Integrating Mercury C API on BareMetal Platform

1 Introduction

This document describes steps to use C API on BareMetal platforms. All the described steps are not IDE specific. Please go through examples for better understanding.

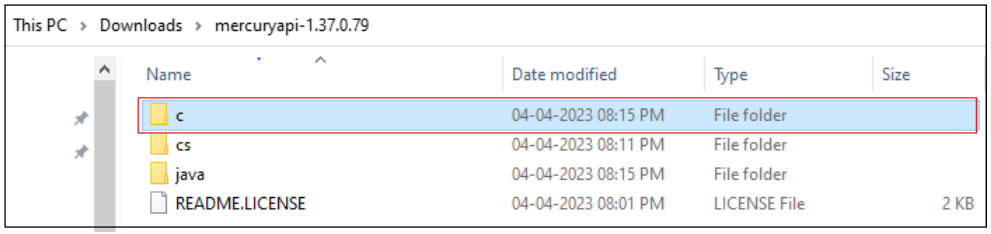
2 Change Log

Rev	Author	Date	Changes
1.3	Titiksha Tyagi	5/2/2024	Added new section for Arduino Nano ESP32
1.2	Satya	10/20/2023	Added Supported baud rates.
1.1	Krishna	8/18/2023	Reorganized a few sections, fixed formatting, and wording.
1.0	Anjali	2/20/2023	Updated Pictures and Description in some sections.

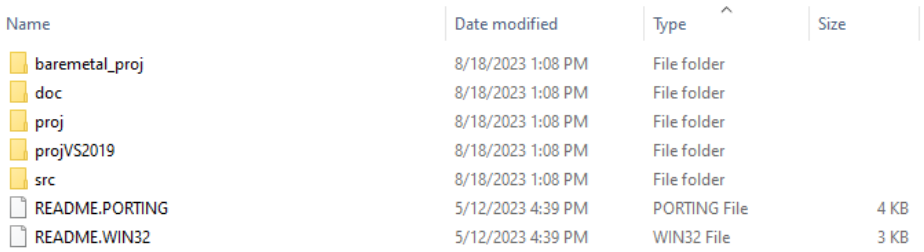
Contents

3 Introduction to Mercury API SDK

Mercury API SDK contains API support in three languages, C, C#, and JAVA. We use C API to perform RFID operations on BareMetal platform.

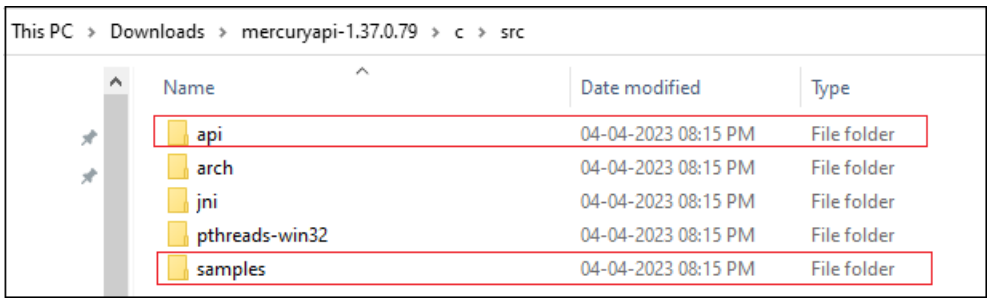


3.1 C API Directory Structure



Note: *baremetal_proj folder contains BareMetal demo projects for Arduino and STM32 platforms.*

The s*rc* directory contains API source files. As shown in below image, *api* and *samples* are main directories. The *api* directory contains all **.c and .h source files**, and *samples* contains example codelets to demonstrate different RFID operations.



4 Primary Steps to Create a Project

Note: *The following steps are not specific to an IDE. It can be used to create BareMetal project on any development platform (like, Keil, Atmel, IAR, etc).*

4.1 Prerequisites

1. Mercury C API

2. IDE to create the project.
3. This section shows the steps to integrate API source files in BareMetal project. The user will have to create UART driver functions to send and receive the data for serial command transmission.

4.2 Create Project

1. Unzip the mercury API SDK.
2. Create new project in IDE with required low-level drivers based on the processor selected, required clock frequency, UART baud rate, etc.
3. The demo project will be generated. Rename the project based on the requirement.
4. Add all API files, file path and sample code as described below.
5. **Source files:**
   Pull all the '.c' files (listed below) from the *<mercuryAPI>\c\src\api* directory to the source directory of the project.
   a. hex_bytes.c
   b. serial_reader.c
   c. serial_reader_l3.c
   d. tm_reader.c
   e. tm_reader_async.c
   f. tmr_param.c
   g. tmr_strerror.c
   h. tmr_utils.c
6. In *<mercuryAPI>\c\src\api* folder, we have provided '**serial_transport.c**' and '**osdep.c**' files for **samD21j18a, arduinomega2560, stm32F103RB,** and **stm32L476VG** controllers. But if user is using different controller, then he needs to create these files for that specific controller.

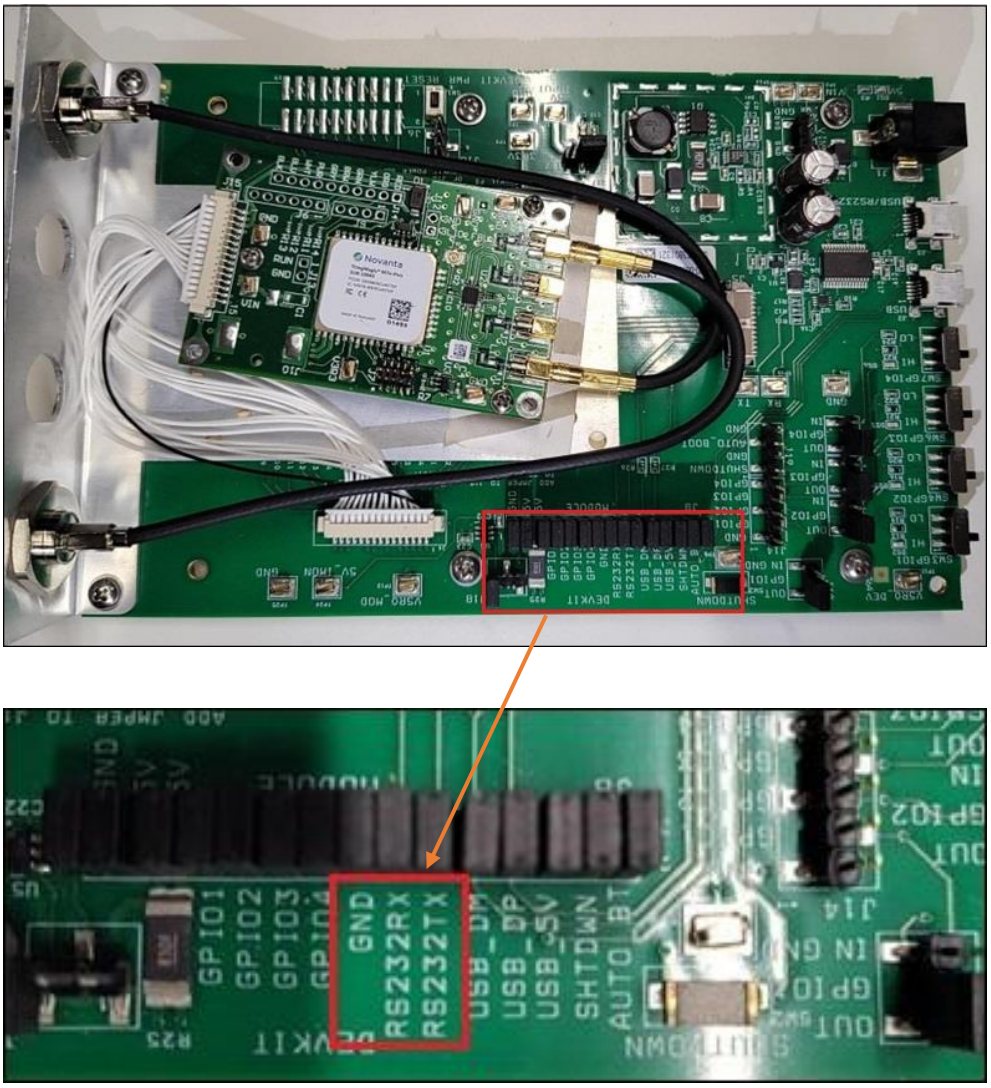
   '**Serial_transport.c**' contains UART driver functions to communicate with the RFID module, and '**osdep.c**; contains time functions.
7. **Main.c:**
   The user needs to ensure that we have one single main() function in their project.

   Please refer '**read.c**' sample codelet from the *<mercuryAPI>\c\src\samples* folder which is used to perform sync read. Note that it already has a main() function.
8. **Additional directory path:**
   Go to Additional directory setting in project properties and add below mentioned path.
   a. Path of *<mercuryAPI>\c\src\api* folder
9. **Pre-processor Macros:**
   Go to pre-processor settings and define **BARE_METAL** macro.
10. Build the project. If the build is successful, the binary generated by the IDE can be flashed on the processor.

5 Hardware Connections

To initiate UART communication, we need to connect TX, RX, V_{CC} and GND pins.

ThingMagic devkit pins are shown in the figure. User will have to figure out UART TX and RX pins of the controller and connect them.



6 Types Of Reads

Basically, there are two types of read supported to read the tag EPC. Apart from this, there are lot of other operations that can be performed, like, standalone tag operation, embedded tag operation, filter, reader stats, etc. But this section describes read types.

6.1 Synchronous Read

Sync read is the timed based read. In this type of read, host will have to provide the read time to the module, and once module receives the command, it will start buffering the tag reports. Till then host will stay in idle state. Once read time expires, module will push all the tag reports to the host.

Use '**read.c**' codelet (*c/src/samples*).

6.2 Asynchronous Read

In this type of read, module starts reading the tag as soon as it receives start read command. Here, module does not buffer the tag reports, instead it sends each report to the host. To stop the read, host must send stop read.

In Mercury API, Async Read (Continuous Read) uses multiple threads to read the tags. Async Read won't work on Bare Metal, since it does not provide multiple threads. A work around to achieve continuous read is to call read synchronously over a loop but this might miss out on reading some tags. To overcome this limitation of calling read synchronously multiple times, it is required to implement the read async behavior on Bare metal platform without any threads using '**readasync.c**' codelet (*c/src/samples*)

Readasync.c

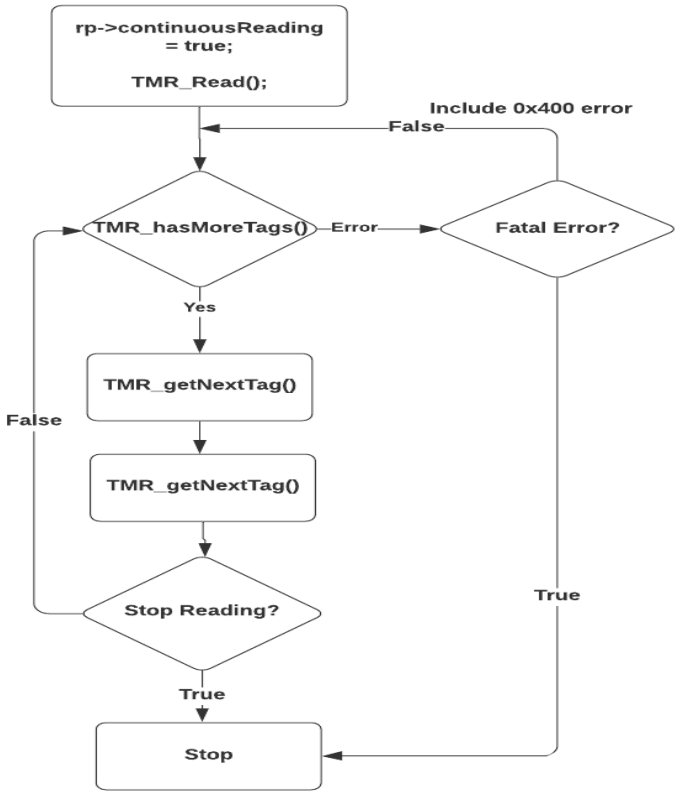
This codelet demonstrates the async read functionality.

TMR_startReading(rp), starts the read. Once read is started, parseSingleThreadedResponse(rp, READ_TIME) function will check if the report is present, then parse and report it.

parseSingleThreadedResponse(rp, READ_TIME)

Here, the second argument READ_TIME specifies the total read time. This can be changed as per the requirement.

Overview of Async Read without Threads



7 Examples

7.1 Arduino Mega 2560 using Arduino IDE

This document describes how to configure, compile C API, and run sample application for Arduino Mega 2560.

7.1.1 Prerequisites

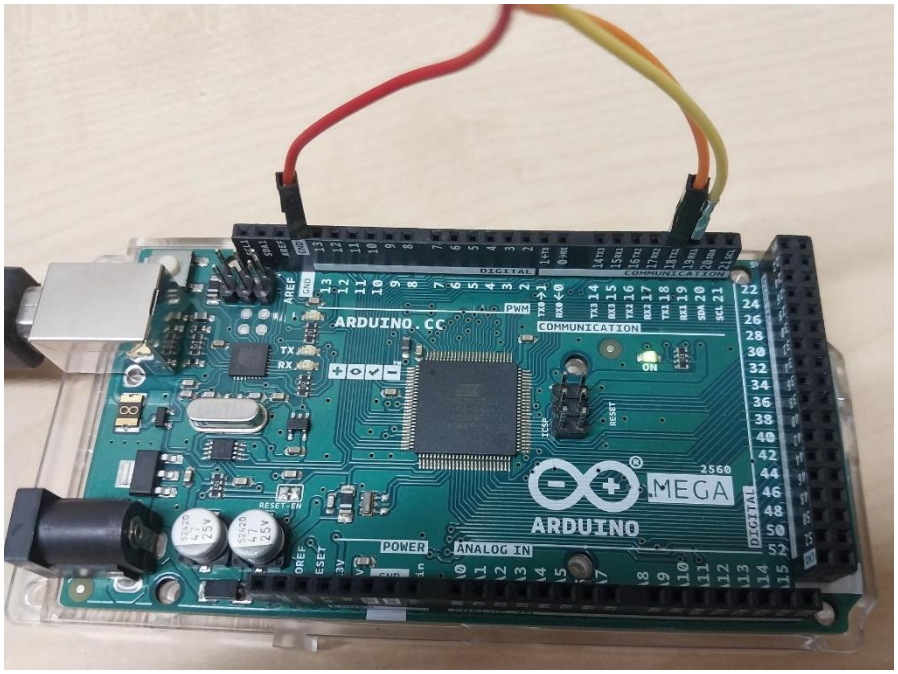
- Download the Arduino IDE from
  - Linux: https://www.arduino.cc/en/Guide/Linux
  - Windows: https://www.arduino.cc/en/Guide/Windows
- Download Mercury API SDK from the Jadak website.

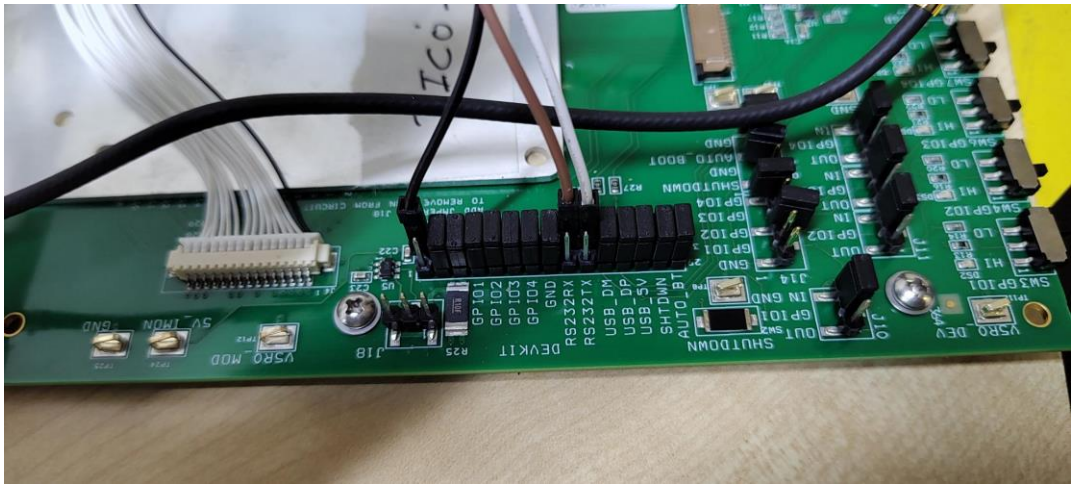
7.1.2 Hardware Connection

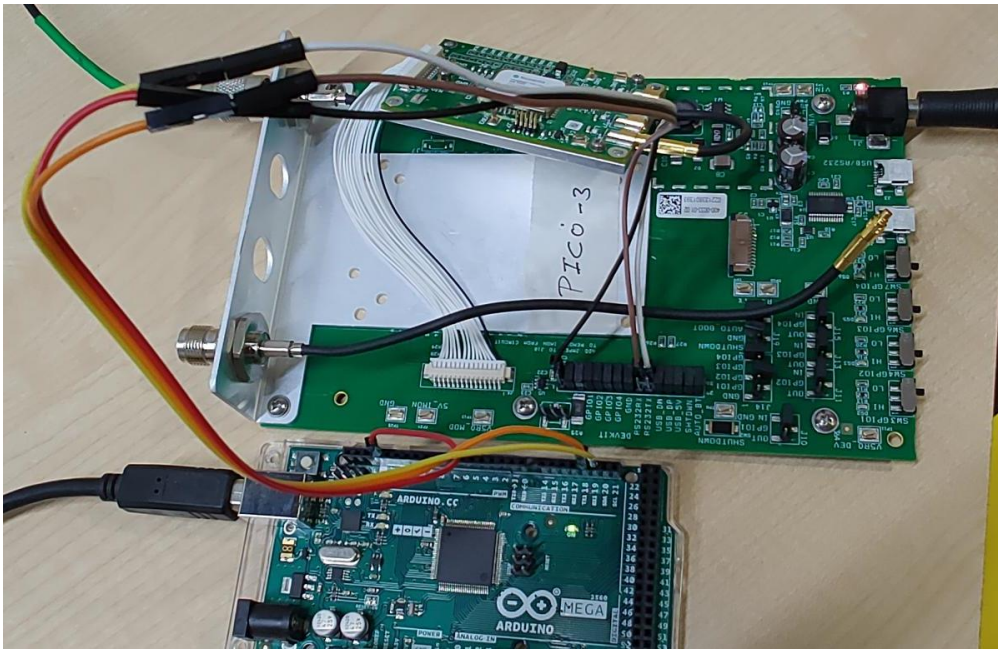
Serial 0 of the Arduino board should be configured for host communication and Serial 1 should be configured for RFID module communication.

Please note that Serial 0 is by default configured on USB port of the board. Connect the serial1 to RFID module as shown below.

1. TX of Micro Devkit <-> Arduino RX1 (Pin-19)
2. RX of Micro Devkit <-> Arduino TX1 (Pin-18)
3. GND <-> Arduino GND


ARDUINO.CC
ARDUINO
MEGA
2560
POWER
ANALOG IN
COMMUNICATION
DIGITAL


SHUTDOWN
DEVKIT
VSRO_MOD
VSRO_DEV


PICO-3
ARDUINO.CC
ARDUINO
MEGA

7.1.3 Project Creation

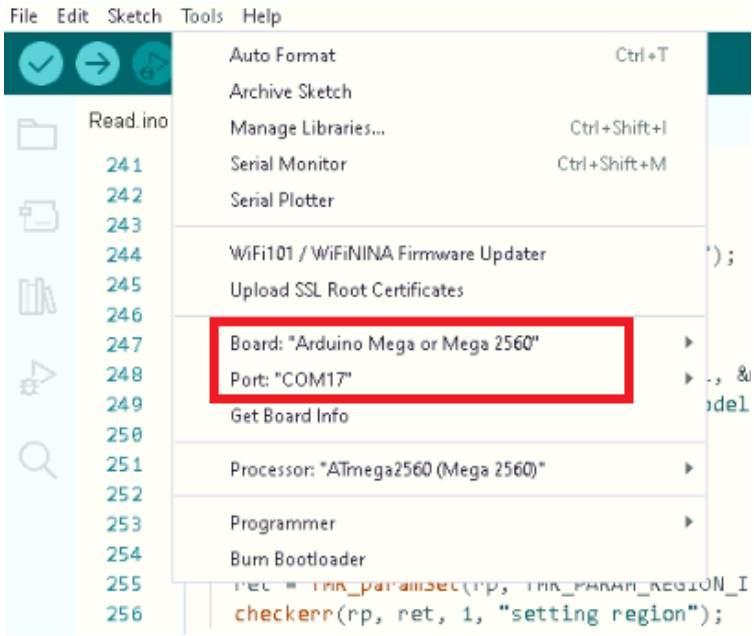
- Download latest (mercuryapi-BILBO-1.37.1.xx) file. And unzip it.
- Create a new folder and rename it to *mercuryapi_src*.
- To add dependency files in Arduino project, copy below files from the folder *<mercuryAPI>\c\src\api* to the newly created folder *mercuryapi_src*.

Hex_bytes.c, Osdep.h, Osdep_arduino.c, Serial_reader.c, Serial_reader_imp.h, Serial_reader_l3.c, Serial_transport_arduino.c, tm_config.h, tm_reader.c, tm_reader.h, tm_reader_async.c, tmr_filter.h, tmr_gen2.h, tmr_gpio.h, tmr_ipx.h, tmr_iso14443a.h, tmr_iso14443b.h, tmr_iso15693.h, tmr_iso180006b.h, tmr_lf125khz.h, tmr_lf134khz.h, tmr_param.c, tmr_params.h, tmr_read_plan.h, tmr_region.h, tmr_serial_reader.h, tmr_serial_transport.h, tmr_status.h, tmr_strerror.c, tmr_tag_auth.h, tmr_tag_data.h, tmr_tag_lock_action.h, tmr_tagop.h, tmr_tag_protocol.h, tmr_types.h, tmr_utils.h, tmr_utils.c.

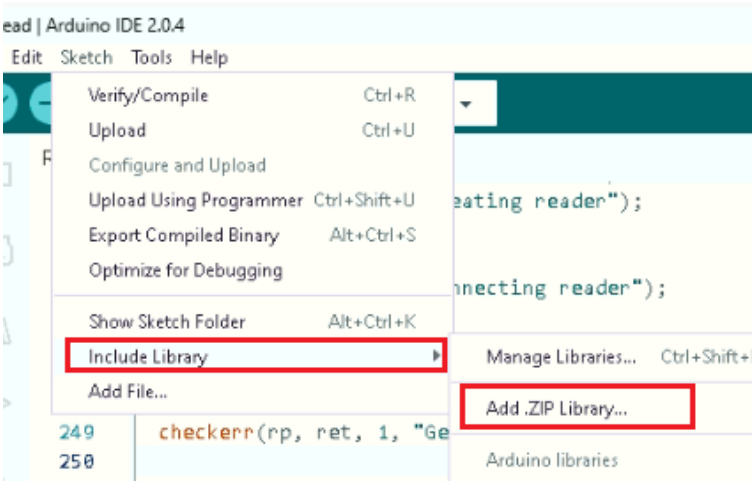
- Open the **tm_config.h** file (from *mercuryapi_src).* Uncomment below line of code and define BARE_METAL macro.

  /** To build API for baremetal platform. */
  //#define **BARE_METAL**

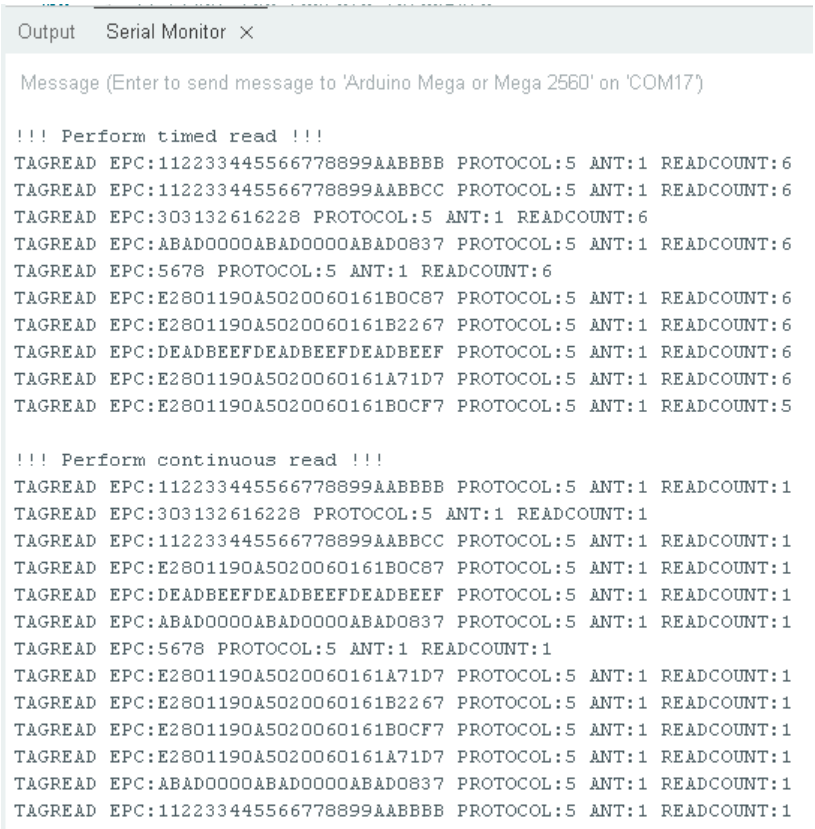
- Rename **Serial_transport_arduino.c** (from *mercuryapi_src)* to **Serial_transport_arduino.cpp.**
- Zip the folder *mercuryapi_src -> mercuryapi_src.zip*
- Open the Arduino IDE and copy the Read.ino sample code from the path*<mercuryAPI>\\c\baremetal_proj\arduino_proj\Arduino_Mega2560\Read.*
- Select board in **tools->Board->Arduino AVR Boards->Arduino Mega or Mega 2560**.
- Select comport in **Tools->Port**.



- Give path of *mercuryapi_src.zip* file in **Sketch->Include Library-> Add .ZIP Library**.



- Build and run.
    - ENABLE_CONTINUOUS_READ macro is used to select either CONTINUOUS READ or TIMED READ. This is set to 1 by default. The CONTINUOUS READ is performed for 500 milliseconds.
    - If ENABLE_CONTINUOUS_READ macro is set to 0, then the module performs TIMED READ for 500 milliseconds.
- Tag reports can be seen in Serial Monitor as shown below.



7.1.4 Arduino Mega 2560 Supported baud rates

Arduino Mega 2560, The supported baud rates are 9600, 19200, 38400, 57600 and 115200.

7.2 SAMD21 Xplained Pro using Microchip Studio (formerly known as Atmel Studio)

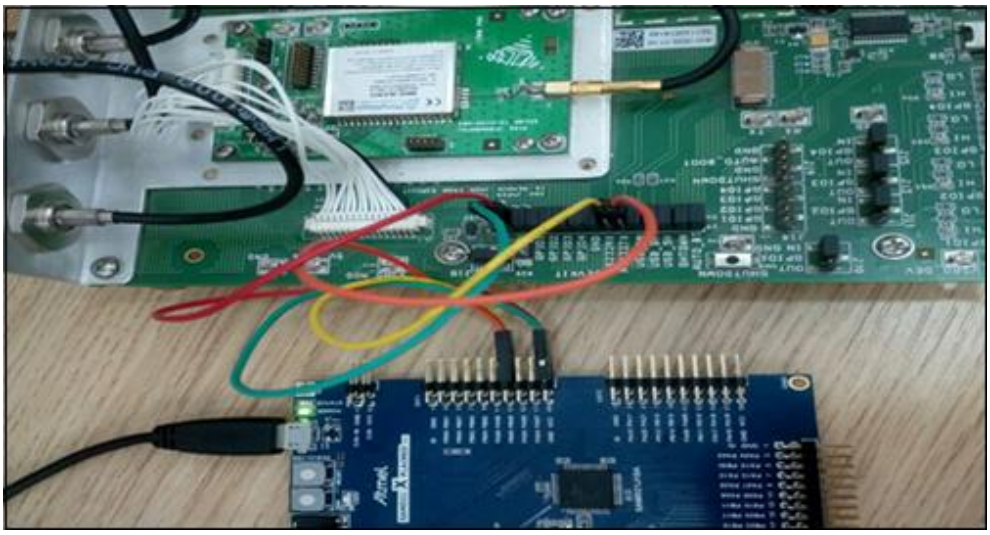
This section explains how Read Async works with Bare Metal. ATMEL SAMD21 Xplained Pro has been used as a Bare metal device and connected to ThingMagic Reader (Nano). Microchip Studio 7 is used for creating the solution.

7.2.1 Hardware Connection

Connection of SAMD21 Xplained Pro and Nano Reader

Connect the Nano reader with SAMD21 Xplained Pro as shown below .and run the application. Following connection needs to be done as shown below picture:

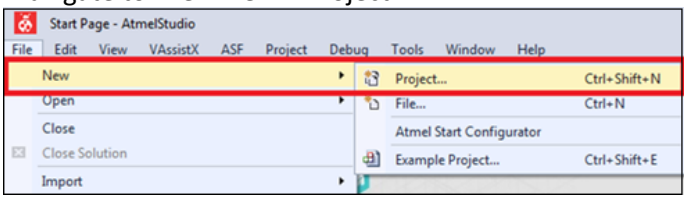
a. Tx of Nano Dev Kit < - > SAMD21 Rx (PB09)
1. Rx of Nano Dev Kit < - > SAMD21 Tx (PB08)
c. Connect 5v and ground appropriately.



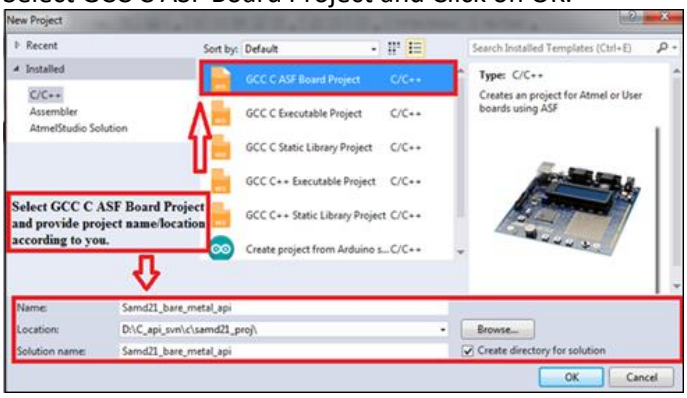
7.2.2 Project Creation

1. Open Atmel Studio
2. Create a new project with the below steps.

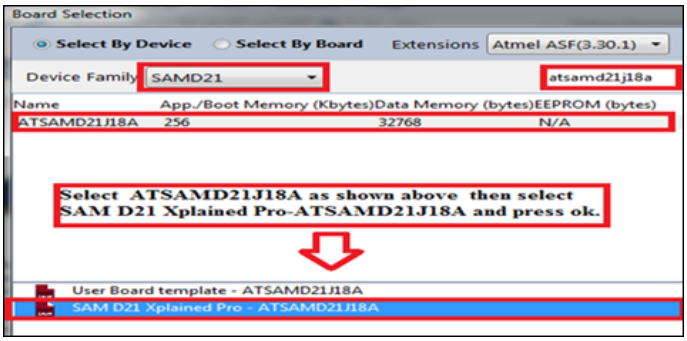
a. Navigate to File > New > Project.



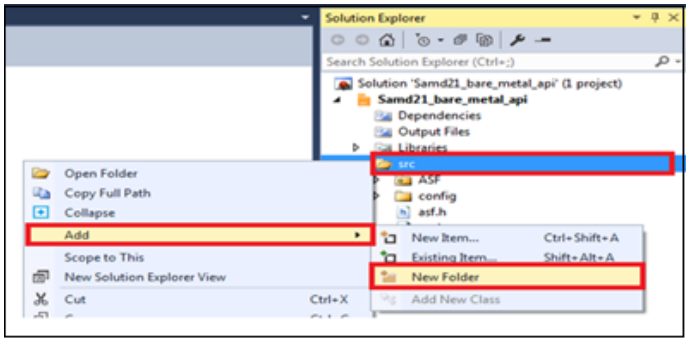
b. Select GCC C ASF Board Project and Click on OK.



c. Under Board Selection, select "**SAMD21**" as Device Family and select **ATSAMD21J18A** board. It will take few moments to create basic project for samd21.



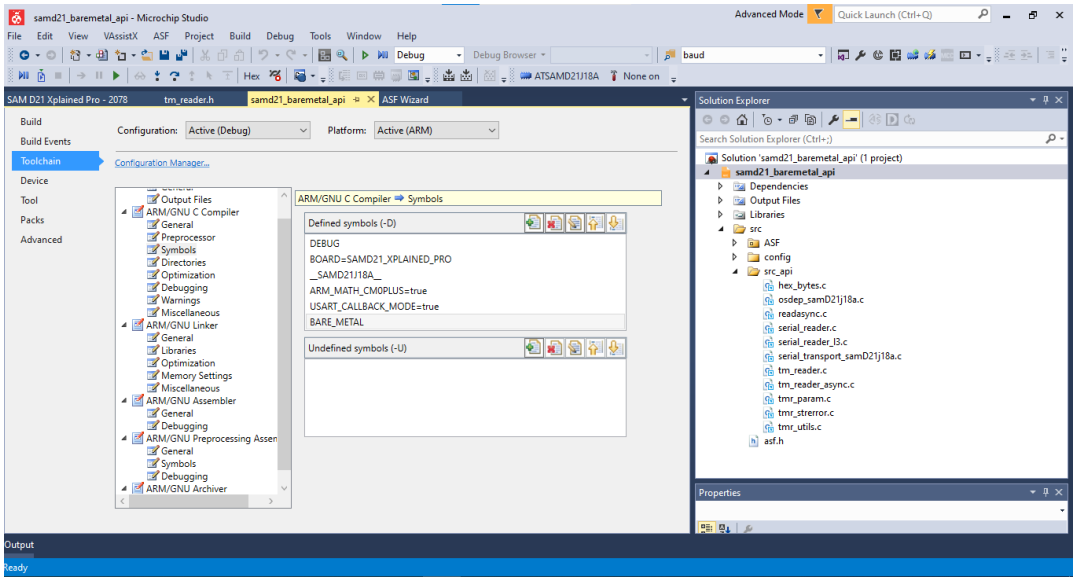
3. In Solution explorer you will get the project distribution.
4. Right click on *src* folder>>>click to add>>>click on New Folder and give name to that folder (let name it *src_api*).



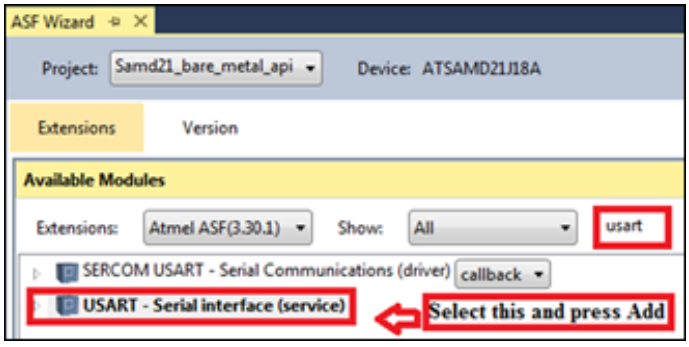
5. Right click on *src_api* folder>>>click to add>>>click on Existing Item and now add all required bare_metal_api files as shown below.

   Add source files from *<mercuryAPI>\c\src\api* location and **readasync.c** from

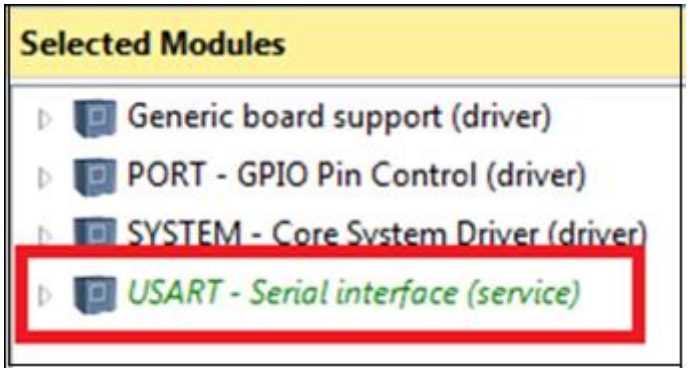
<mercuryAPI>\c\src\samples location.



6. Delete **main.c** file from solution explorer since main function is available in **readasync.c**
7. Now right click on project and select ASF Wizard.



After adding this “Selected Modules” block in ASF Wizard Page will look like this.



8. Press Apply and then ok button.
9. Right click on the project and select properties. Make changes to the property as shown below:

Toolchain > ARM/GNU C Compiler > Symbols : Add BARE_METAL.

Toolchain > ARM/GNU C Compiler > Directories : Provide the api folder path(\c\src\api).
Toolchain > ARM/GNU C Compiler > Optimization : Optimize for size(-Os)

10. Now save the changes and build the project. After building the project successfully download code to the flash (Ctrl+Alt+F5) and run the api.

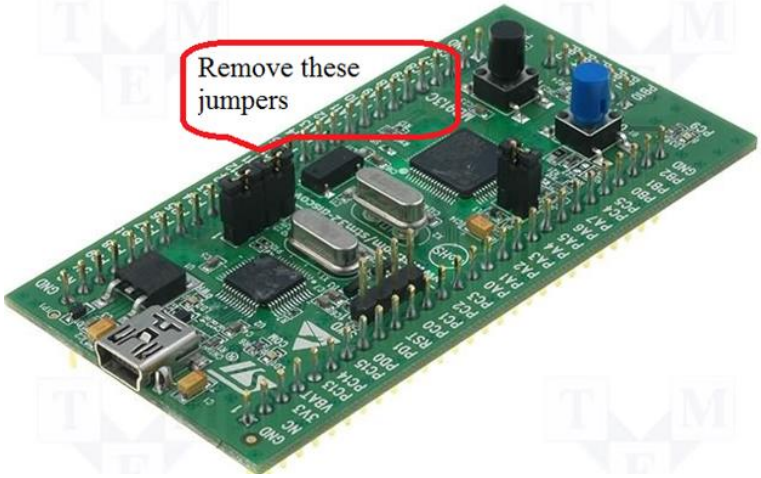
7.3 Embedded Pi using Keil

This example describes how to configure, compile C API, and run sample application for **STM32F103RB**.

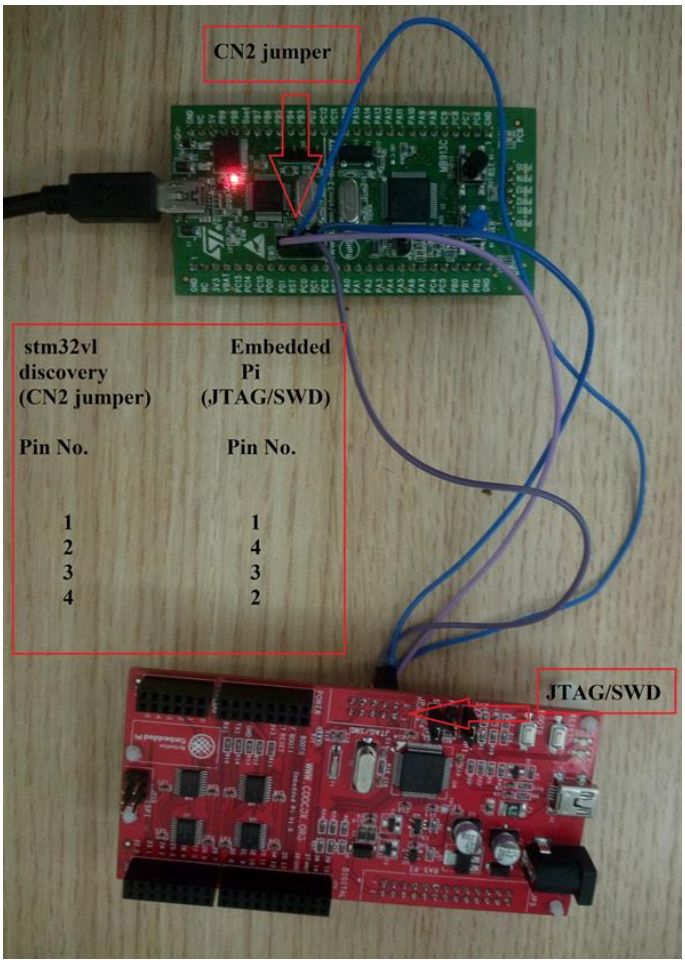
7.3.1 Hardware Connection

EMBEDDED PI and M6e Reader Setup:

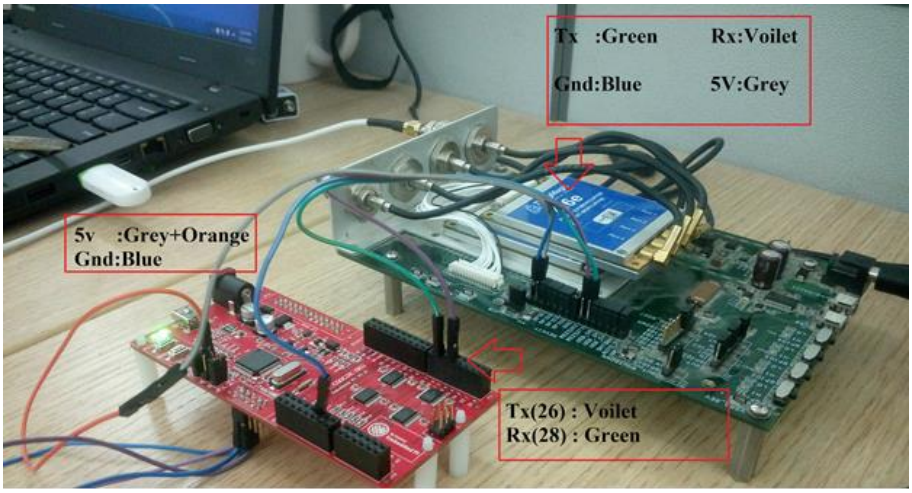
1. To download the binary file to the board, we need to use some debugger since we don't have on board debugger in this Embedded Pi board.
2. For our development, we used STM32vl Discovery on board debugger to download the binary files to the embedded pi flash.
3. Connection of stm32vl Discovery and embedded pi are as shown below:



4. Connect both stm32vl Discovery and embedded board with their SWD section as shown.



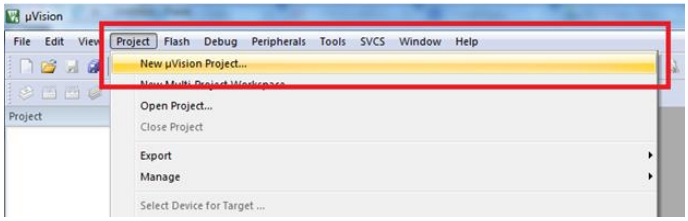
5. Now stm32vl discovery is connected to embedded pi. So, we can download flash of embedded pi using stm32vl Discovery board.
6. Now connect the M6e reader with embedded pi as shown below and run the application.



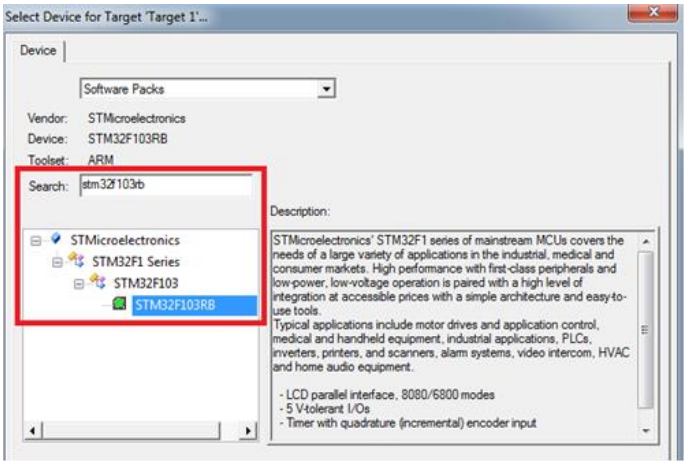
7.3.2 Project Creation

I. Open Keil MDK(ARM) Tool

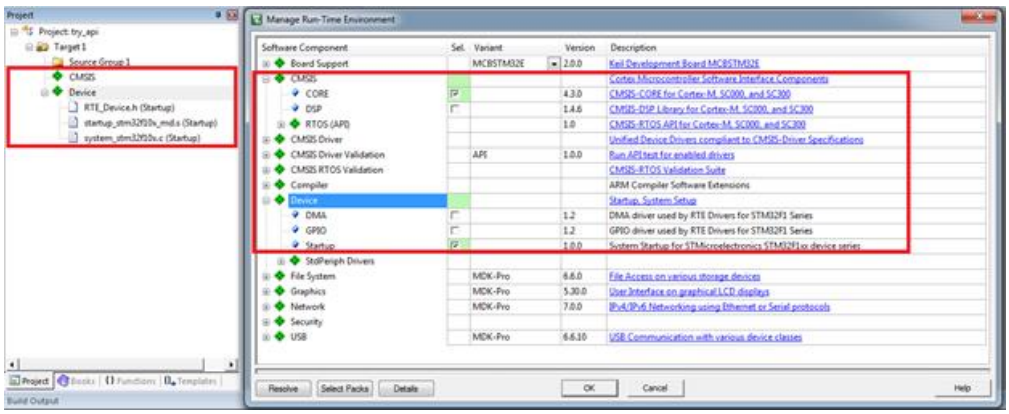
II. Create project.



III. Now select device "**STM32F103RB**".



IV. Add startup and CMSIS core files of **stm32F103RB** only. (As shown in screenshot below)



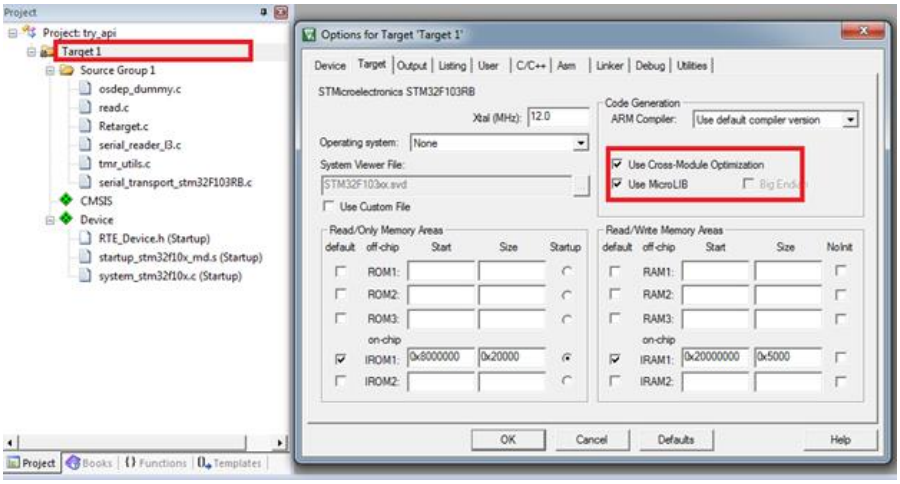
V. Now add the below files (which are bare minimum files required for api) to the project: -

- hex_bytes.c
- Osdep_dummy.c
- Serial_reader.c
- Serial_reader_l3.c
- serial_transport_stm32F103RB.c
- tm_reader.c
- tmr_param.c
- tmr_sterror.c
- tmr_utils.c
- Read.c

Get these files from **source>>source files** folder in “bare_min_final_stm32.rar” project (folder is attached with mail).

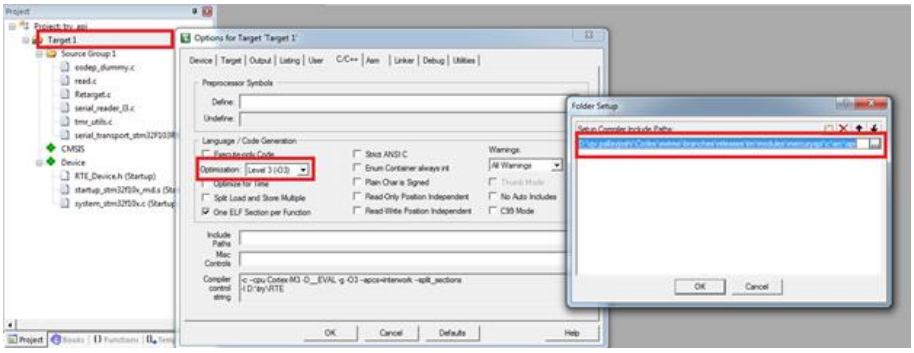
Note: *If you are using Nano module, please change the region to ‘TMR_REGION_NA2’ in bare_min_final_stm32\Source\Source file\read.c file.*

VI. Need to change some settings and include some paths as a reference to the project as shown below:

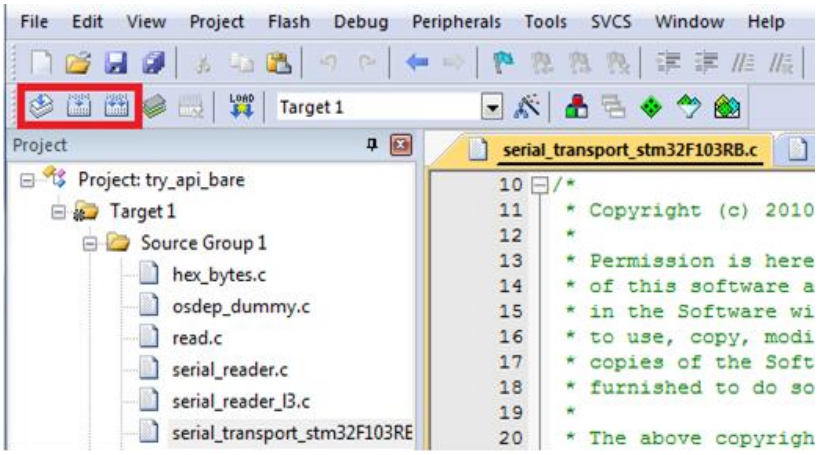


Navigate to C/C++ tab in options of target project and do below changes: -

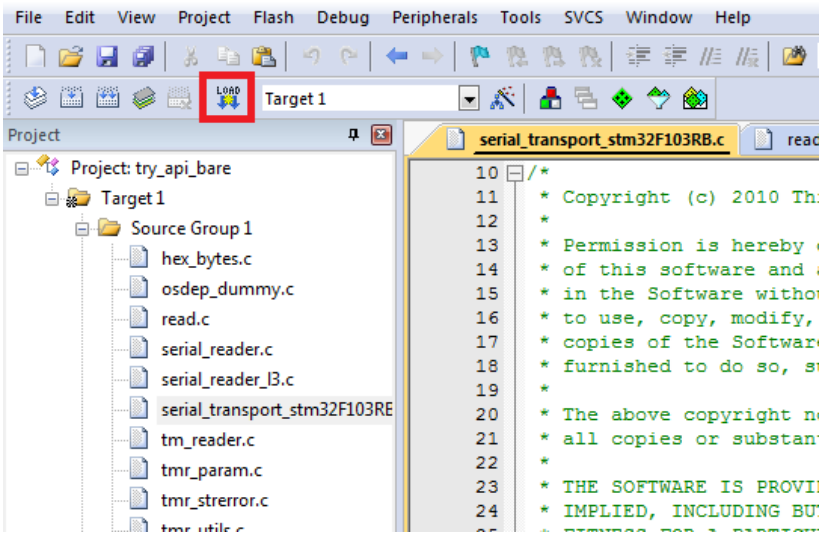
- ➔ Change **“Include Paths”** reference of your headers files (reference to **source>>include files** folder in project folder).
  <mercuryapi>\c\src\api
- ➔ Define pre-processor macro – **BARE_METAL**
- ➔ Under Language/Code Generation, change **“Optimization”** to “**Level 3 (-O3)”**



VII. Save the changes and build the project. (Shortcut key is F7)



VIII. Now download flash to the STM32F103RB board. (Shortcut key is F8)



7.4 STM32 (Olimax) using Keil

This section describes how to install, connect, build, compile C API and run sample application for **STM32-P103**. Embedded Pi board is being used as an example of working with **STM32-P103** processor.

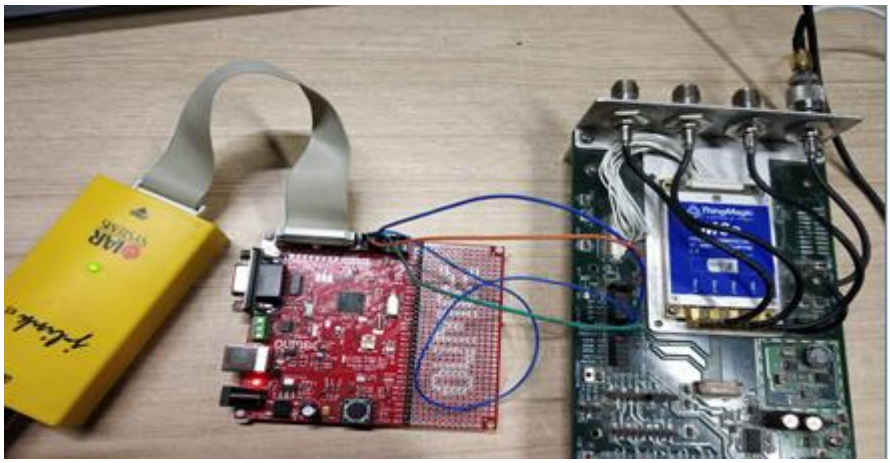
7.4.1 Prerequisites

- Install Keil IDE

- Install Keil: STM32F1xx_DFP Pack installer appropriate to the board you are using. Here we are using **STM32F103RBT6** microcontroller.

7.4.2 Hardware Connection

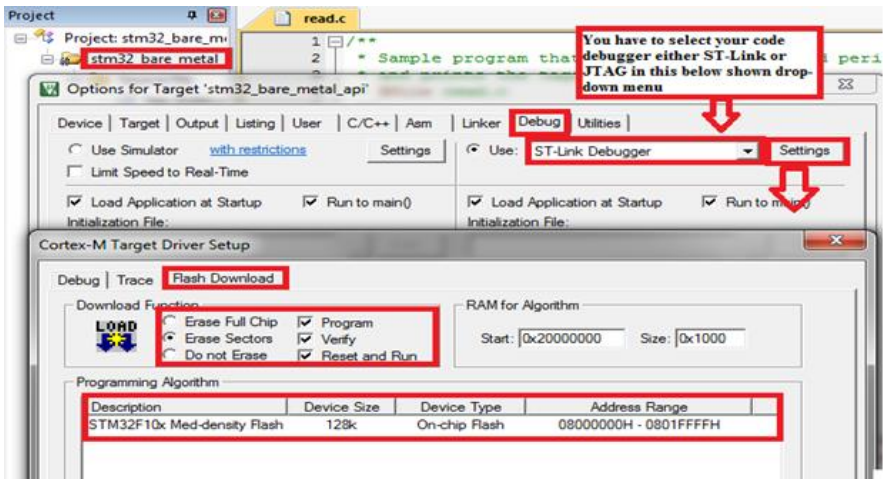
1. Use ST-Link or JTAG debugger to download the code to the board as there is no onboard debugger in STM32.
2. Connect the M6e Reader with STM32 board and run the application. Following are the connection details as depicted in the picture below:
    1. TX of M6e Devkit <-> STM32 RX (Pin 4)
    2. RX of M6e Devkit <-> STM32 TX (Pin 3)
    3. GND <-> STM32 GND (Pin 2)
    4. VCC<->STM32 VCC



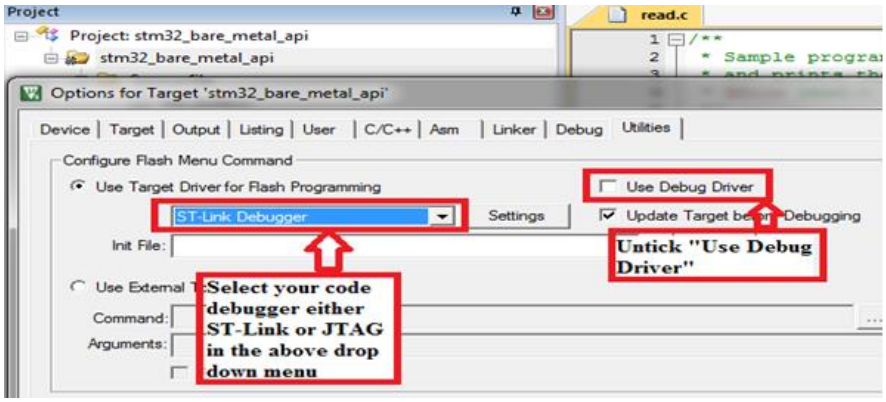
Connection of Embedded Pi and M6e Reader

7.4.3 Project Creation

1. Right click on the target project **stm32_bare_metal_api** in Keil IDE >> Options for Target '**stm32_bare_metal_api'**.
2. Navigate to Debug>>Settings>>Flash Download and select the check boxes and programming algorithm shown below:



3. Navigate to Utilities tab of the target project and deselect “Use Debug Driver” option and select the code debugger as shown below:



4. Save the above changes to the project and build the project using shortcut key F7.
5. Download flash to the STM32F103RB board using shortcut key F8.

7.5 STM32L476VG using Keil

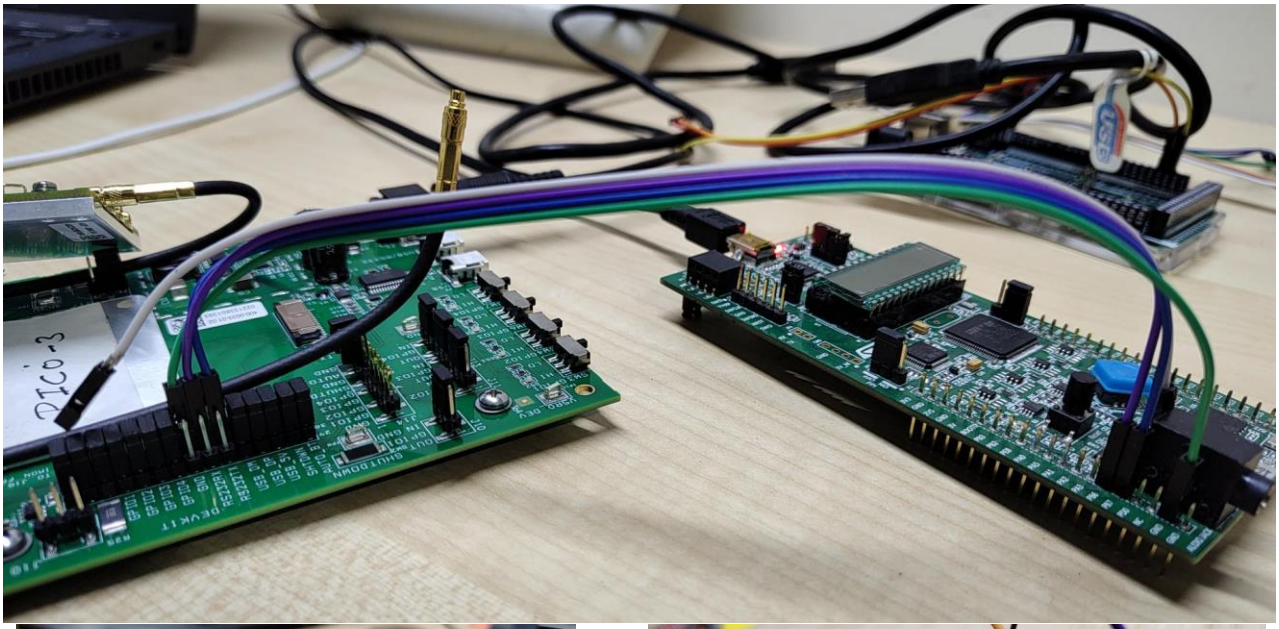
This section describes how to compile C API and run sample applications on **STM32L476VG**.

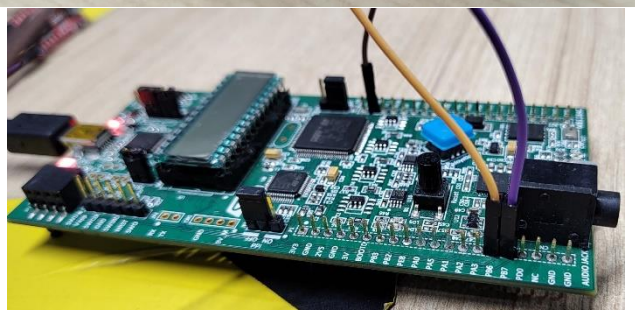
7.5.1 Prerequisites

- Install Keil IDE
- Install Keil:STM32L4xx_DFP Pack installer appropriate to the board you are using. Here we are using an **STM32L476VG** microcontroller.

7.5.2 Hardware Connection

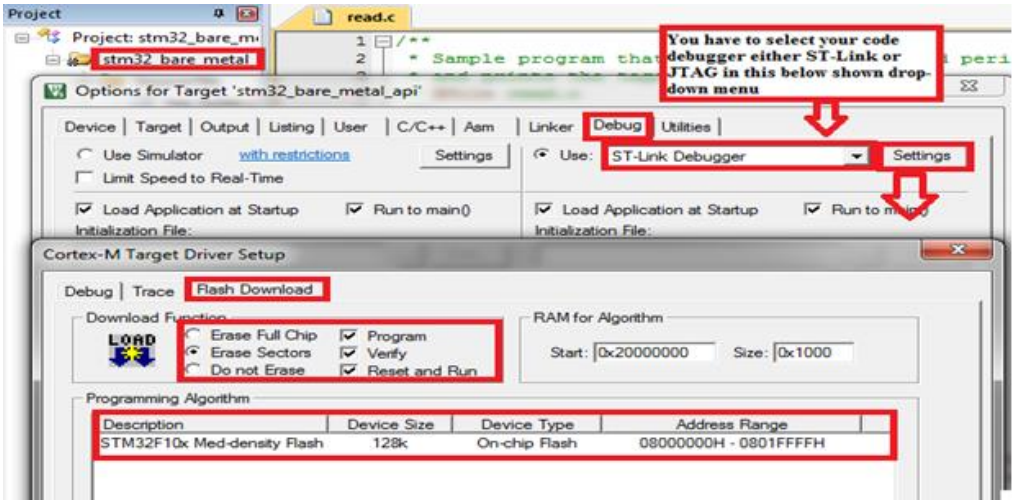
1. We are not using any ST-Link or JTAG debugger to download the code to the board as there is an Onboard ST-LINK debugger in STM32.
2. Connect the Micro module with STM32 board and run the application. Following are the connection details as depicted in the picture below:
    1. TX of Micro Devkit <-> STM32 RX (PB7)
    2. RX of Micro Devkit <-> STM32 TX (PB6)
    3. GND <-> STM32 GND



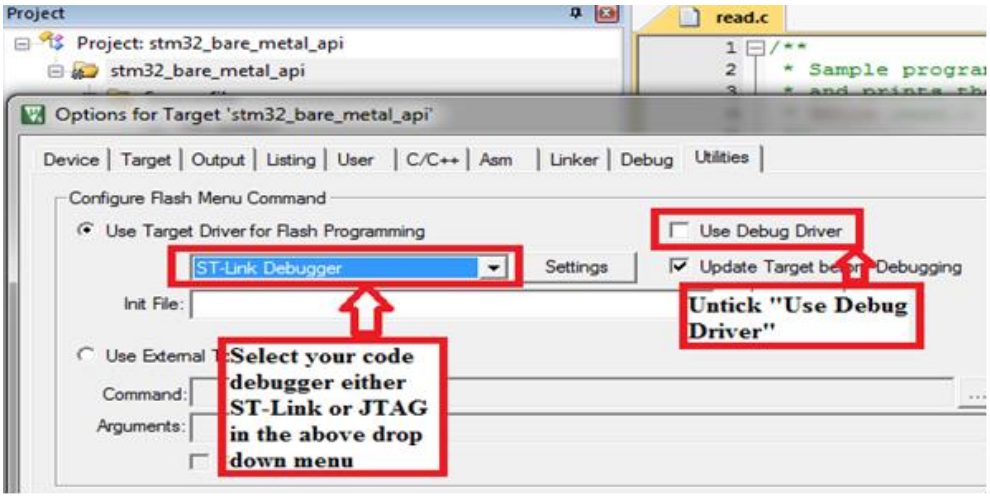


7.5.3 Project Creation

1. Download the MercuryAPI SDK and navigate to c\stm32_proj\STM32L476VG.
2. Open ***stm32l476_bare_metal_api.uvprojx*** file using Keil IDE.
3. Right click on the target project **stm32l476_bare_metal_api** in Keil IDE >> Options for Target '**stm32l476_bare_metal_api**'.
4. Navigate to Debug>>Settings>>Flash Download and select the check boxes and programming algorithm shown below:



5. Navigate to Utilities tab of the target project and deselect “Use Debug Driver” option and select the code debugger as shown below:



6. Save the above changes to the project and build the project using shortcut key F7. Download flash to the **STML476VG** board using shortcut key F8.

7.5.4 STM32L476VG Supported baud rates.

STM32L476VG, the supported baud rates are 9600, 19200, 38400, 57600 ,230400 and 115200.

7.6 Arduino Nano ESP32 using Arduino IDE

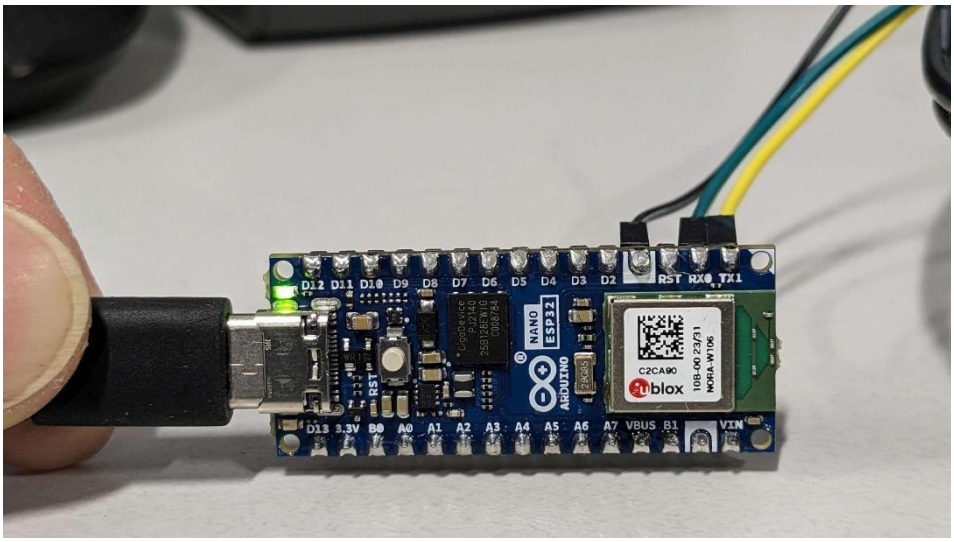
This document describes how to configure, compile C API, and run sample application for Arduino Nano ESP32.

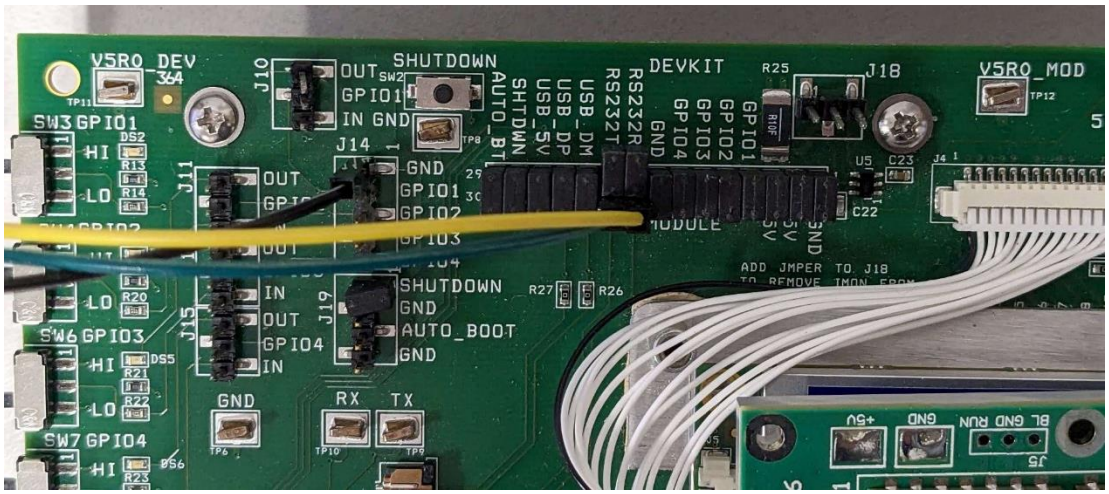
7.6.1 Prerequisites

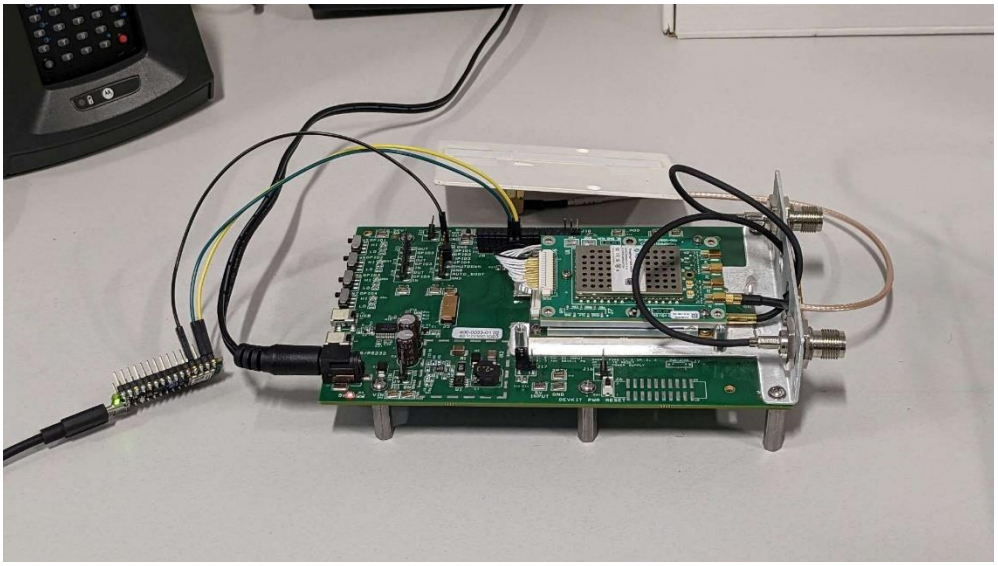
- Download the Arduino IDE from
  - Linux: https://www.arduino.cc/en/Guide/Linux
  - Windows: https://www.arduino.cc/en/Guide/Windows
- Download Mercury API SDK from the Jadak website.

7.6.2 Hardware Connection

Arduino	**DevKit**
TX1	RS232_RX
RX0	RS232_TX
GND	GND


D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2
RST RX0 TX1
NANO ESP32
ARDUINO
C2CA90
ublox
NORA-W106
D13 3.3V B0 A0 A1 A2 A3 A4 A5 A6 A7 VBUS B1
VIN


V5R0_DEV
J10
OUT
GPIO1
IN GND
SHUTDOWN
SW3 GPIO1
HI
LO
J14
GND
GPIO1
GPIO2
GPIO3
J11
OUT
IN
OUT
GPIO4
IN
J15
J19
SHUTDOWN
GND
AUTO_BOOT
GND
SW6 GPIO3
HI
LO
SW7 GPIO4
HI
GND
RX
TX
DEVKIT
J18
V5R0_MOD
R25
R27
R26
ADD JMPER TO J18
+5V
GND
RUN
BL



7.6.3 Project Creation

- Download latest (mercuryapi-BILBO-1.37.1.xx) file. And unzip it.
- Create a new folder and rename it to *mercuryapi_src*.
- To add dependency files in Arduino project, copy below files from the folder *<mercuryAPI>\c\src\api* to the newly created folder *mercuryapi_src*.

Hex_bytes.c, Osdep.h, Osdep_arduino.c, Serial_reader.c, Serial_reader_imp.h, Serial_reader_l3.c, Serial_transport_arduinoNanoESP32.c, tm_config.h, tm_reader.c, tm_reader.h, tm_reader_async.c, tmr_filter.h, tmr_gen2.h, tmr_gpio.h, tmr_ipx.h, tmr_iso14443a.h, tmr_iso14443b.h, tmr_iso15693.h, tmr_iso180006b.h, tmr_lf125khz.h, tmr_lf134khz.h, tmr_param.c, tmr_params.h, tmr_read_plan.h, tmr_region.h, tmr_serial_reader.h, tmr_serial_transport.h, tmr_status.h, tmr_strerror.c, tmr_tag_auth.h, tmr_tag_data.h, tmr_tag_lock_action.h, tmr_tagop.h, tmr_tag_protocol.h, tmr_types.h, tmr_utils.h, tmr_utils.c.

- Open the **tm_config.h** file (from *mercuryapi_src).* Uncomment below line of code and define BARE_METAL macro.

  /** To build API for baremetal platform. */

  //#define **BARE_METAL**

- Rename **Serial_transport_arduinoNanoESP32.c** (from *mercuryapi_src)* to **Serial_transport_arduinoNanoESP32.cpp.**
- Zip the folder *mercuryapi_src -> mercuryapi_src.zip*
- Open the Arduino IDE and copy the Read.ino sample code from the path*<mercuryAPI>\\c\baremetal_proj\arduino_proj\Arduino_Mega2560\Read.*

  Note- Read.ino sample code is compatible with Arduino_Mega2560 by default. To make it compatible with ArduinoNano ESP32, need to make few changes in Read.ino sample code.

  - TAG_SEARCH_TIME is 500 ms by default. Need to run longer (ex- 5000ms) for ArduinoNano ESP32 because 500ms is too short to actually receive any tag reads on your host, because the ESP32's USB-CDC port takes a second or two to come up after boot.

    ```
    #define TAG_SEARCH_TIME        5000 //ms
    ```

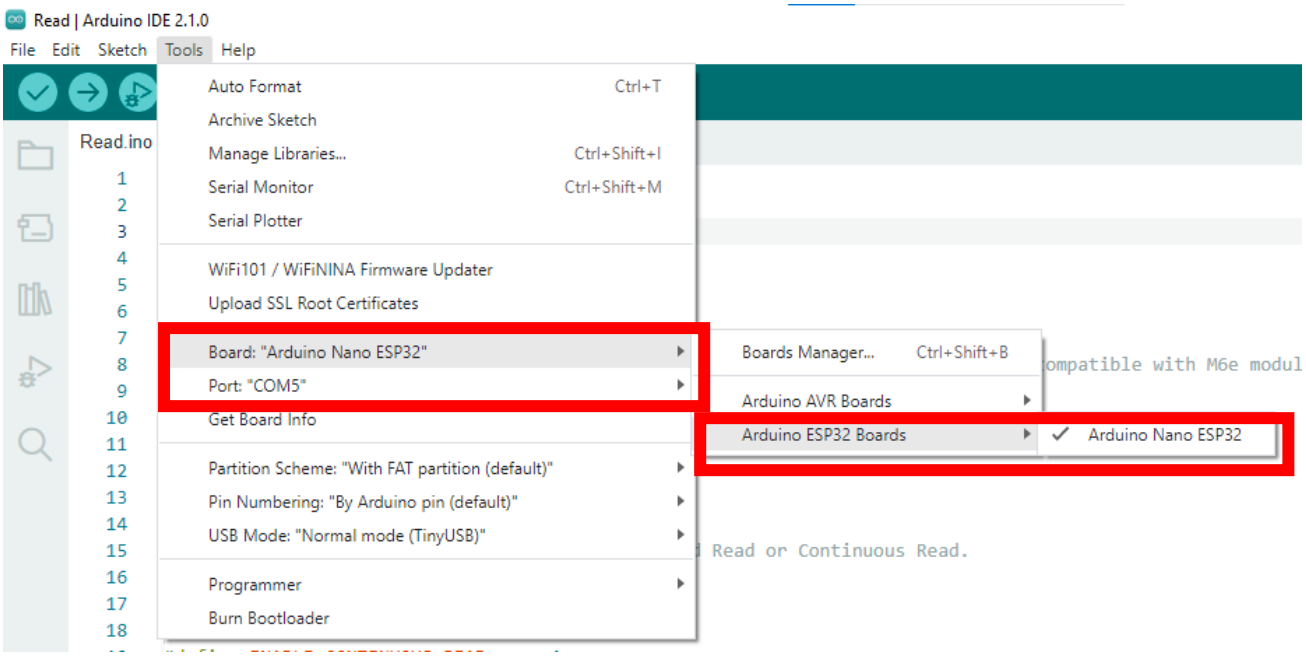
  - Use 'console' type as "USBCDC" instead of "HardwareSerial".

    ```
    //Comment below line for Arduino Nano ESP32
    static HardwareSerial *console = &Serial;
    //Uncomment below line for Arduino Nano ESP32
    //static USBCDC *console = &Serial;
    ```

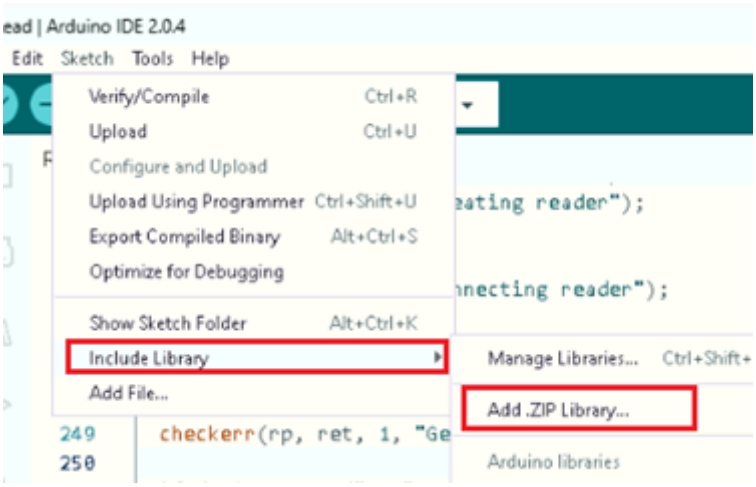
- In initializeReader() function, use device ‘Serial0’ in TMR_create()

```
ret = TMR_create(rp, "tmr:///Serial0");
```

- Select board in **tools->Board->Arduino ESP32 Boards->Arduino Nano ESP32**. If **Arduino Nano ESP32** is not visible in Board, first install libraries required for **Arduino Nano ESP32.**
- Select comport in **Tools->Port**.



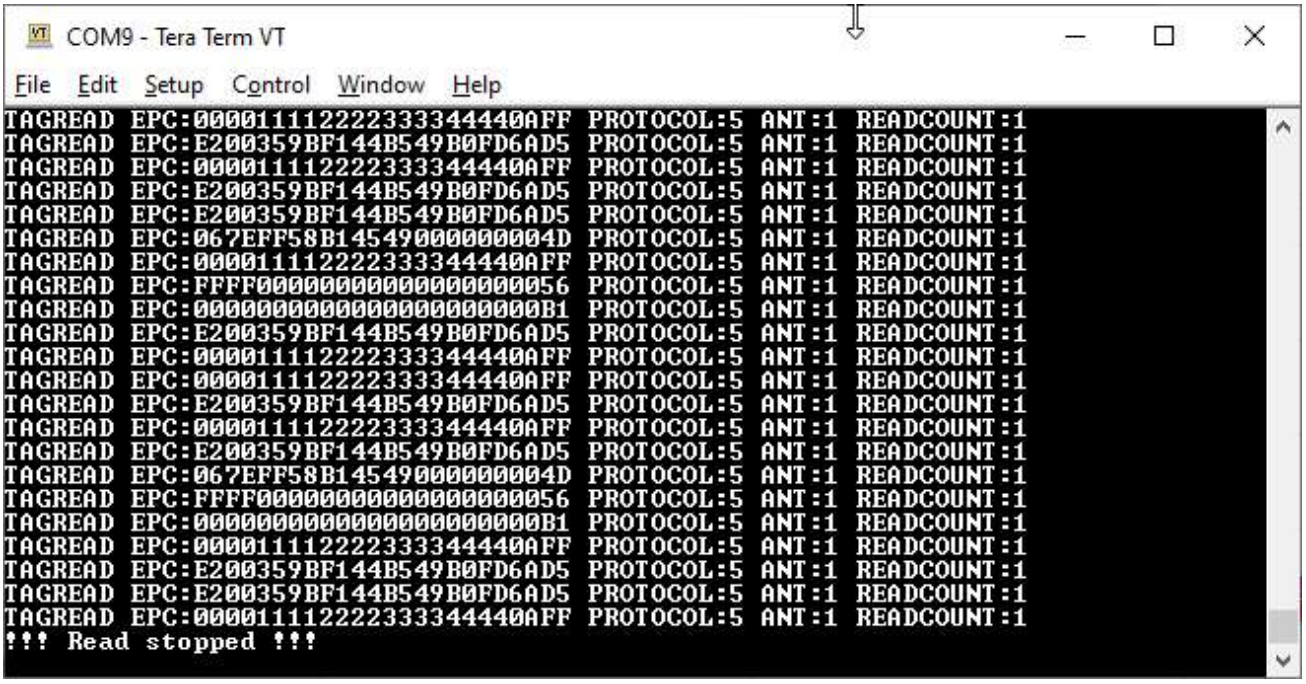
- Give path of *mercuryapi_src.zip* file in **Sketch->Include Library-> Add .ZIP Library**.



- Build and run.
  - ENABLE_CONTINUOUS_READ macro is used to select either CONTINUOUS READ or TIMED READ. This is set to 1 by default. The CONTINUOUS READ is performed for specified time.

- If ENABLE_CONTINUOUS_READ macro is set to 0, then the module performs TIMED READ for specified time.

- If you are not able to see the tag reports on Arduino Serial Monitor, then use '**Tera Term**' to see the tag reports because it has automatic reconnect. The Arduino's USB virtual COM port goes away and comes back when the Arduino is reset, and you'll lose messages if you don't reconnect promptly.
  - Open Tera Term on the Arduino's USB COM port.
  - You should see tag read output on the USB COM port a few seconds after resetting the Arduino (either through the Arduino IDE or by clicking the reset button).



  - If you try to run again (continuous read or timed read) and observe this error "ERROR connecting reader: 0x1000001" then make sure first reset module using DevKit RESET button and then reset Arduino (either through the Arduino IDE or by clicking the reset button).

8 How to Fix the Code Size of the Image

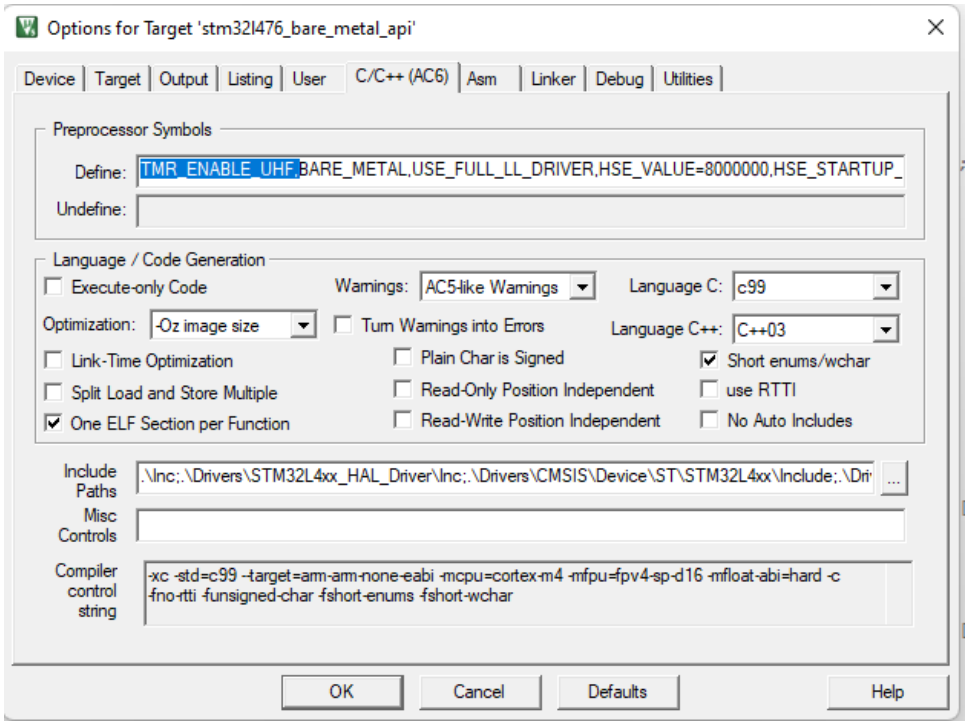
8.1 In Keil

1. **Model Selection**

   By default, C API contains both UHF and HF/LF code. Based on the use case, user can disable either UHF or HF code to reduce the code size.

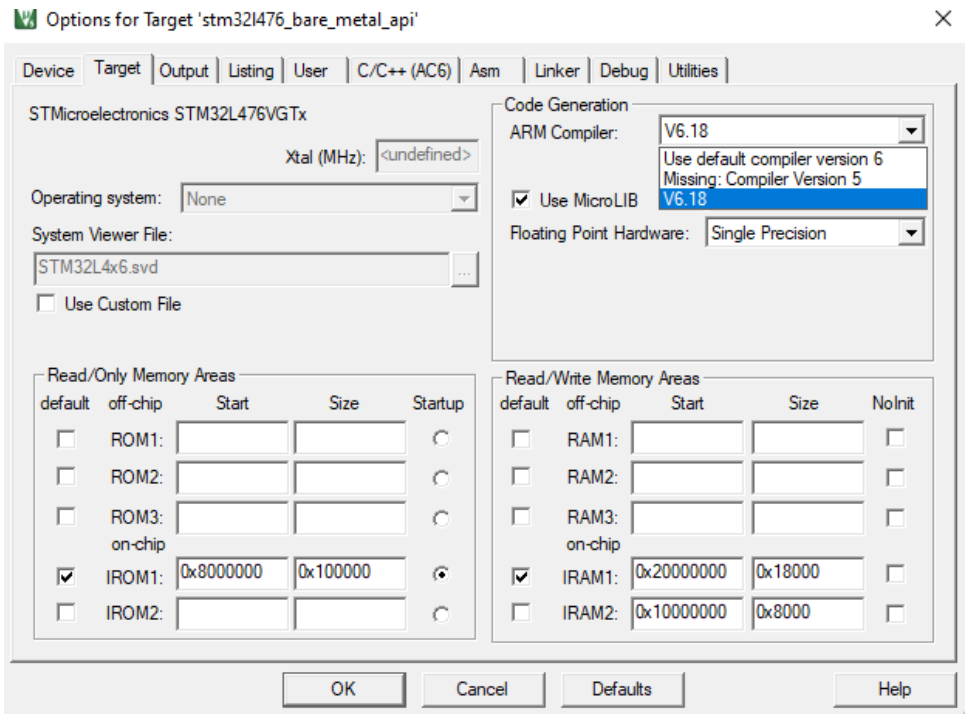
   If HF/LF module is used, enable HF/LF and disable UHF:
   Define preprocessor macro "**TMR_ENABLE_HF_LF**" in options->C/C++.

   If UHF module is used, enable enable UHF and disable HF/LF:
   Define preprocessor macro "**TMR_ENABLE_UHF**" in options->C/C++.



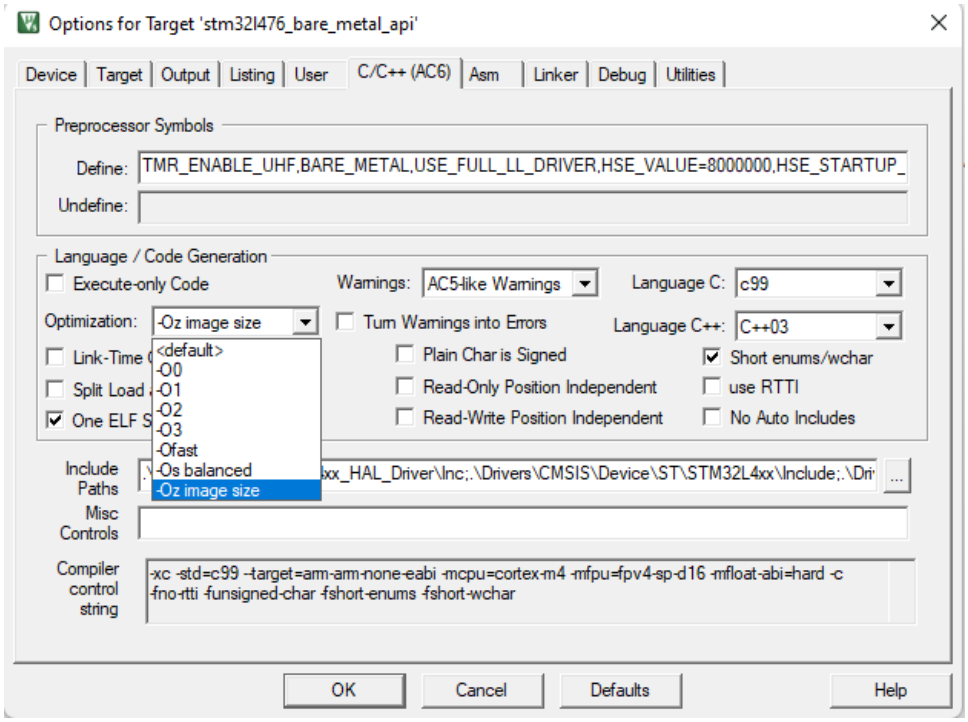
2. **ARM compiler Selection**

   Select latest ARM compiler version.



3. **Optimization Option**

   Select optimization for size option.


Options for Target 'stm32l476_bare_metal_api'
Device
Target
Output
Listing
User
C/C++ (AC6)
Asm
Linker
Debug
Utilities
Preprocessor Symbols
Define:
TMR_ENABLE_UHF,BARE_METAL,USE_FULL_LL_DRIVER,HSE_VALUE=8000000,HSE_STARTUP_
Undefine:
Language / Code Generation
Execute-only Code
Warnings:
AC5-like Warnings
Language C:
c99
Optimization:
-Oz image size
Turn Warnings into Errors
Language C++:
C++03
<default>
-O0
-O1
-O2
-O3
-Ofast
-Os balanced
-Oz image size
Plain Char is Signed
Short enums/wchar
Read-Only Position Independent
use RTTI
Read-Write Position Independent
No Auto Includes
Include Paths
Misc Controls
Compiler control string
-xc -std=c99 -target=arm-arm-none-eabi -mcpu=cortex-m4 -mfpu=fpv4-sp-d16 -mfloat-abi=hard -c
-fno-rtti -funsigned-char -fshort-enums -fshort-wchar
OK
Cancel
Defaults
Help

* * *