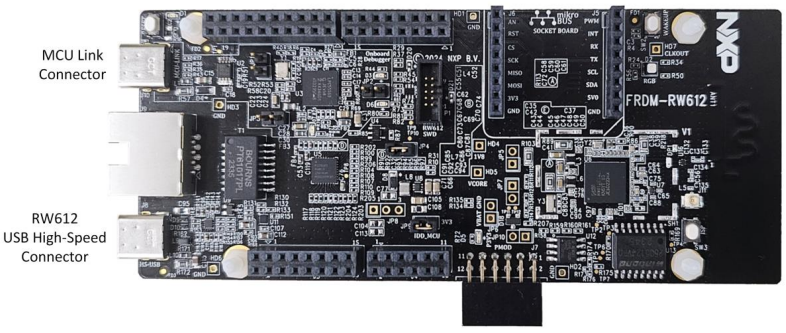
FRDM RW612 – Bluetooth Low Energy Temperature Sensor demo using Zephyr

Contents

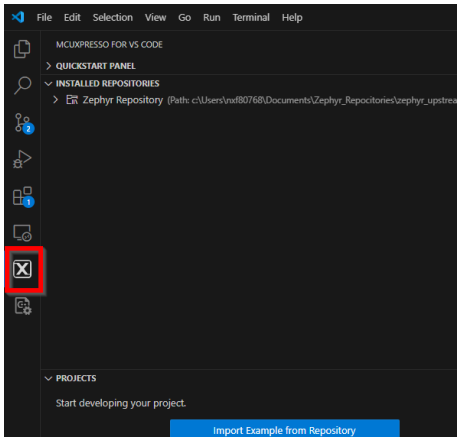
BLE peripheral Thermometer demo

This demo shows the temperature measured from the i2c temperature sensor integrated in the board. The information can be monitored in the UART terminal or in the IoT Toolbox app.

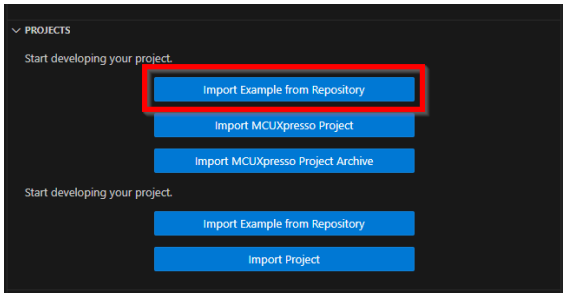


Import example from Zephyr repository.

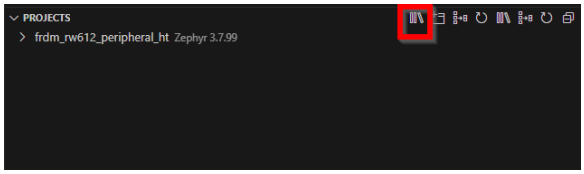
1. Open VSCode and go to the MCUXpresso extension.



2. Click on "PROJECTS/Import Example from Repository".



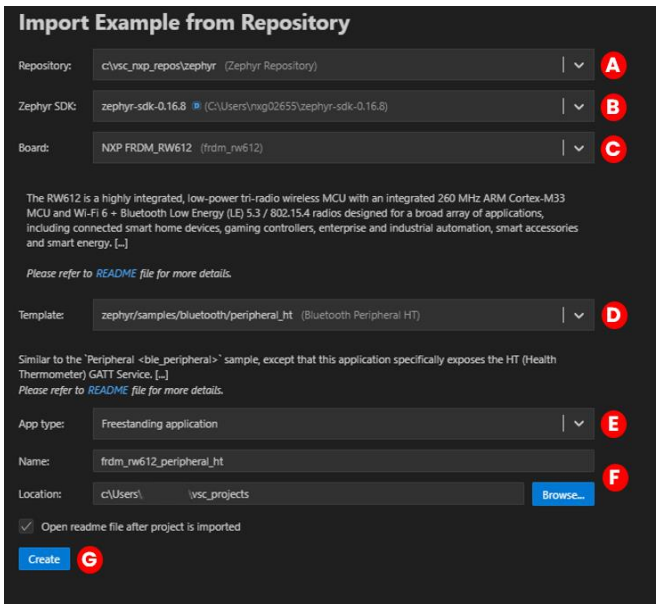
If you already have a project in your workspace, then you will see this view:



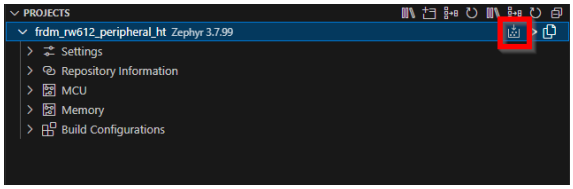
And the red box indicates witch is the button used to add a new example.

3. Configure Zephyr project in VSCode
   a. Select the Zephyr repo previously downloaded.
   b. Select "Zephyr SDK 0.16.0".
   c. Select "NXP FRDM_RW612".
   d. Select "Bluetooth Peripheral HT demo".

e. App type: “Freestanding application”.
f. You can choose the name and the preferred location.
g. Click on the “Create” button.



4. Click on Build icon

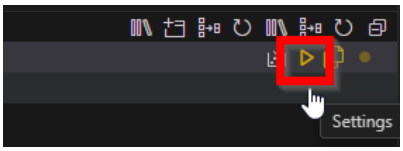


Make sure you select the name of the project to see the build button.

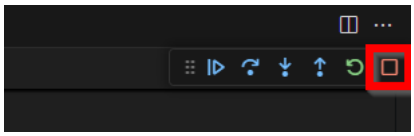
Debug your project.

** Before debugging projects based on Zephyr we need to apply a patch to the project. Steps 1 to 5 deals with this.

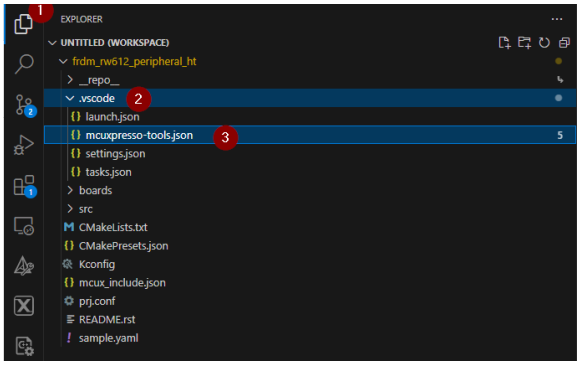
1. Click on debug.



2. Stop the debug session.

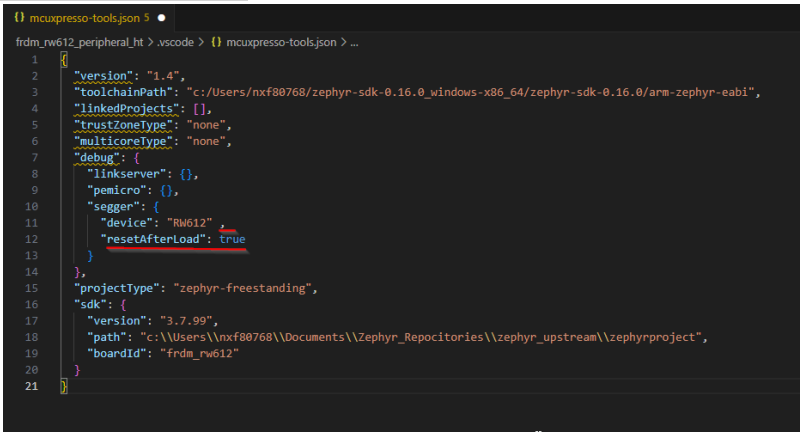


3. After the first build we will see a new folder into the file Explorer.



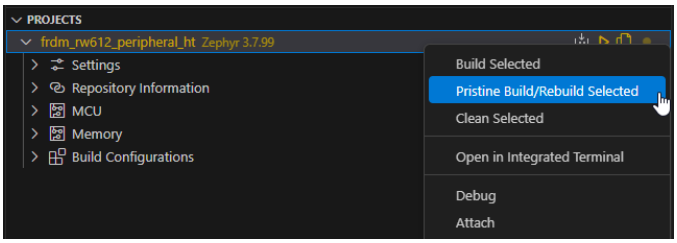
4. Open the mcuxpresso-tools.json file and add the following line:

"resetAfterLoad": true

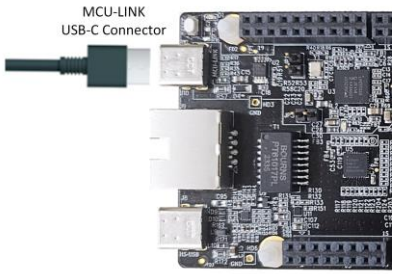


5. Clean and build

   Go back to the MCUXpresso extension, right click in the project name and click on Pristine Build/Rebuild Selected



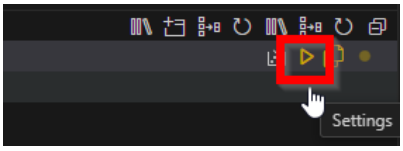
6. Connect the board to your computer with a USB-C cable, MCU Link Connector (J10)



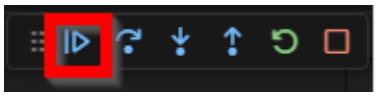
7. Get a serial terminal. You can download Tera Term: https://teratermproject.github.io/index-en.html

8. Open the serial terminal with the following settings:
   - COM Port of your device
   - 115200 baud-rate
   - 8-bit data
   - No parity or flow control
   - 1 stop bit

9. Click on Debug button

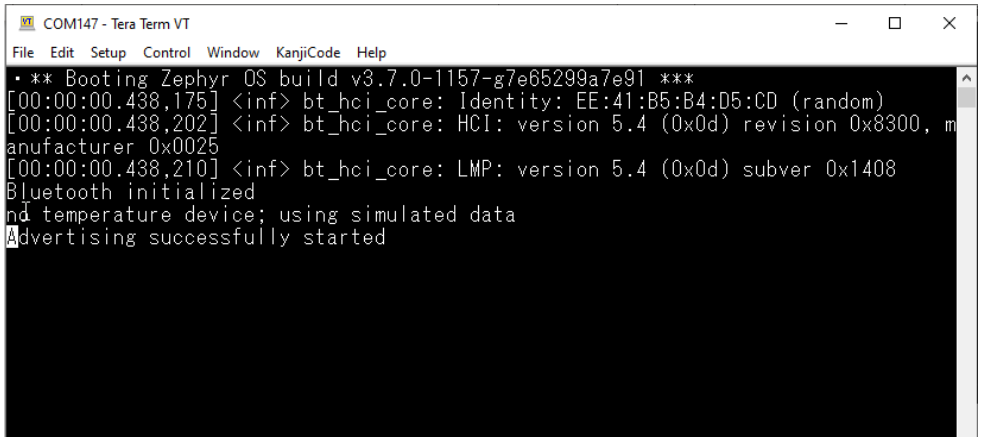


10. Click on Play button



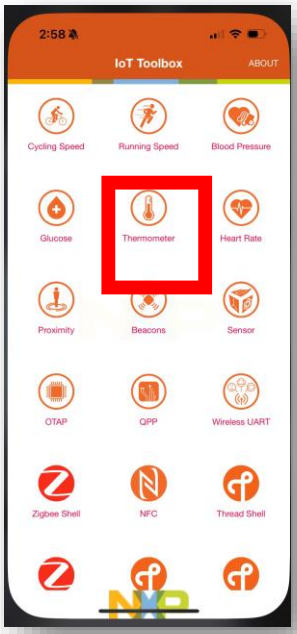
11. See results

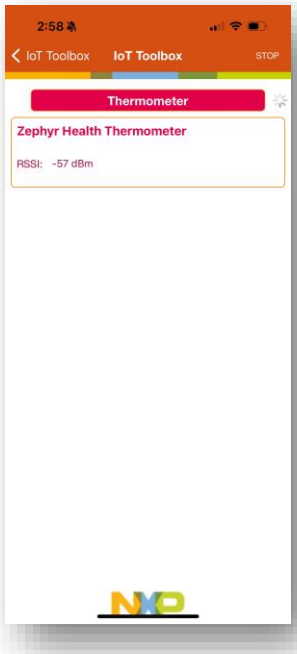
    In the terminal you can see the demo information and the simulated temperature

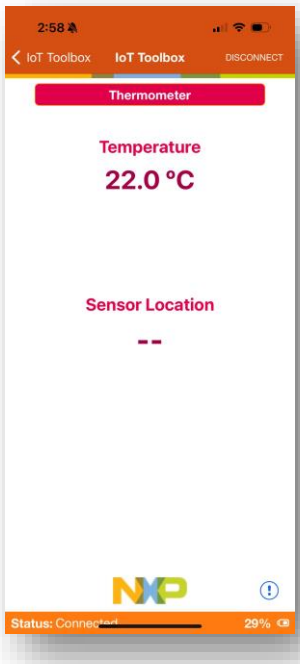


Using IoT Toolbox

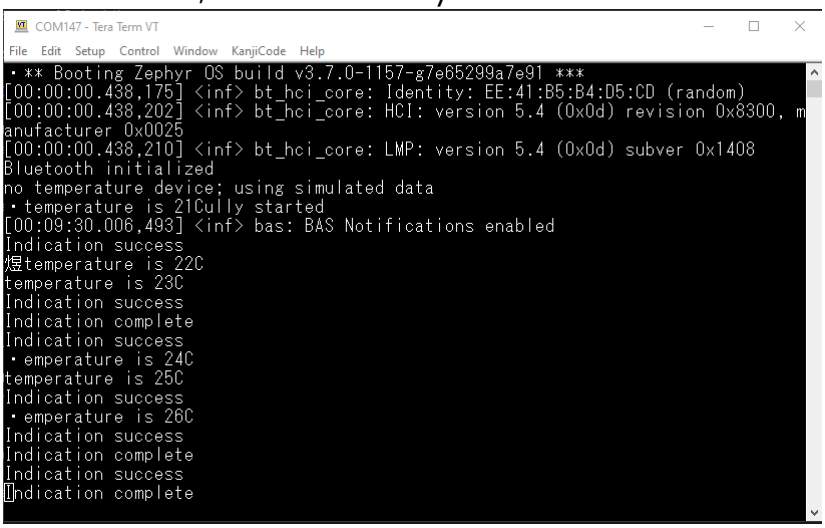
1. Open the IOT Toolbox app in your smartphone.
2. Go to thermometer.
3. Select the Zephyr Health Thermometer device. You will start getting temperature information.







4. Once connected, In the terminal you will see this:

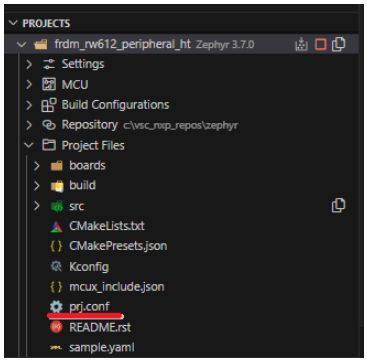


Adding on board I2C Temperature sensor

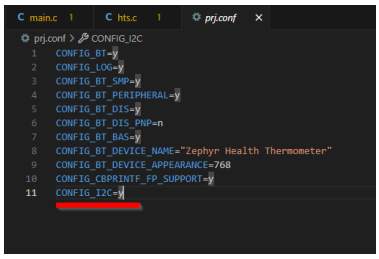
By default, this demo simulates the temperature measurements, the next step in this training is to add the support for i2c, so we can use the on-board sensor and take real measurements.

1. **Enable I2C device Driver on device Tree**

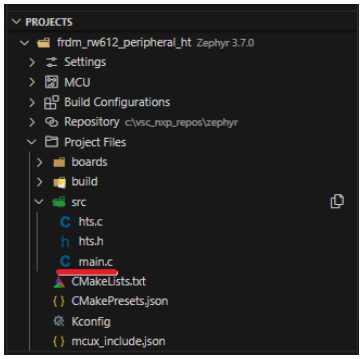
   In zephyr we can add drivers using some macros. In the project folder we can find the file “prj.conf”



   Add this to the bottom : CONFIG_I2C=y



2. Now we need to add the device in the code, go to main.c and add the following lines first:



```
#include <zephyr/drivers/i2c.h>
#define TEMPSEN_ADD (0x48)
#define I2C_DEV_NODE DT_ALIAS(i2c_0)
const struct device *const i2c_dev = DEVICE_DT_GET(I2C_DEV_NODE);
uint32_t i2c_cfg = I2C_SPEED_SET(I2C_SPEED_STANDARD) |
I2C_MODE_CONTROLLER;
extern double temperature;
```

```
 * SPDX-License-Identifier: Apache-2.0
 */

#include <zephyr/types.h>
#include <stddef.h>
#include <string.h>
#include <errno.h>
#include <zephyr/sys/printk.h>
#include <zephyr/sys/byteorder.h>
#include <zephyr/kernel.h>

#include <zephyr/bluetooth/bluetooth.h>
#include <zephyr/bluetooth/hci.h>
#include <zephyr/bluetooth/conn.h>
#include <zephyr/bluetooth/uuid.h>
#include <zephyr/bluetooth/gatt.h>
#include <zephyr/bluetooth/services/bas.h>

#include "hts.h"
#include <zephyr/drivers/i2c.h>

#define TEMPSEN_ADD (0x48)
#define I2C_DEV_NODE DT_ALIAS(i2c_0)

const struct device *const i2c_dev = DEVICE_DT_GET(I2C_DEV_NODE);
uint32_t i2c_cfg = I2C_SPEED_SET(I2C_SPEED_STANDARD) | I2C_MODE_CONTROLLER;
extern double temperature;
```

3. Now we are adding the functions to initialize and read the temperature sensor. This is also in main.c.
    a. First put the prototypes on top of the file.

```
void configure_sensor(void);
void update_temperature(void);
```

```
#include "hts.h"
#include <zephyr/drivers/i2c.h>
#define TEMPSEN_ADD (0x48)
#define I2C_DEV_NODE DT_ALIAS(i2c_0)

void configure_sensor(void);
void update_temperature(void);


const struct device *const i2c_dev = DEVICE_DT_GET(I2C_DEV_NODE);
uint32_t i2c_cfg = I2C_SPEED_SET(I2C_SPEED_STANDARD) | I2C_MODE_CONTROLLER;
extern double temperature;
```

b. Now, put the functions' implementation along with the other functions already in file.

```
void update_temperature(void){
    unsigned char datas[2];
    if (!device_is_ready(i2c_dev)) {
            printk("I2C device is not ready\n");
    }
    (void)memset(datas, 0, sizeof(datas));
    /*i2c_read() */
    if (i2c_read(i2c_dev, datas, 2, 0x48)) {
            printk("Fail to fetch sample from sensor \n");
    }
      temperature = (double)((((uint16_t)datas[0] << 8U) |
(uint16_t)datas[1]) >> 4U);
     temperature = temperature * 0.0625;
            k_sleep(K_MSEC(1));

}

void configure_sensor(void){
    if (!device_is_ready(i2c_dev)) {
            printk("I2C device is not ready\n");
    }
    /*Verify i2c_configure() */
    if (i2c_configure(i2c_dev, i2c_cfg)) {
            printk("I2C config failed\n");
    }

    k_sleep(K_MSEC(1));

}
```

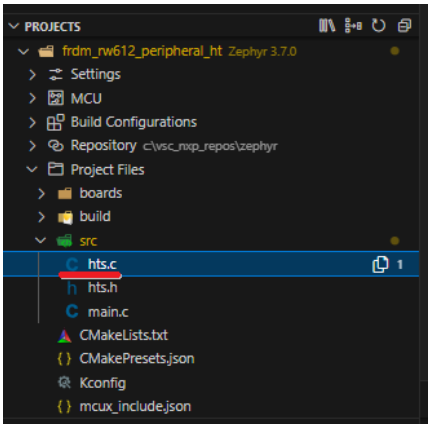
4. Now you need to call this functions:
   We need to configure de sensor before the loop in main()

```
configure_sensor();
update_temperature();
```

```
    /* Implement indicate. At the moment there is no suitable way
     * of starting delayed work so we do it here
     */
    configure_sensor();
    while (1) {
        k_sleep(K_SECONDS(1));

        /* Temperature measurements simulation */
        hts_indicate();
        /*Read i2c sensor*/
        update_temperature();
        /* Battery level simulation */
        bas_notify();
    }
    return 0;
}
```

5. Now we need go into the hts_indicate() function, located in hts.c, to disable the temperature simulation.

   In hts.c we are going to delete some lines:



In the function “hts_indicate” remove the following lines:

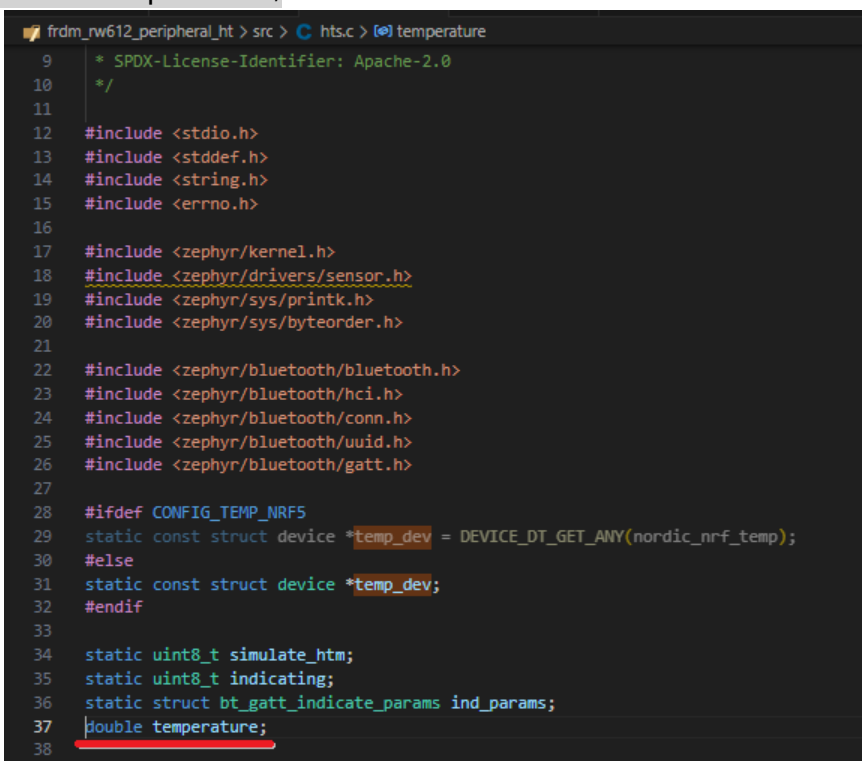
```
void hts_indicate(void)
{
    /* Temperature measurements simulation */
    struct sensor_value temp_value;

    if (simulate_htm) {
        static uint8_t htm[5];
        static double temperature = 20U;
        uint32_t mantissa;
        uint8_t exponent;
        int r;

        if (indicating) {
            return;
        }

        if (!temp_dev) {
            temperature++;
            if (temperature == 30U) {
                temperature = 20U;
            }

            goto gatt_indicate;
        }

        r = sensor_sample_fetch(temp_dev);
        if (r) {
            printk("sensor_sample_fetch failed return: %d\n", r);
        }

        r = sensor_channel_get(temp_dev, SENSOR_CHAN_DIE_TEMP,
                               &temp_value);
        if (r) {
            printk("sensor_channel_get failed return: %d\n", r);
        }

        temperature = sensor_value_to_double(&temp_value);

gatt_indicate:
        printf("temperature is %gC\n", temperature);

        mantissa = (uint32_t)(temperature * 100);
        exponent = (uint8_t)-2;
```

After the change it should look like this:

```
void hts_indicate(void)
{
    /* Temperature measurements simulation */
    struct sensor_value temp_value;

    if (simulate_htm) {
        static uint8_t htm[5];
        uint32_t mantissa;
        uint8_t exponent;
        int r;

        if (indicating) {
            return;
        }

        if (!temp_dev) {

            goto gatt_indicate;
        }

        r = sensor_sample_fetch(temp_dev);
        if (r) {
            printk("sensor_sample_fetch failed return: %d\n", r);
        }

        r = sensor_channel_get(temp_dev, SENSOR_CHAN_DIE_TEMP,
                               &temp_value);
        if (r) {
            printk("sensor_channel_get failed return: %d\n", r);
        }


gatt_indicate:
        printf("temperature is %gC\n", temperature);
```

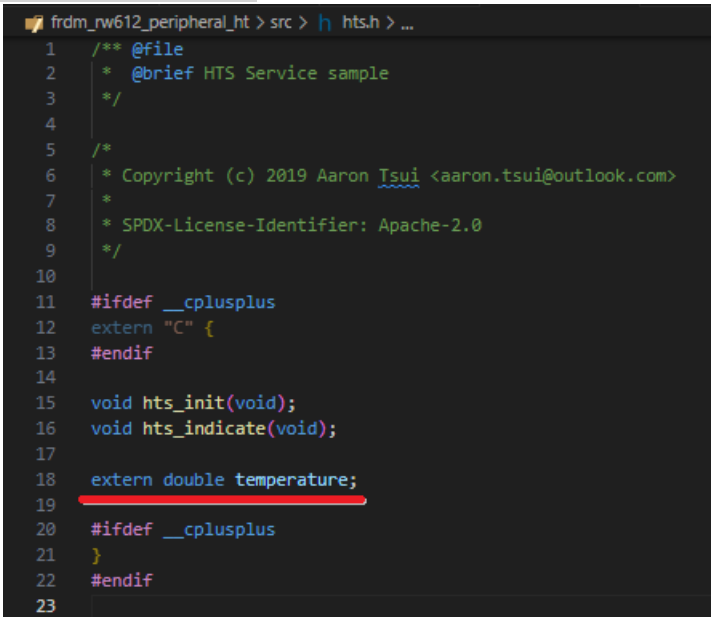
6. Add the following global variable to hts.c:

   double temperature;

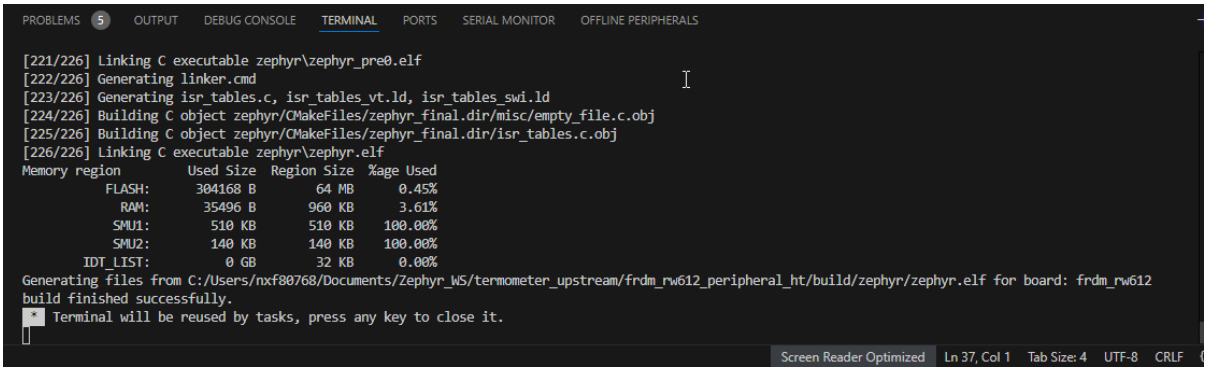


7. Open hts.h, from the same folder, and add the next line of code:

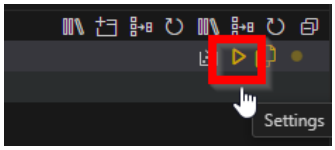
   extern double temperature;

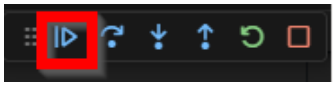


8. Now let's Pristine Build/Rebuild Selected again



9. Press debug and play again





10. In tera term and in the IoT Toolbox you can see the real temperature value and if you put your finger in the sensor, you can see how the temperature increase.