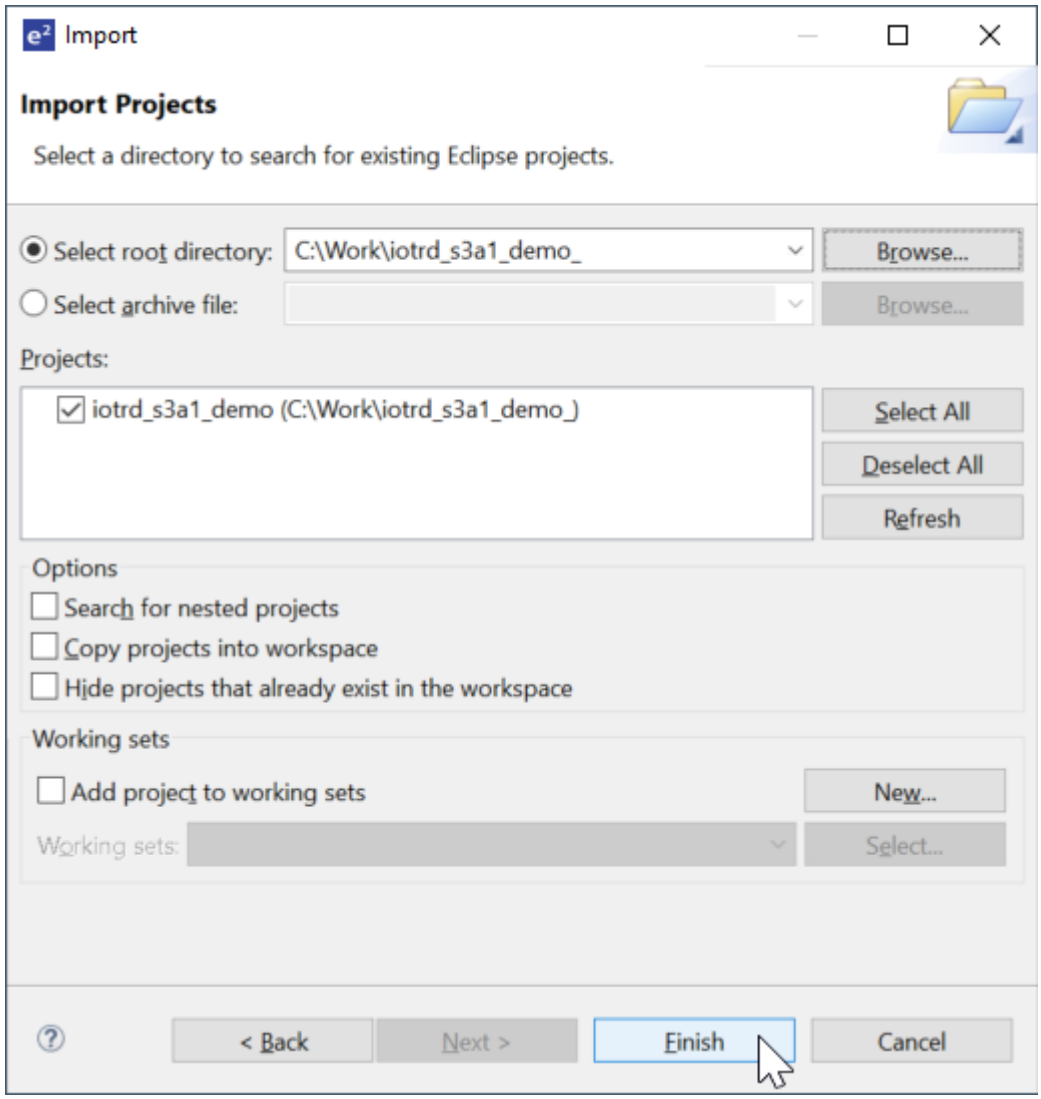
6.2 Project structure

File/Folder	Description
PtxIot/	PTX IoTRD library
PtxIot/ptx_IOT_READER.h	IoT demo app API available with the PTX IoTRD library
src/PLAT/	HAL configuration for communication: SPI, Timer, and IRQ
src/consoleThread_entry.c	Thread containing the implementation of Console Framework over USB

File/Folder	Description
src/ptxThread_entry.c	Thread containing the implementation of the PTX IOT demo application

6.3 Importing the demo project

To get started with the demo project quickly, it needs to be imported in e² studio. This can be done by **File ⟶ Import... ⟶ Existing Projects into Workspace** and selecting the folder the archive had been extracted to.



After importing, the project is ready for compiling. With selecting **Project ⟶ Build All** from the menu, the compiling process will start and a similar table will summarize the memory usage after a successful build:

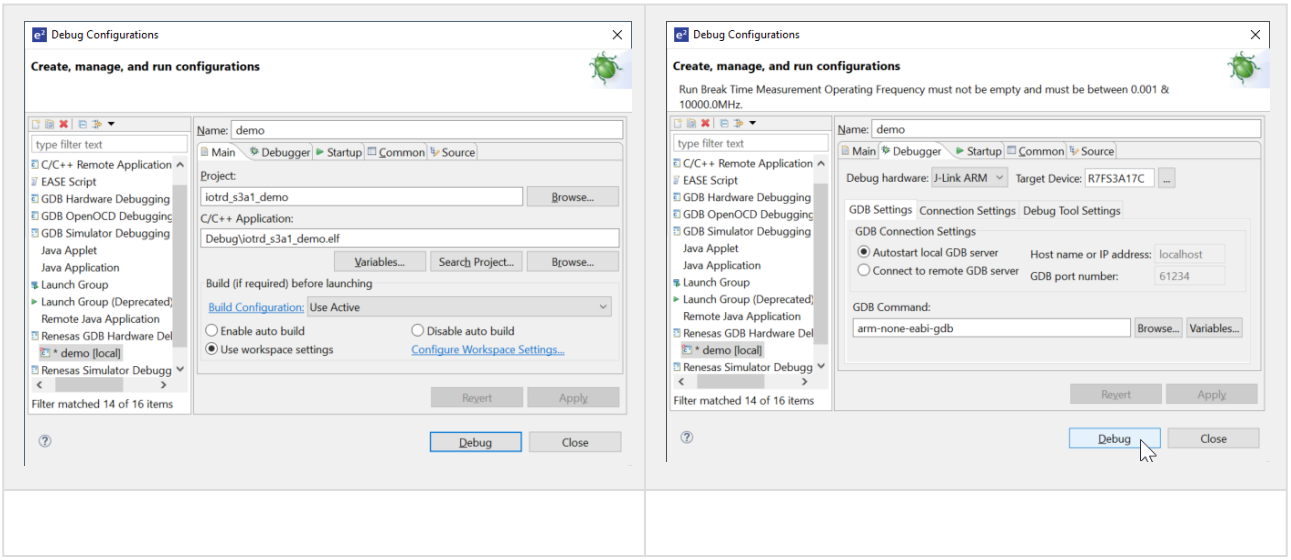
```
arm-none-eabi-size --format=berkeley "iotrd_s3a1_demo.elf"
   text    data     bss     dec     hex filename
  97120     635   40608  138363   21c7b iotrd_s3a1_demo.elf
```

6.4 Running the PTX IOT reader demo application

Once the project is compiled, it needs to be flashed into the MCU and run.

6.4.1 Preparing the debug configuration

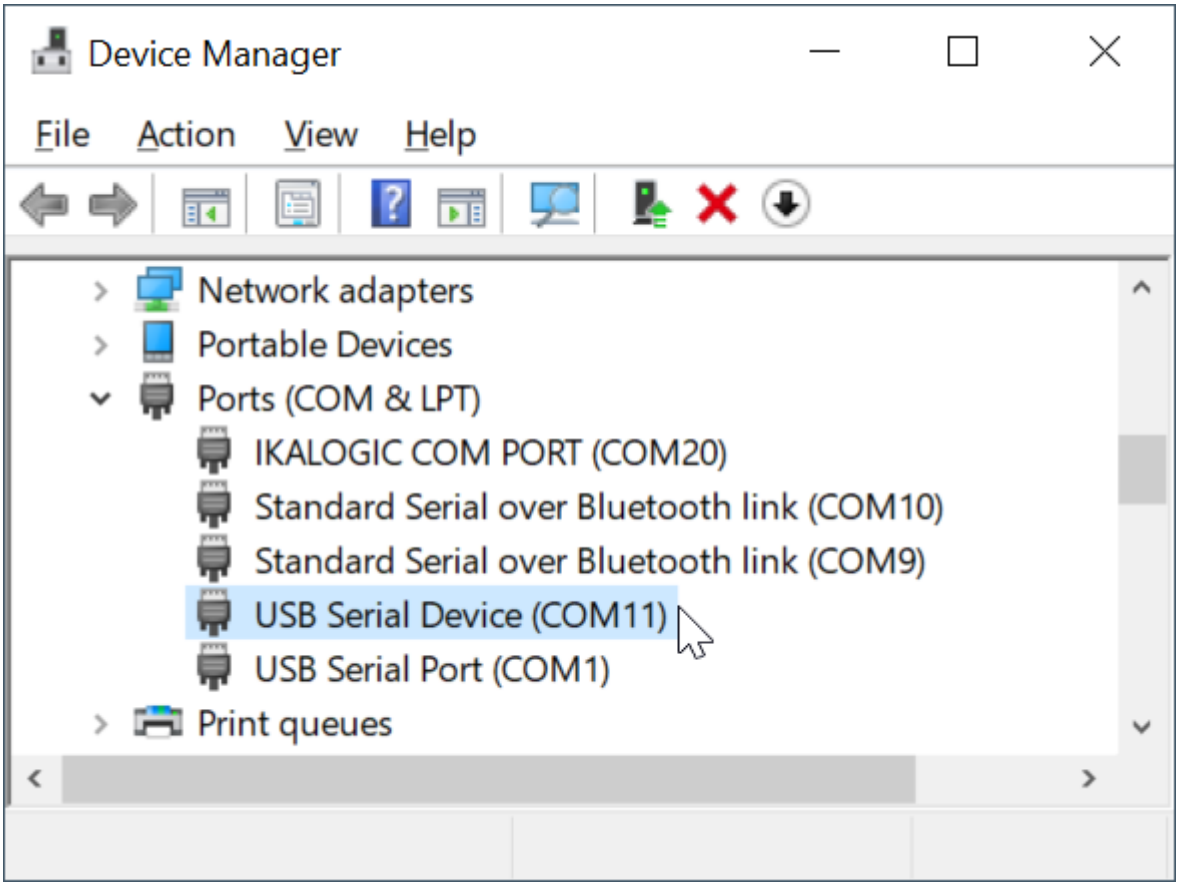
Since the debug configuration is not part of the project, it cannot be imported, therefore a new one needs to be created for the workspace used. In the **Run ⟶ Debug Configurations...** menu. The TB-S3A1 evaluation board has a **SEGGER J-Link** on-board debug probe, which is supported by the **Renesas GDB Hardware Debugging** driver, therefore a new configuration needs to be created and configured according to the following images:



Note: The **Target Device** needs to be set to the actual MCU used, it will not be taken over from the project.

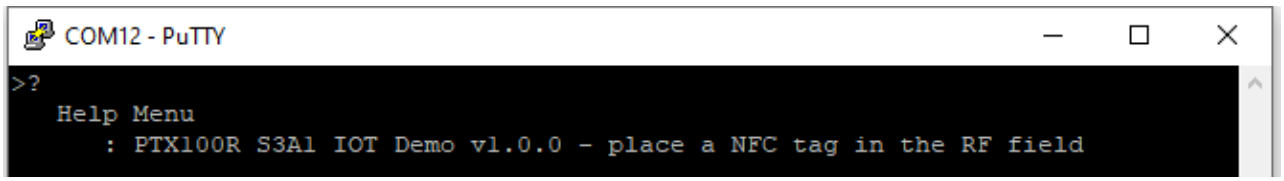
6.4.2 Flashing and running/debugging

Finally, the debug session can be started by pressing the **Debug** button. As soon as the firmware is running on the board, a new **USB Serial Device** will be discovered by the operating system.

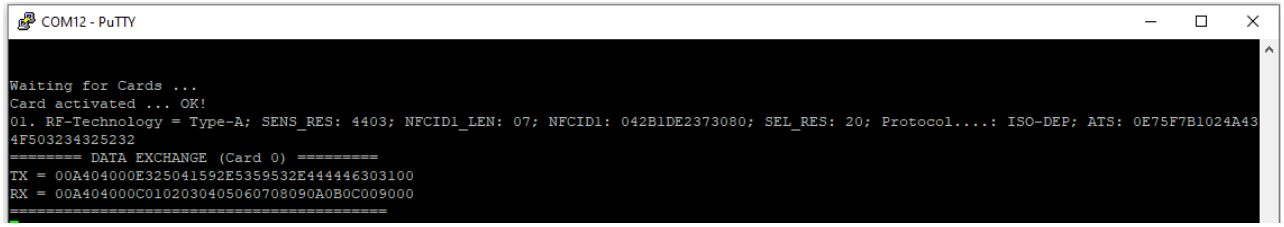


A new connection can be established using PuTTY (or any other app) by connecting to this serial port using a baud rate of 115200.

If **?** is sent to the console, the following message should appear



If an NFC card is present in the field, the details will be read and data exchange will be triggered as shown in the below picture.



Copyrights & Disclaimer

Legal Notice - Purchase of Panthronics ICs with MIFARE Classic® compatibility:

PTX100R IC offers modes to be compatible with MIFARE Classic® RFID tags, allowing to build MIFARE Classic® compatible reader systems. MIFARE® and MIFARE Classic® are trademarks of NXP B.V., High Tech Campus 60 NL-5656 AG EINDHOVEN, NL. Purchase of Panthronics' MIFARE Classic® compatible products does not provide a license of any NXP rights, in particular does not provide the right to use MIFARE® or MIFARE Classic® as a trademark to brand such systems.