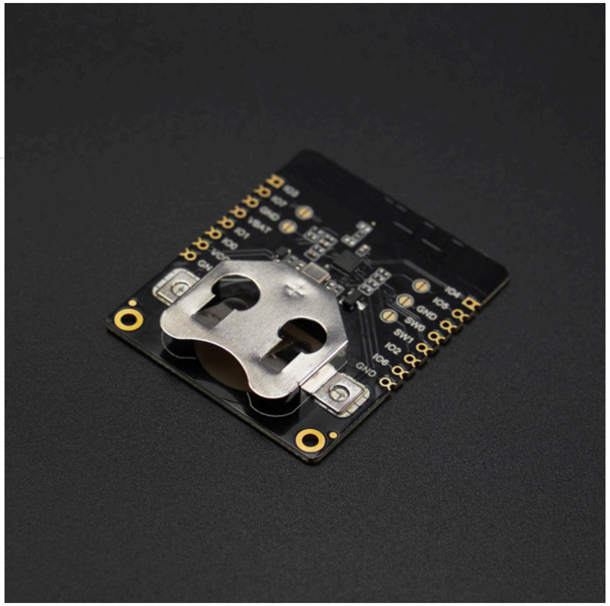
SKU:TEL0168 (https://www.dfrobot.com.cn/goods-3799.html)

(https://www.dfrobot.com.cn/goods-3799.html)



Introduction

Fermion: BLE Sensor Beacon, a wireless beacon that broadcasts sensor data via Bluetooth, with built-in 11-bit ADC acquisition and I2C write/read functionality, can be connected to digital or analogue sensors for data acquisition and broadcasting. Sensor data broadcasted by the beacon can be accessed within the beacon's broadcast range using mobile phones, ESP32 and other devices that support BLE reception.

Fermion: BLE sensor beacons integrate low-power Bluetooth 5.3 technology with self-configurable data formats, such as iBeacon, Eddystone, user-defined, and more. The data format of the beacon broadcast, the content of the broadcast, the broadcast interval and so on can be configured through the graphical interface, without the need for any code programming to complete a Bluetooth beacon. After the configuration is completed, the device power supply is running as a Bluetooth beacon, which will automatically collect sensor data and broadcast to the outside world according to the configuration information. It is suitable for IoT sensor nodes, such as smart farms, offices, factories, warehouses and other scenarios in the data collection node.

Compared to Gravity: BLE Sensor Beacons , the Fermion version mproved:

- Fermion version can be powered by CR2032 coin cell battery
- Added two independent I2C interfaces to acquire I2C sensor data
- Leads to 6 configurable GPIOs

Note: Fermion: BLE sensor beacons need to be configured using the 3.3V USB-TTL tool

If the module is bricked after burning due to wrong configuration file, DFRobot will not provide after-sales service or technical support for the Beacon. The module can only be burned once, please don't click "Burn/Program" to burn before the configuration information is confirmed. Non-I2C sensors can be tested by "Run in RAM", before burning "Run in RAM" can be used indefinitely. But the Beacon needs to be completely power off(VCC or coin cell) each time before the second "Run in RAM" test, otherwise the Beacon would not be connected normally.

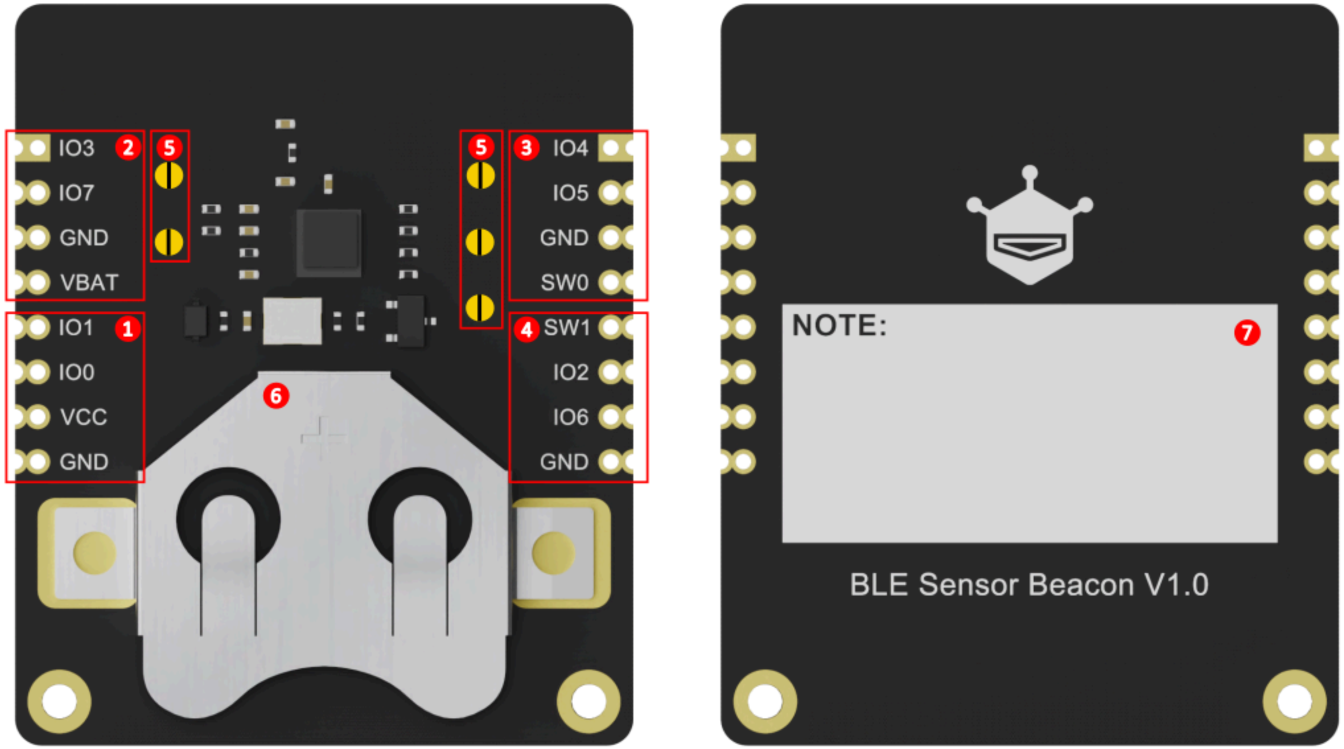
I2C sensors do not support the "Run in RAM" test can only be burned directly, it is recommended to directly use the sensor sample configuration file (https://github.com/DFRobot/DFRobot_FermionBLE) provided by DFRobot!To use an I2C sensor for which a profile is not provided, please check the Wiki below to use it.

Specification

- Operation Voltage： 1.1 ~3.6V DC
- Operation Current：9.1~13.8mA @2.4GHz TX mode-1Mbps
- Standby Current：0.625uA @Sleep with 32kHz RC, sleep timer
- Supported sensors: Digital/Analog/I2C sensors
- Input signals: digital/analogue, I2C
- Operating band: 2.4 GHz ISM
- Modulation: GFSK
- Transmit power: +5.0dBm
- PCB Size：35 * 42 mm

- Mounting Hole Size: inner diameter of 2mm/outer diameter of 4mm

Pinout



No	Name	Function
1	Serial port burning area	VCC：3.3V Input GND：Ground IO0：RX（3.3V) IO1：TX （3.3V)

No	Name	Function
2	I2C/Other sensors (coin cell powered)	VBAT：Coin Cell Positive IO3/IO7：Can be set to SDA/SCL or digital/analogue inputs and outputs
3	I2C/Other sensors (dynamic power control)	SW0：Configurable dynamic power supply that outputs a high level to power the sensor only when broadcasting IO4/IO5：Can be set to SDA/SCL or digital/analogue inputs and outputs
4	Digital/Analogue Sensors	SW1：Configurable dynamic power supply that pulls down only when broadcasting IO2：Can be set to SDA/SCL IO6：Can be set to digital/analogue input, this GPIO has a built-in voltage divider, recommended for analogue sensors.
5	Pull-up Resistor Selection Pad	IO3/IO7/IO4/IO5/IO2Pull-up resistor soldering point, connect the corresponding IO to the pull-up resistor, the default is not connected to the pull-up. Fermion and Gravity series sensors have pull-up resistors on the sensor side.
6	Battery socket	CR2032 button cell Connect positive to VBAT, negative to GND
7	NOTE	Beacon Information Marker to mark the key information of the burned beacons

Digital/Analog Sensors Tutorial

As an example of a custom data format, the sensor data is acquired through the mobile app and ESP32.

1. Requirements

- **Hardware**
  - TEL0168 Fermion: BLE Sensor Beaconx1
  - 3.3V USB-TTL convertor x1
  - Gravity: Analog LM35 Temperature Sensor (https://www.dfrobot.com/product-76.html) x1 or other analog sensors
  - Windows/Linux/Mac OS computer x1
  - ESP32
- **Software**
  - Recommended Mobile App： **NanoBeacon BLE Scanner**(IOS/Android (https://inplay-tech.com/nanobeacon-ble-scanner))
  - Beacon Config Tool： NanoBeaconConfigTool_V3.2.11 (https://inplay-tech.com/nanobeacon-config-tool)
  - Arduino IDE&ESP32 Setup： FireBeetle_ESP32_E Setup Tutorial (https://wiki.dfrobot.com.cn/_SKU_DFR0654_FireBeetle_Board_ESP32_E#target_5)

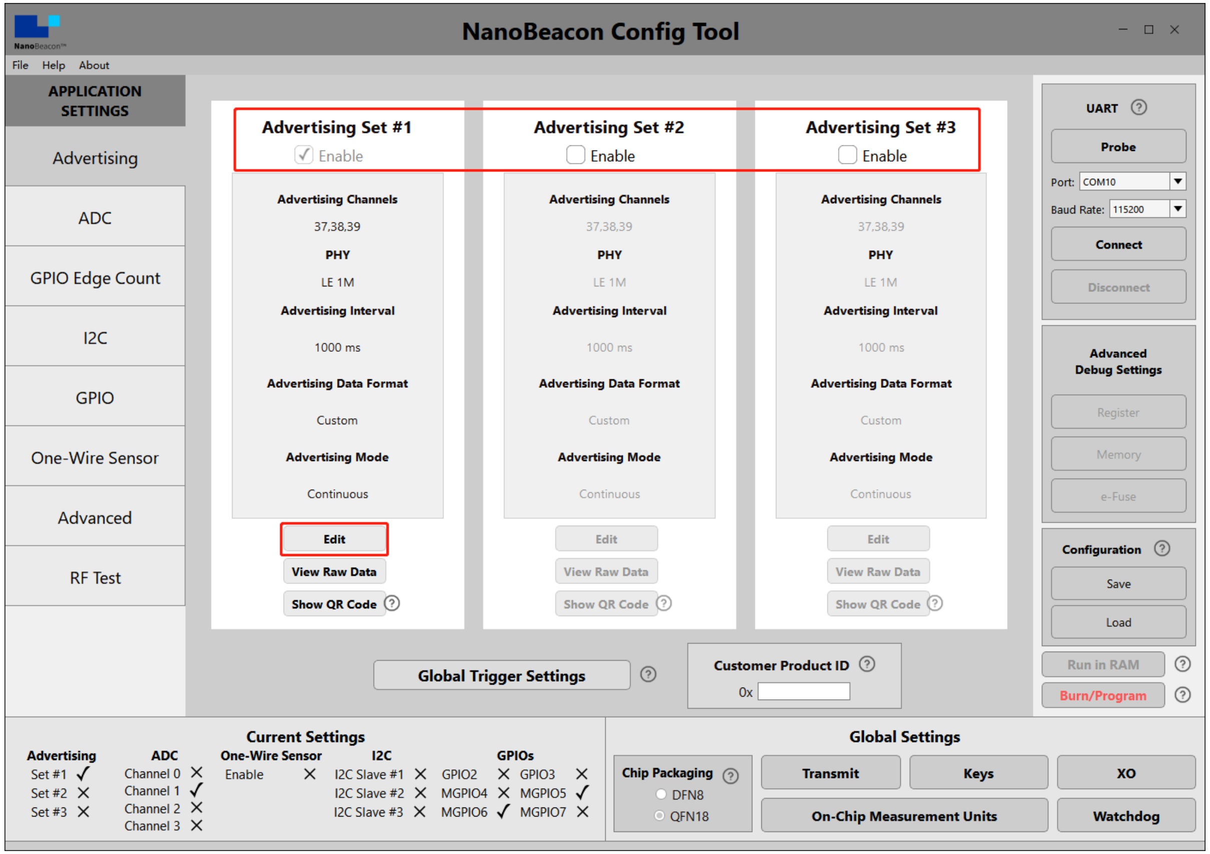
2.Configure Sensor Beacon

*Note: The module can only be burned once, do not click "Burn/Program" directly to burn before confirming the configuration information. You can test the module by "Run inRAM", and "Run in RAM" can be used for unlimited times before burning, and the system will be reset after power failure.

The input voltage of the module's ADC interface must not exceed 1.6V. If the voltage of the accessed sensors may exceed 1.6V, the sensors need to be connected to GPIO6, which adds 2.06 times the divider resistor to support an input voltage of up to 3.3V.

- 1.DownloadNanoBeaconConfigTool_V3.2.11 (https://inplay-tech.com/nanobeacon-config-tool)， Run NanoBeaconConfig.exe
- **2.Advertising**

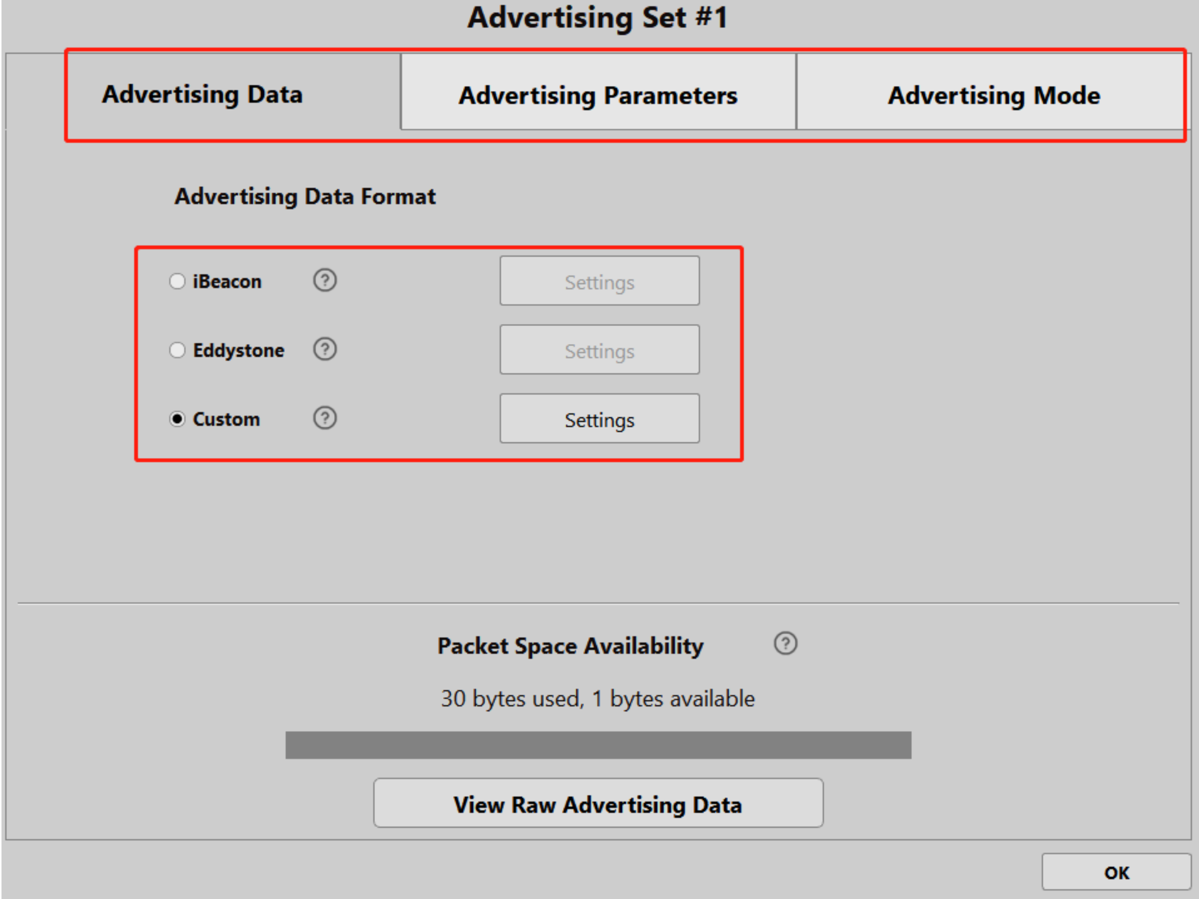
  Fermion: BLE Sensor beacons can be set to three broadcast channels, check Enable to open the corresponding broadcast channel, the default is to open one, Edit to enter the configuration page.



(https://img.dfrobot.com.cn/wiki/5cabf4771804207b131ae8cb/2fa011bdb3168dacac6d038e4ee98082.png)

- **3.Advertising Set#1 - Edit - Avdertising Data**

Three data formats can be set: iBeacon, Eddystone and Custom. In the tutorial, we will mainly use Custom.

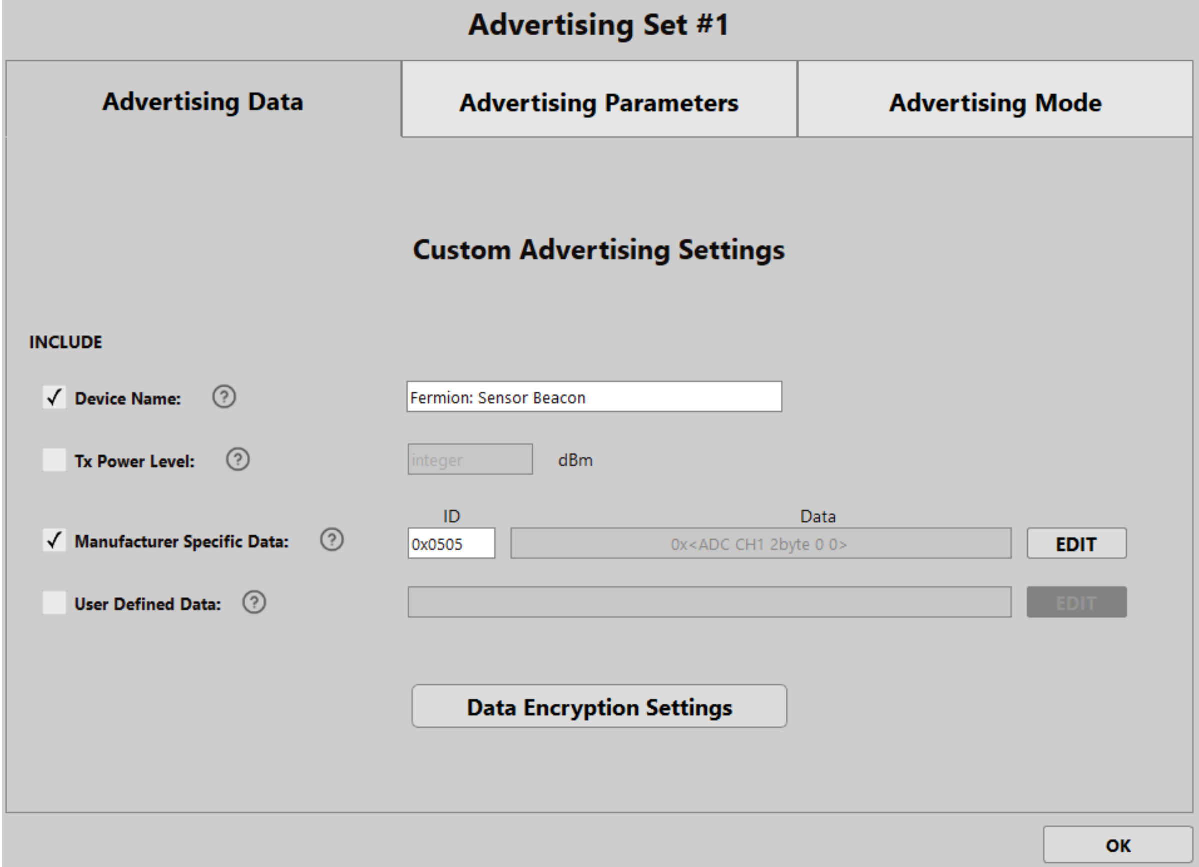


(https://img.dfrobot.com.cn/wiki/5cabf4771804207b131ae8cb/bc11abe621246af8634c89ea0edd247d.png)

- **4.Advertising Set#1 - Edit - Avdertising Data - Custom Settings**

  Tick "Device Name", enter "Fermion: Sensor Beacon", the name can be any one, mobile phones and ESP32 scanning can be directly based on the name of the filter.

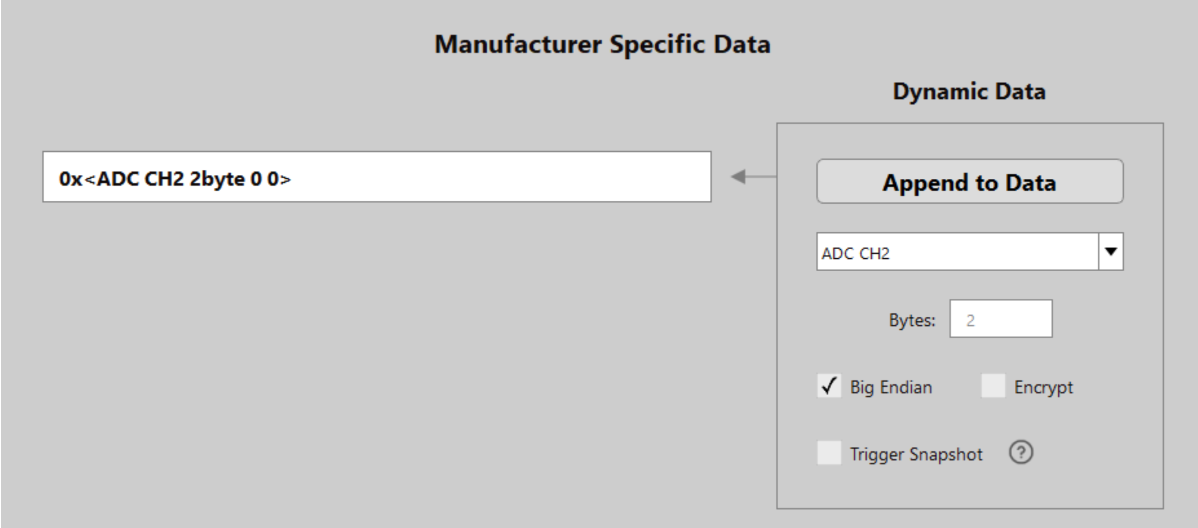
  Tick "Manufacturer Specific Data", click "EDIT" to configure the data.



(https://img.dfrobot.com.cn/wiki/62b2fb5caa613609f271523c/d87b89626484dd97ec072b7ca99b97d7.png)

- **5.Advertising Set#1 - Edit - Avdertising Data - Custom Settings - EDIT**

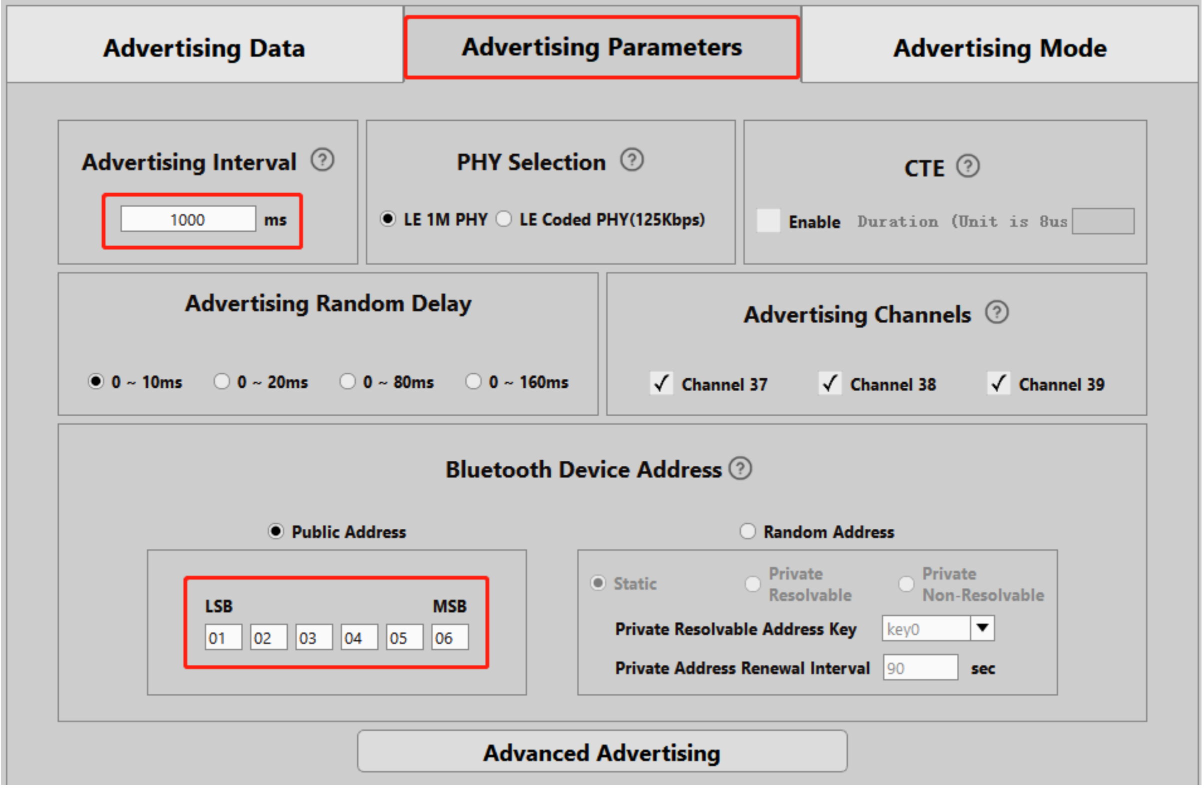
  Here only configure an analogue data, drop-down box select "ADC CH1", check: "Big Endian", click "Append to Data", you can see in the window "0x<ADC CH1 2byte 1 0>", click OK to exit!



(https://img.dfrobot.com.cn/wiki/62b2fb5caa613609f271523c/d584996b72a91cdf1634681539bc6fff.png)

- **6.Advertising Set#1 - Edit - Avdertising Parameters**

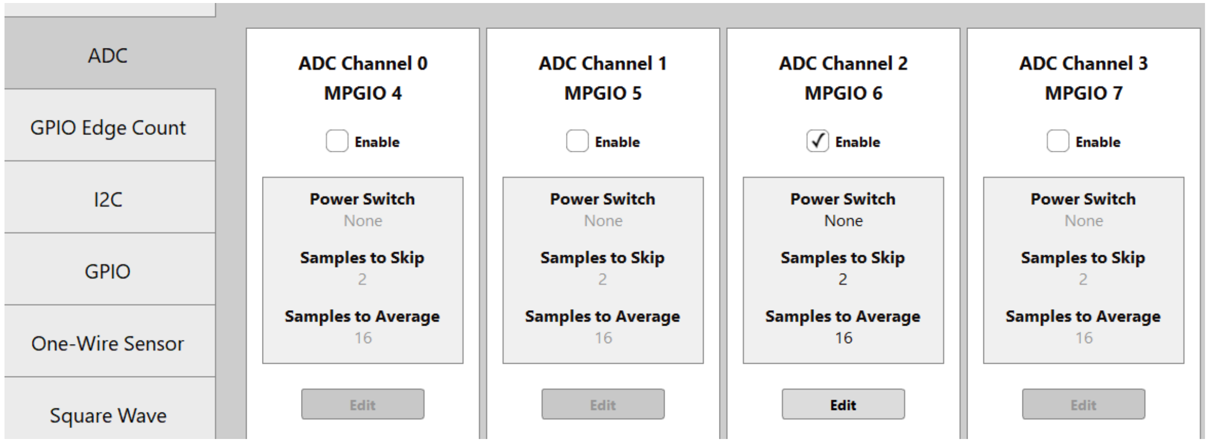
  Here to set the broadcast interval and address, according to the need to modify can be completed after the completion of the OK exit, so that the broadcast data format configuration is complete, the module will be 1S / time broadcast data



(https://img.dfrobot.com.cn/wiki/5cabf4771804207b131ae8cb/f65bcb4a4d943212f5ecc9d7b9a1c13f.png)

- **7.ADC**

Next, do the ADC related configuration, Fermion: BLE sensor beacons use IO6 for analogue acquisition, so Enable "ADC Channel 2 MPGIO 6" in the ADC screen and click on Edit to configure it



(https://img.dfrobot.com.cn/wiki/62b2fb5caa613609f271523c/132799f5956c8bb8b4ebe0af4954f6b9.png)

- **8.ADC - ADC Channel 1 MPGIO 6 - Edit**

A review of the IN100 datasheet shows that the IN100 chip used in the Fermion: BLE Sensor Beacon can only receive a maximum of 1.6V at the ADC.

4.4.1. On-chip VCC monitoring

Measured at: Ta = 25°C, VCC=3.0V, unless otherwise noted (if On-chip VCC monitoring is used, VCC should not be greater than 3.0V).

Table 6 : VCC monitoring characteristics

Parameter	Test conditions	Min.	Typ.	Max.	Unit
Resolution	Using on-chip VOP8 as reference		1.85		mV/LSB
Range	Input to ADC = 0.4*VCC. Input range of ADC is 0V - 1.6V (FS).	1		3.6	V
Accuracy	With VREF calibration only	-3.3	1.1	3.3	%
	With ADC offset and VREF calibration	-0.9	0.3	0.9	%

(https://img.dfrobot.com.cn/wiki/62b2fb5caa613609f271523c/6731753b451a42b3f13bbfcb51fef1de.png)

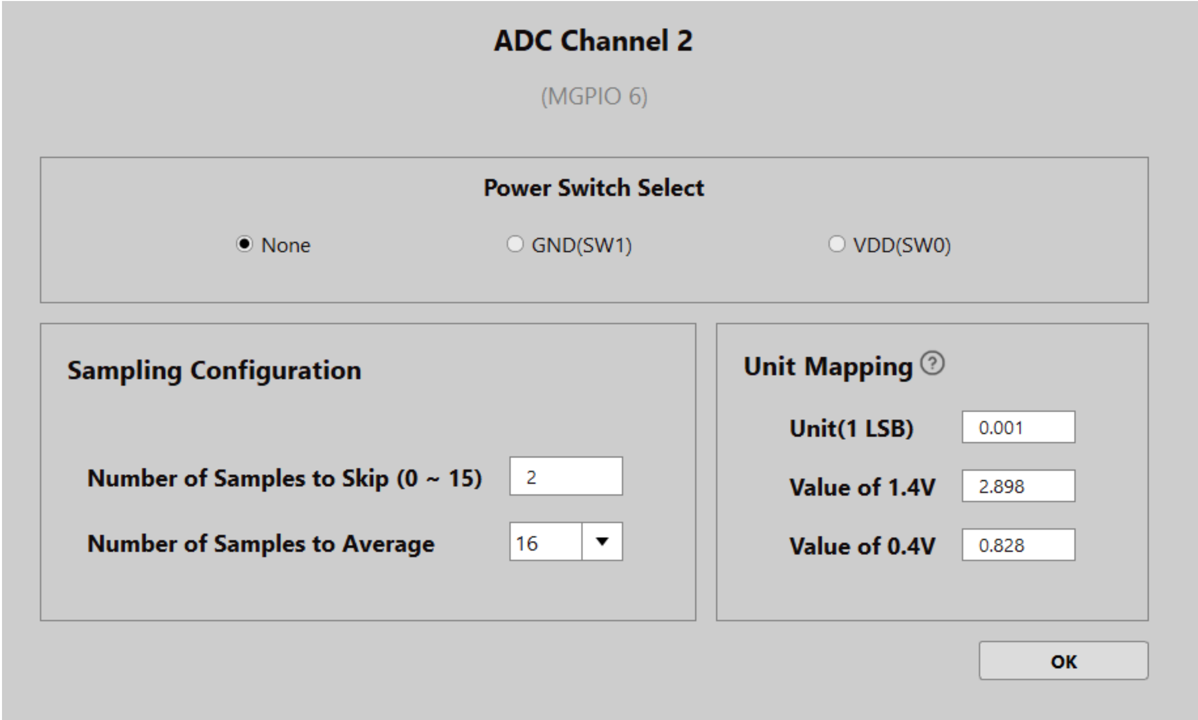
Since the output voltage of many analogue sensors will reach his supply voltage (often 3.3V). So we put a resistor in series with GPIO6 to do the voltage divider 2.06. Here we will modify the relevant configuration in the ADC settings

Modifying the Unit to 0.001 is easy to calculate and has little effect on the accuracy, but can be left unchanged if there is a very high demand for accuracy. Next, we need to remap the voltage value after voltage division.

Value of 1.4V Revised to 2.898

Value of 0.4V Revised to 0.828

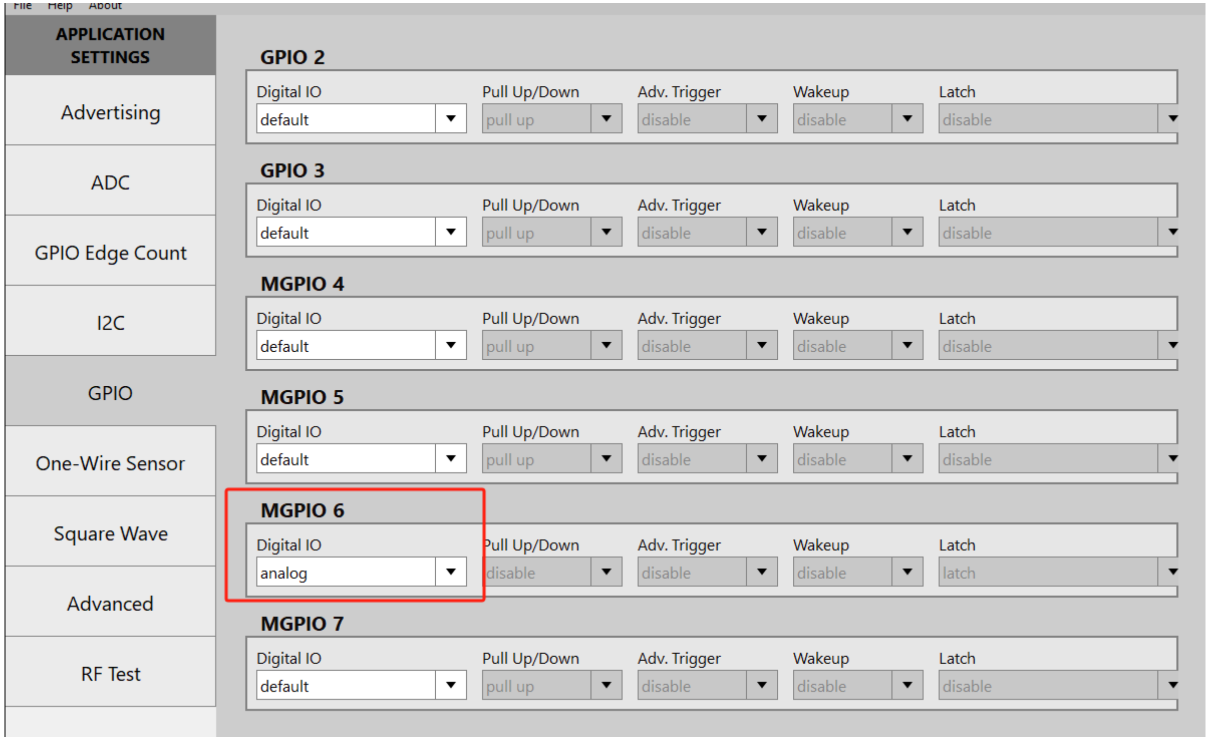
At this point, the ADC acquisition related configuration is completed, at this time the beacon broadcast data will be "sensor signal input" voltage value, unit mv



(https://img.dfrobot.com.cn/wiki/62b2fb5caa613609f271523c/e703d2263eabf085cb39c6377815e41d.png)

- **9.GPIO**

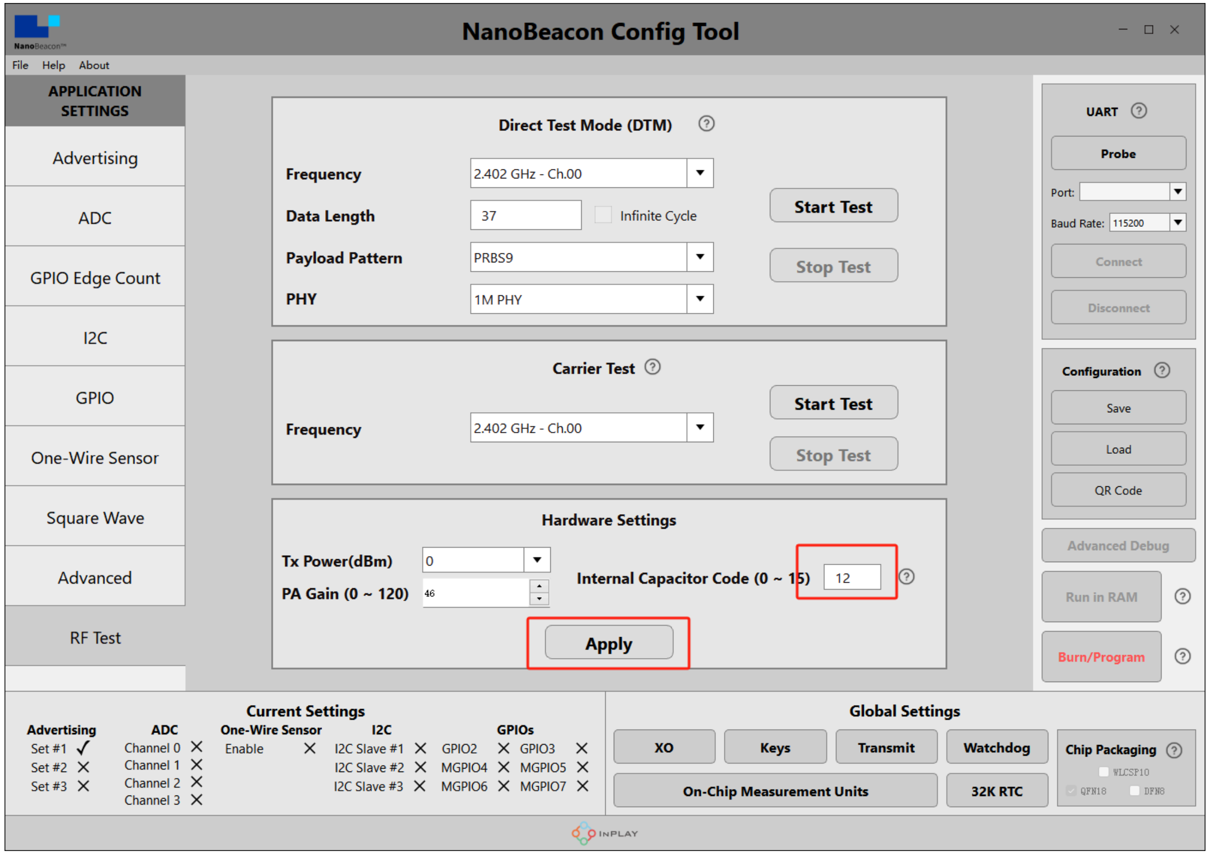
  Since MGPIO 6 is used as an ADC input, it needs to be configured as "analog".



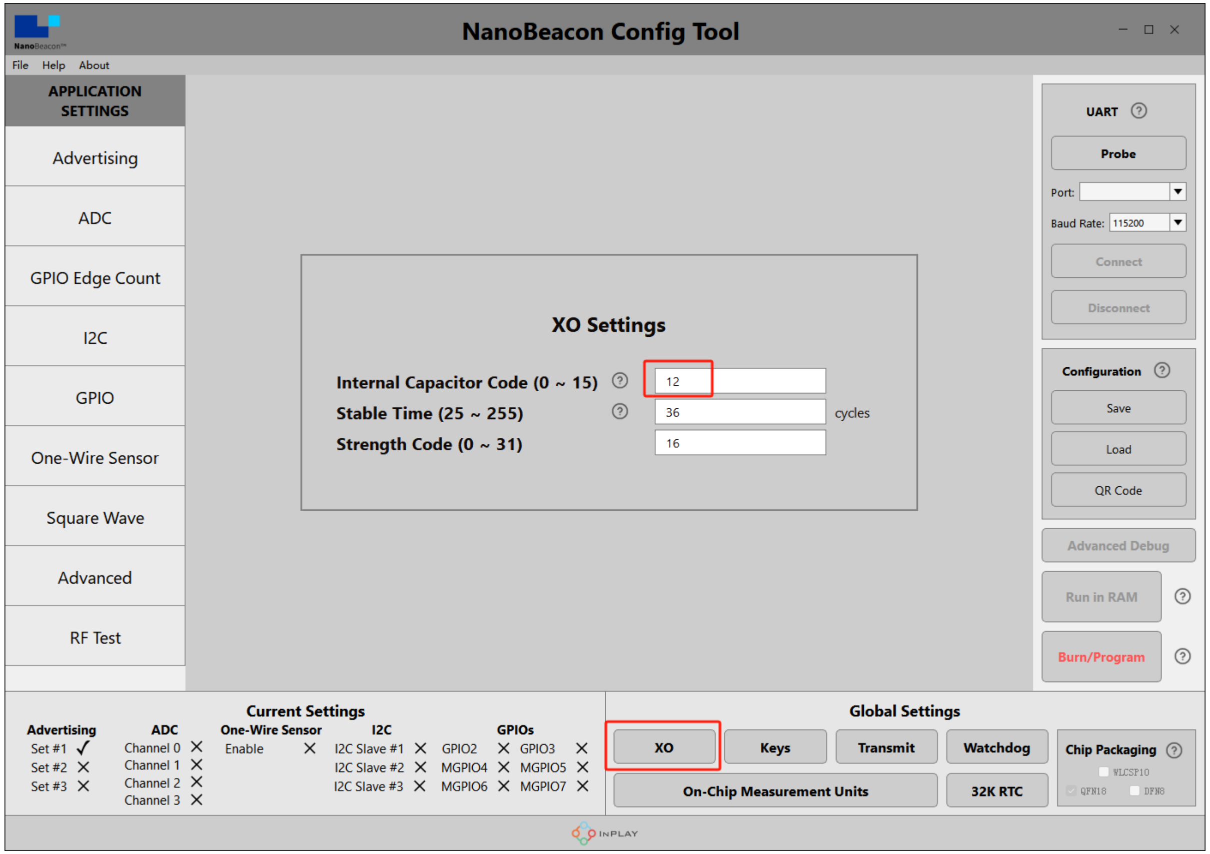
(https://img.dfrobot.com.cn/wiki/62b2fb5caa613609f271523c/b37d3af4ce87456d8f874e58bb84d7b1.png)

- **10.Crystal Capacitance Matching**

The NanoBeaconConfig Tool can be set to match the crystal capacitance, and in conjunction with our circuit, in order to keep the frequency bias at an optimal level, we recommend that you change the following two parameters to 12.



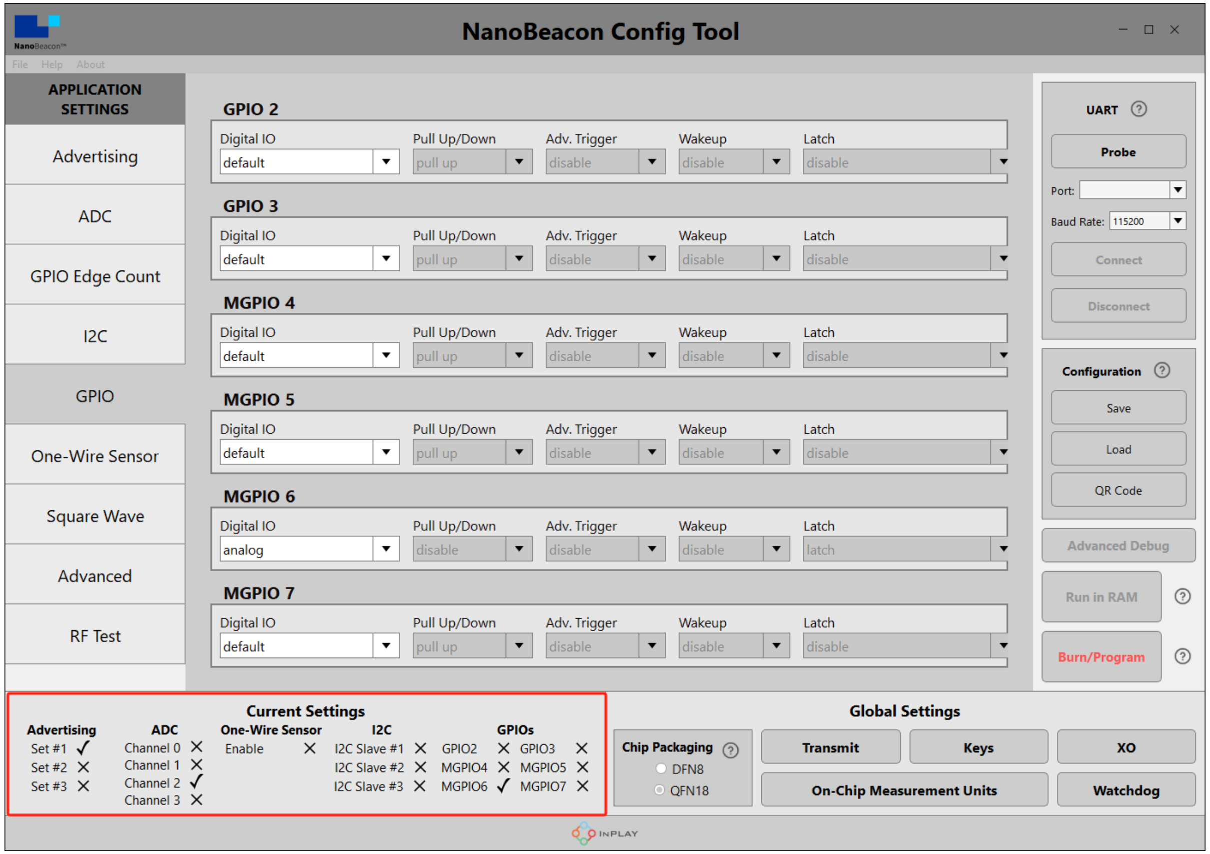
(https://img.dfrobot.com.cn/wiki/62b2fb5caa613609f271523c/6b45f4a6dbe685ae2392096d4813002d.png)



(https://img.dfrobot.com.cn/wiki/62b2fb5caa613609f271523c/602428247045a63ea5ac83d26b44f4d0.png)

- **11.Check Configuration**

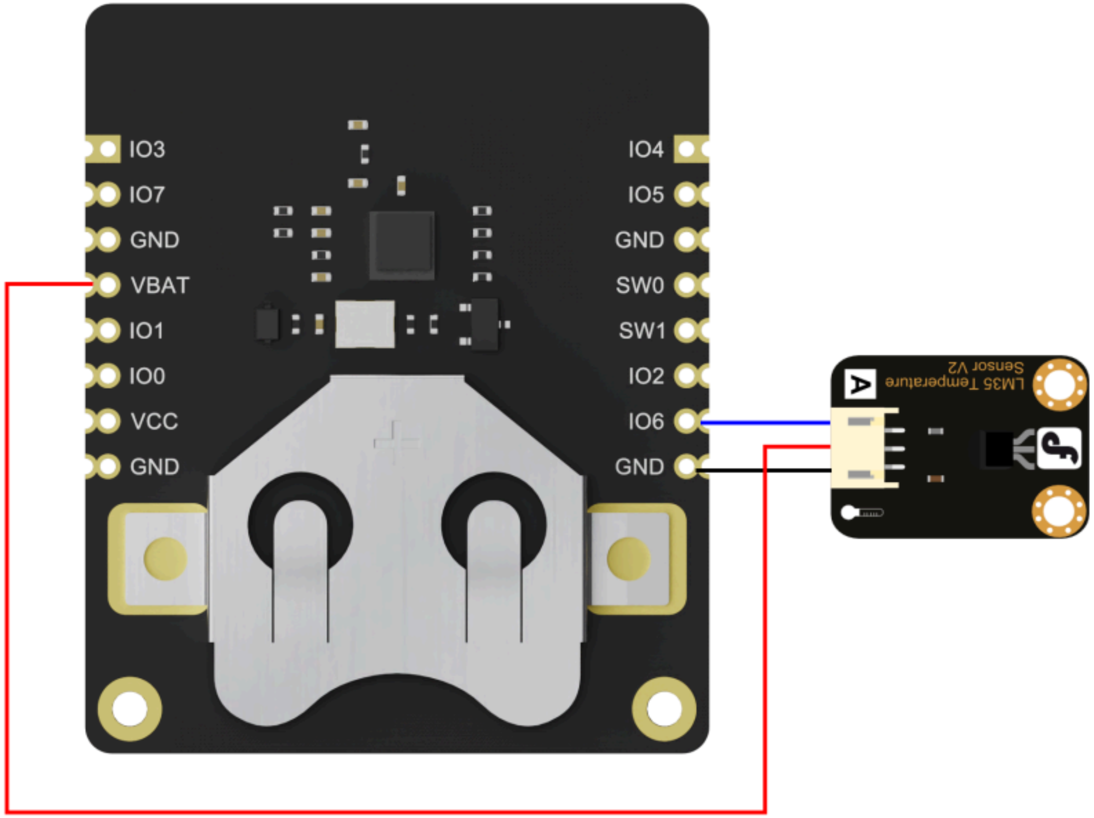
In the lower left corner of the software, you can see that we have enabled Set #1, ADC Channel 2 and MGPIO6.



(https://img.dfrobot.com.cn/wiki/62b2fb5caa613609f271523c/c66b5edb7c31bfed7ae87b36f7c4c381.png)

- **12.Hardware connection**

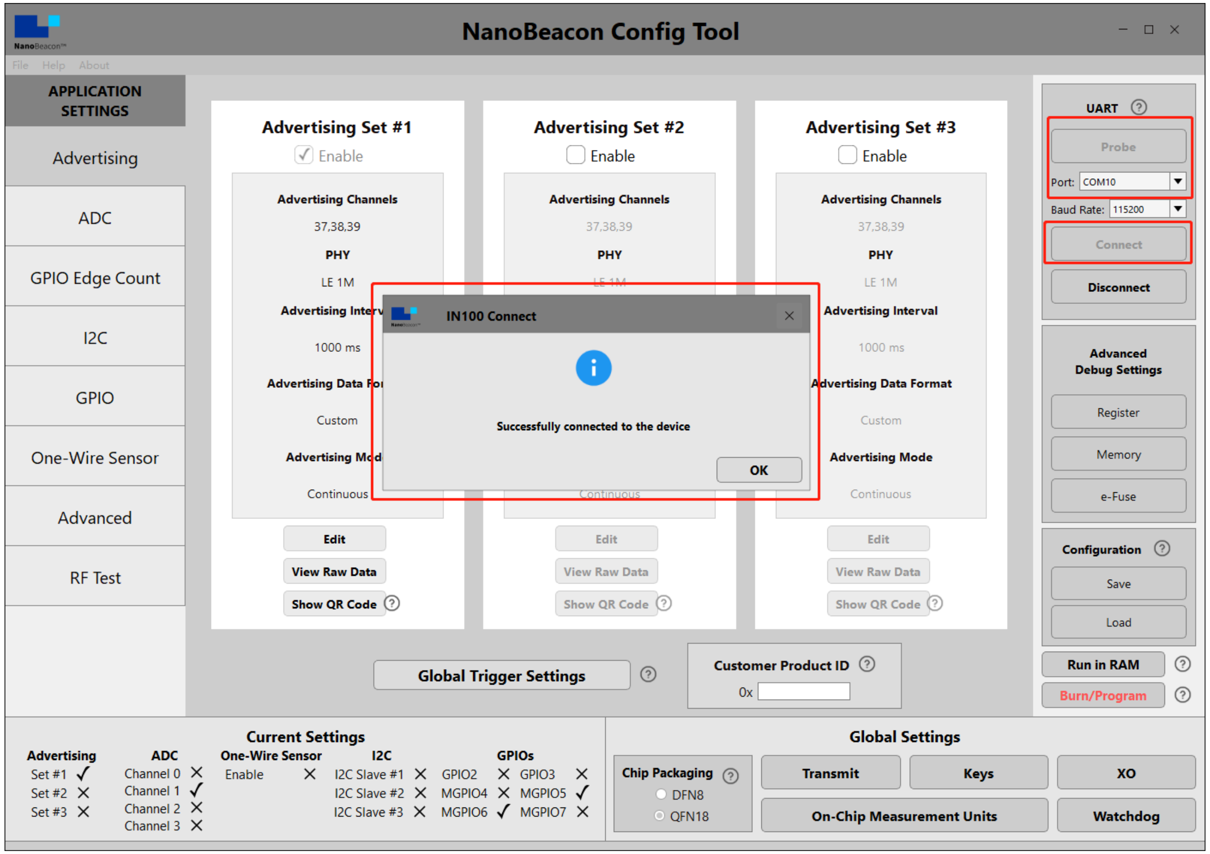
Hardware connections according to the wiring diagram



(https://img.dfrobot.com.cn/wiki/62b2fb5caa613609f271523c/33fedfc17c0798f5b024a23726ab08e5.png)

- **13.Connecting the module to the PC**

  In the upper right corner of the software, click "Probe" to refresh the port, after refreshing, select the corresponding port, click "Connect", there will be a pop-up window after successful connection.

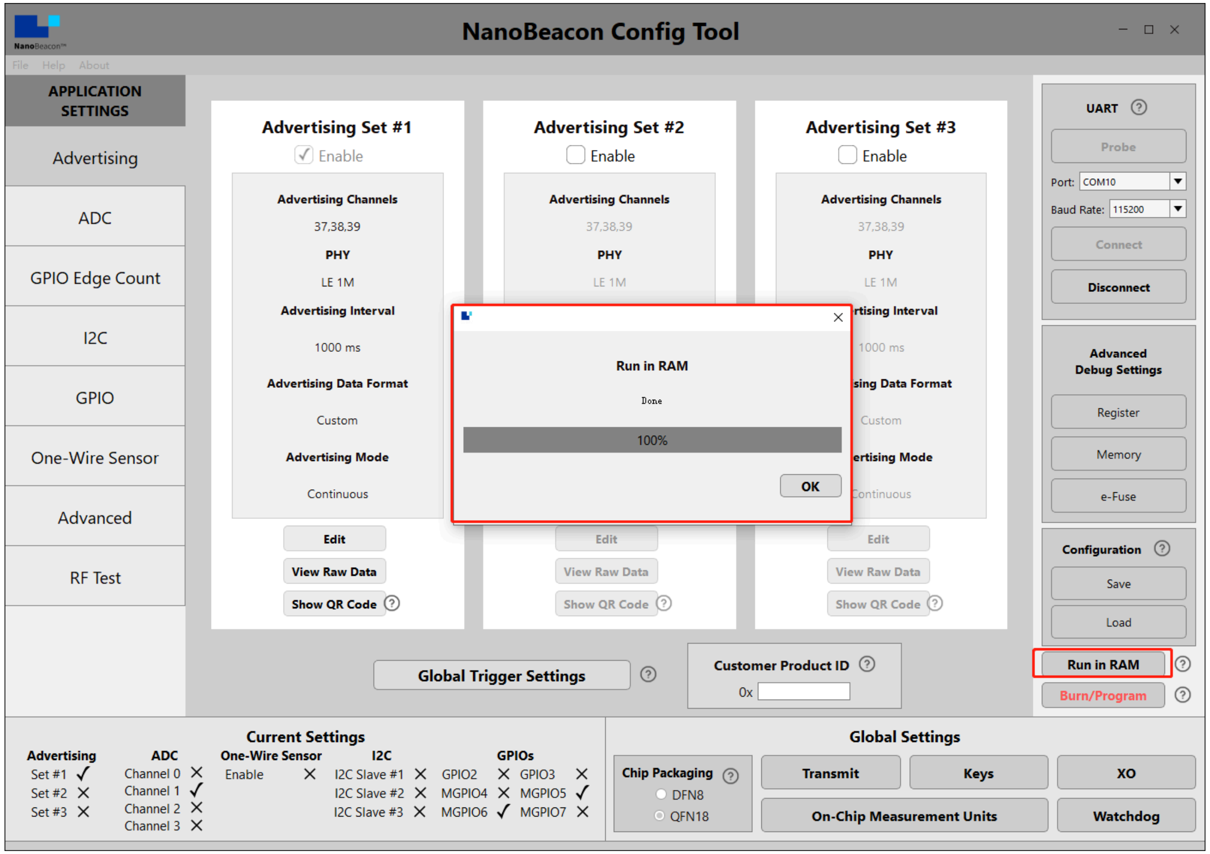


(https://img.dfrobot.com.cn/wiki/5cabf4771804207b131ae8cb/7ed488a2803226c9768e011fe07f0796.png)

- **14.Run test**

Click on "Run in RAM" and a pop-up will appear when it's done.

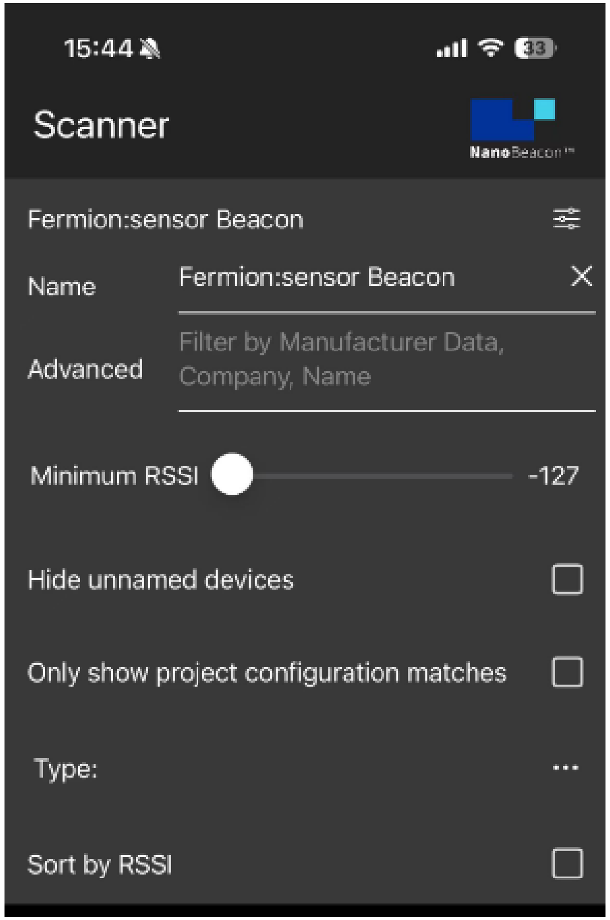
*Note: The module can only be burned once, do not click "Burn/Program" directly to burn before confirming the configuration information. You can test the module by "Run in RAM", and "Run in RAM" can be used for unlimited times before you click "Burn".But before your second "Run in RAM" test the Beacon will need to be completely power off(Both VCC and cell coin battery) .Otherwise, the Beacon could not be connected.



(https://img.dfrobot.com.cn/wiki/5cabf4771804207b131ae8cb/87fa6243339626fba059862288503e7e.png)

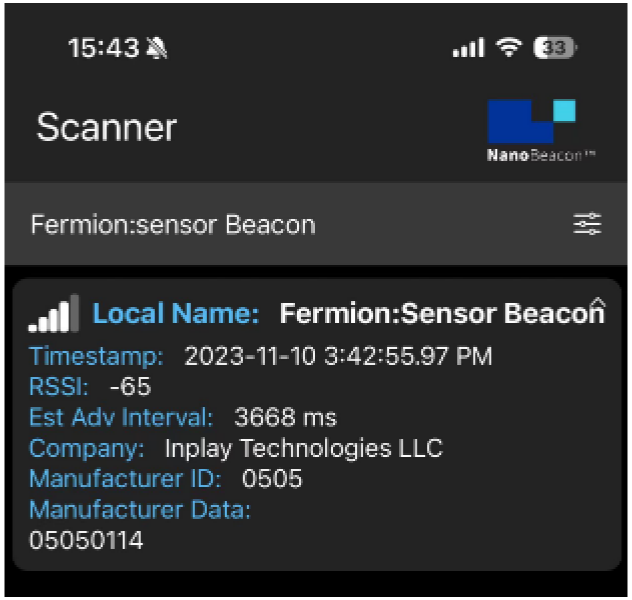
3. Mobile app to get data

- 1.Take an IOS device for example, AppStore install and open InPlay
- 2.If there are too many other beacons in the neighbourhood to find our device, we can enter the device name of the beacon in the filter, in the tutorial for configuring sensor beacons step 4, we named the Device Name "Fermion: Sensor Beacon".



(https://img.dfrobot.com.cn/wiki/62b2fb5caa613609f271523c/f8fcd4c827995e9407ad4825127657d6.png)

- 3.You can see that only "Fermion: Sensor Beacon" remains in the menu, click on it to see the details.



- **4.Data interpretation**

  "Fermion: Sensor Beacon" is the Device Name set in step 4 of the Sensor Beacon configuration.

  "06:05:04:03:02:01" is the address set in step 6 of configuring sensor beacons

  In "05050114", 0505 is the manufacturer's number and 0114 is the ADC acquisition data set in step 5 of configuring the sensor beacon

- **5.Sensor Data Calculation**

  Currently known beacon collected sensor data for "0X0114", will be converted to decimal 276, that is, the beacon collected the voltage value of 276mv

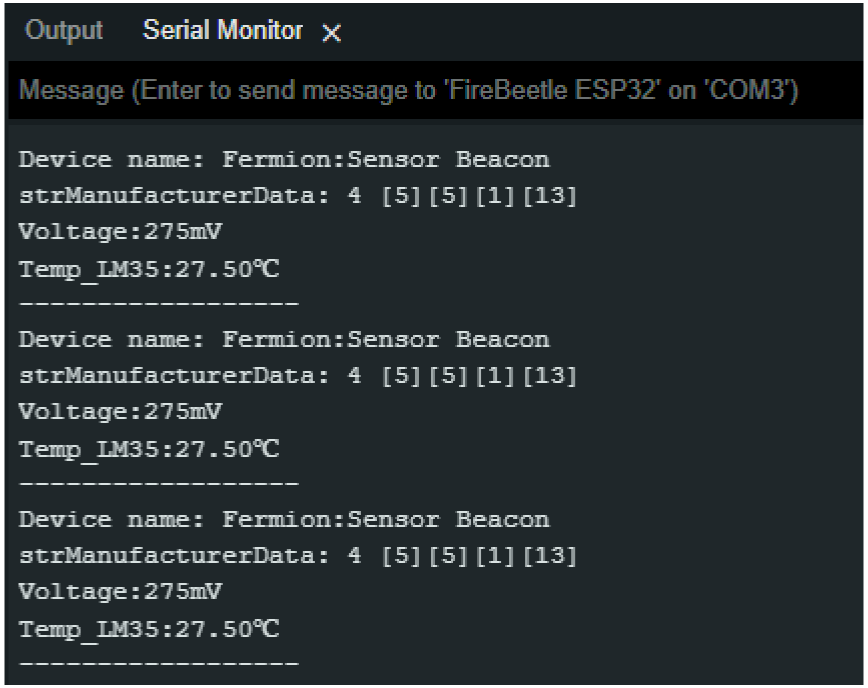
  The sensor we connected is LM35, by checking the Datasheet of LM35 (https://image.dfrobot.com/image/data/DFR0023/DFR0023_Datasheet.pdf), we know that the correspondence between LM35 output voltage and temperature is 10mV/°C, that is, the data of LM35 temperature sensor broadcasted by the beacon is 27.6°C.

4. ESP32 Get Data

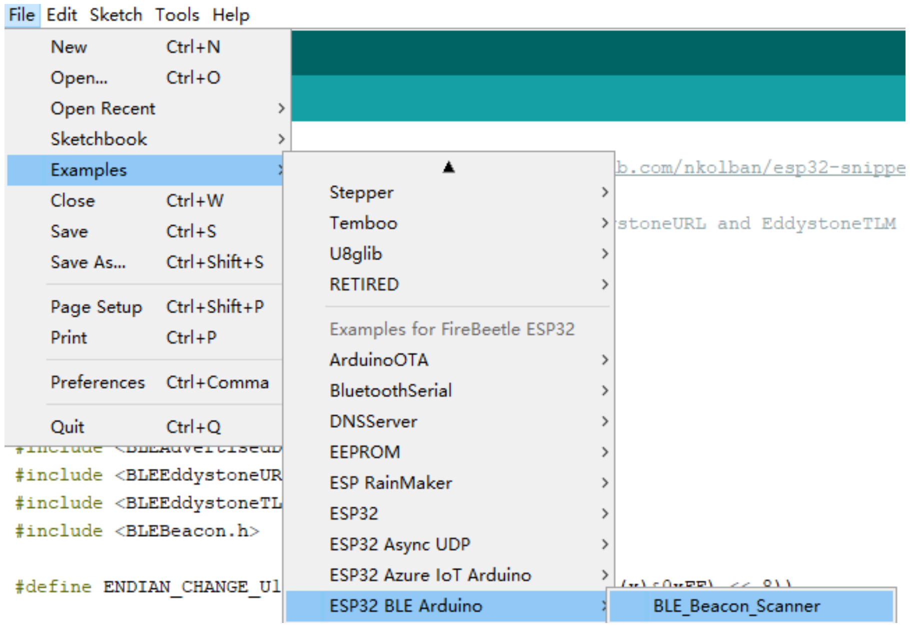
- Prepare the Arduino IDE&ESP32set up：FireBeetle_ESP32_E Set up (https://wiki.dfrobot.com/FireBeetle_Board_ESP32_E_SKU_DFR0654#target_7)
- Upload the following programme for the ESP32

```
/*
   Based on Neil Kolban example for IDF: https://github.com/nkolban/esp32-snippets/blob/maste
   Ported to Arduino ESP32 by Evandro Copercini
   Changed to a beacon scanner to report iBeacon, EddystoneURL and EddystoneTLM beacons by be
*/

#include <Arduino.h>
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEScan.h>
#include <BLEAdvertisedDevice.h>
#include <BLEEddystoneURL.h>
#include <BLEEddystoneTLM.h>
#include <BLEBeacon.h>
#define ENDIAN_CHANGE_U16(x) ((((x)&0xFF00) >> 8) + (((x)&0xFF) << 8))

float Sensor_Data;
int scanTime = 5; //In seconds
BLEScan *pBLEScan;

class MyAdvertisedDeviceCallbacks : public BLEAdvertisedDeviceCallbacks
{
    void onResult(BLEAdvertisedDevice advertisedDevice)
    {
      if (advertisedDevice.haveName())
      {
        if(String(advertisedDevice.getName().c_str()) == "Fermion:Sensor Beacon"){
          Serial.print("Device name: ");
          Serial.println(advertisedDevice.getName().c_str());
          std::string strManufacturerData = advertisedDevice.getManufacturerData();
          uint8_t cManufacturerData[100];
          strManufacturerData.copy((char *)cManufacturerData, strManufacturerData.length(), 0
          Serial.printf("strManufacturerData: %d ", strManufacturerData.length());
            for (int i = 0; i < strManufacturerData.length(); i++)
            {
```

```
                Serial.printf("[%X]", cManufacturerData[i]);
              }
              Sensor_Data = int(cManufacturerData[2]<<8 | cManufacturerData[3]);
              Serial.println();
              Serial.print("Voltage:");Serial.print(int(Sensor_Data));Serial.println("mV");
              Serial.print("Temp_LM35:");Serial.print(Sensor_Data/10);Serial.println("℃");
              Serial.println("------------------");
          }
        }
      }
};
void setup()
{
  Serial.begin(115200);
  Serial.println("Scanning...");

  BLEDevice::init("");
  pBLEScan = BLEDevice::getScan(); //create new scan
  pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
  pBLEScan->setActiveScan(true); //active scan uses more power, but get results faster
  pBLEScan->setInterval(100);
  pBLEScan->setWindow(99); // less or equal setInterval value
}
void loop()
{
  // put your main code here, to run repeatedly:
  BLEScanResults foundDevices = pBLEScan->start(scanTime, false);
  pBLEScan->clearResults(); // delete results fromBLEScan buffer to release memory
  delay(2000);
}
```



- This programme is modified from the BLE_Beacon_Scanner that comes with the ESP32, and can be modified as needed.



(https://img.dfrobot.com.cn/wiki/5cabf4771804207b131ae8cb/8fca136be65ff7bf9843495656a742bd.png)

5.Confirm the data and burn into Beacon

- *Note: The module can only be burned once, if you only want to test the function, you can skip this step.
- The above data format is broadcast in a customised format, intended to enable you to use it quickly, please refer to the software specification to configure yourself according to your needs for other formats.
- After confirming that the data is correct, the data format can be burned and cured into the chip.
- Click the "Burn/Program" button in the lower right corner to burn the program, and there will be a pop-up window prompting the success of burning.

I2C Sensor Tutorial

The custom data format is used as an example to get I2C sensor data via mobile app and ESP32.

1. Requirements

- **Hardware**
  - TEL0168 Fermion: BLE Sensor Beacon x1
  - 3.3V USB-TTL convertor x1
  - Fermion: SHT40 Temperature & Humidity Sensor (Breakout) (https://www.dfrobot.com/product-2437.html) x1 or other I2C sensor
  - Windows/Linux/Mac OS computer x1
  - ESP32
  - CR2032 battery*1
- **Software**
  - Recommended Mobile App： **NanoBeacon BLE Scanner**(IOS/Android (https://inplay-tech.com/nanobeacon-ble-scanner))
    - NanoBeacon BLE Scanner (https://inplay-tech.com/nanobeacon-ble-scanner)
  - Beacon Config Tool： NanoBeaconConfigTool (https://inplay-tech.com/nanobeacon-config-tool)
  - Arduino IDE & ESP32 Setup tutorial： FireBeetle_ESP32_E Setup Tutorial (https://wiki.dfrobot.com.cn/_SKU_DFR0654_FireBeetle_Board_ESP32_E#target_5)

- .cfg file/I2C sensor test sample code / ESP32 test sample code：Sensor Sample Configuration File (https://github.com/DFRobot/DFRobot_FermionBLE)

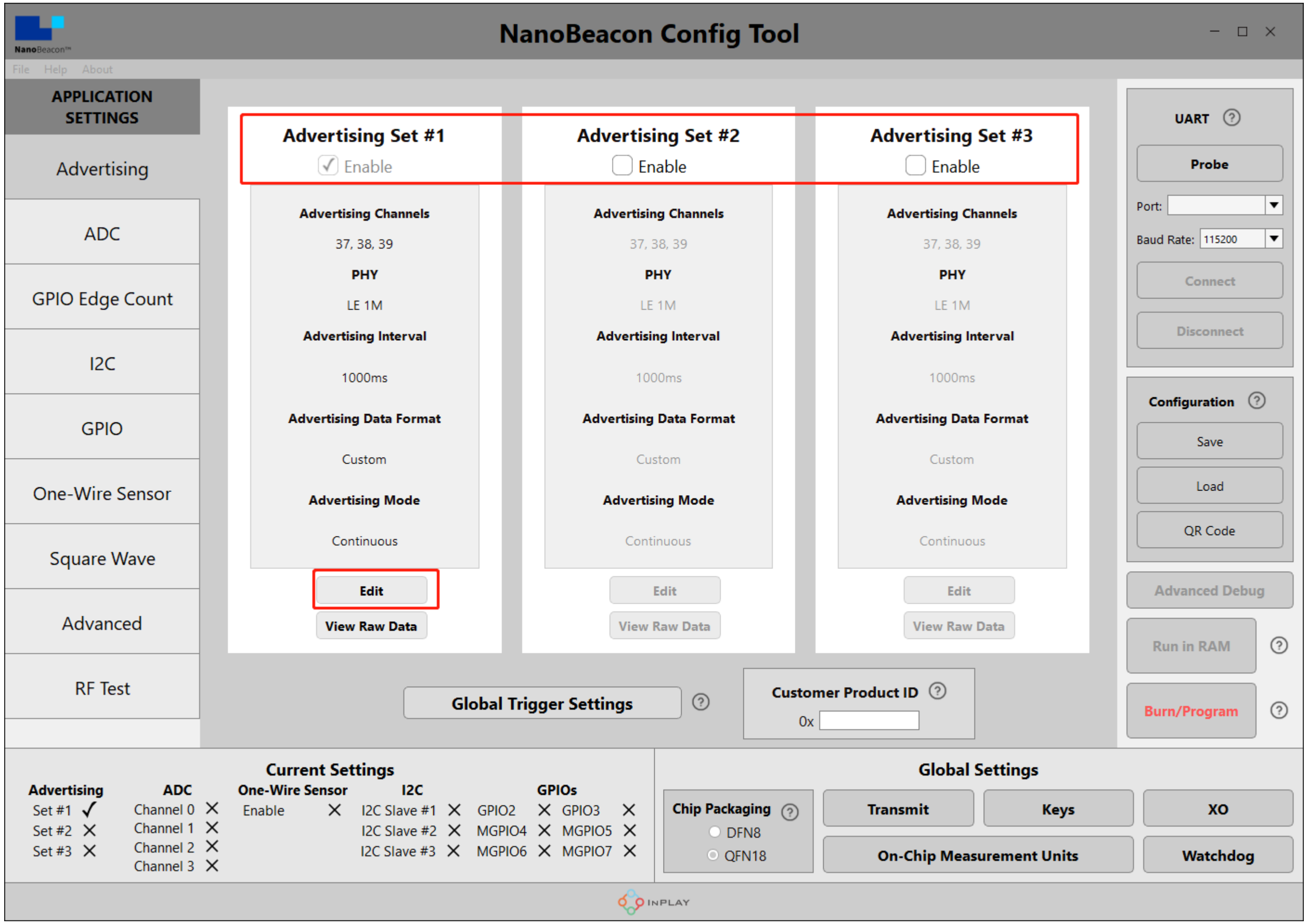
*Note: If the module is bricked after burning due to configuration file error, the user will be responsible for it. The module can only be burned once, please don't click "Burn/Program" to burn the module before confirming the configuration information, I2C sensors don't support "Run in RAM" test and can only be burned directly, it is recommended to use the Sensor Sample Configuration File (https://github.com/DFRobot/DFRobot_FermionBLE) provided by DFRobot. To use I2C sensors for which no profile is provided, please consult the tutorials in the Wiki.

2. Configuration of sensor beacon

- 1.DownloadNanoBeaconConfigTool (https://inplay-tech.com/nanobeacon-config-tool)，Run NanoBeaconConfig.exe

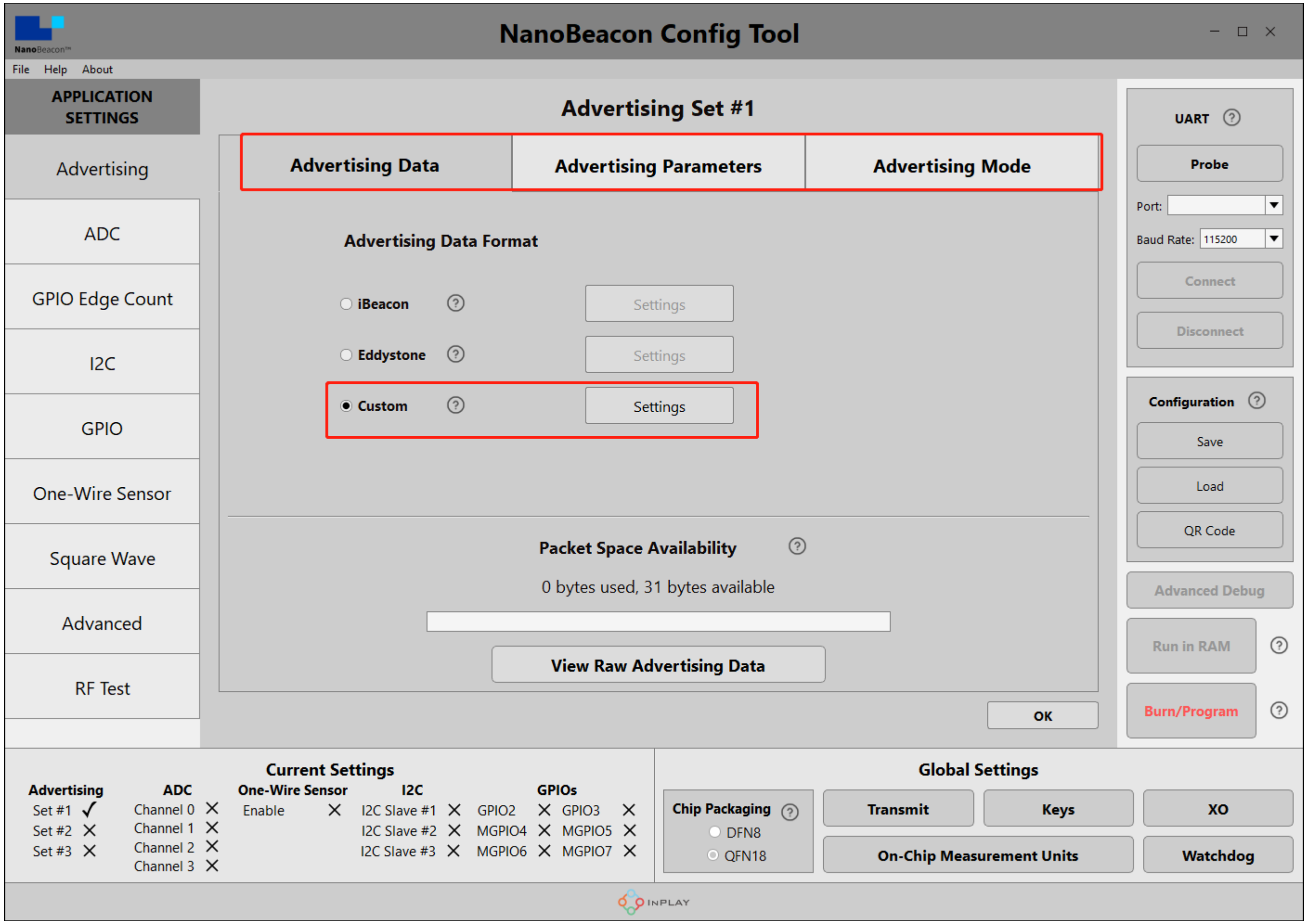
- **2.Advertising**

  Fermion: BLE Sensor beacons can be set to three broadcast channels, check Enable to open the corresponding broadcast channel, the default is to open one, Edit to enter the configuration page.



- **3.Advertising Set#1 - Edit - Avdertising Data**

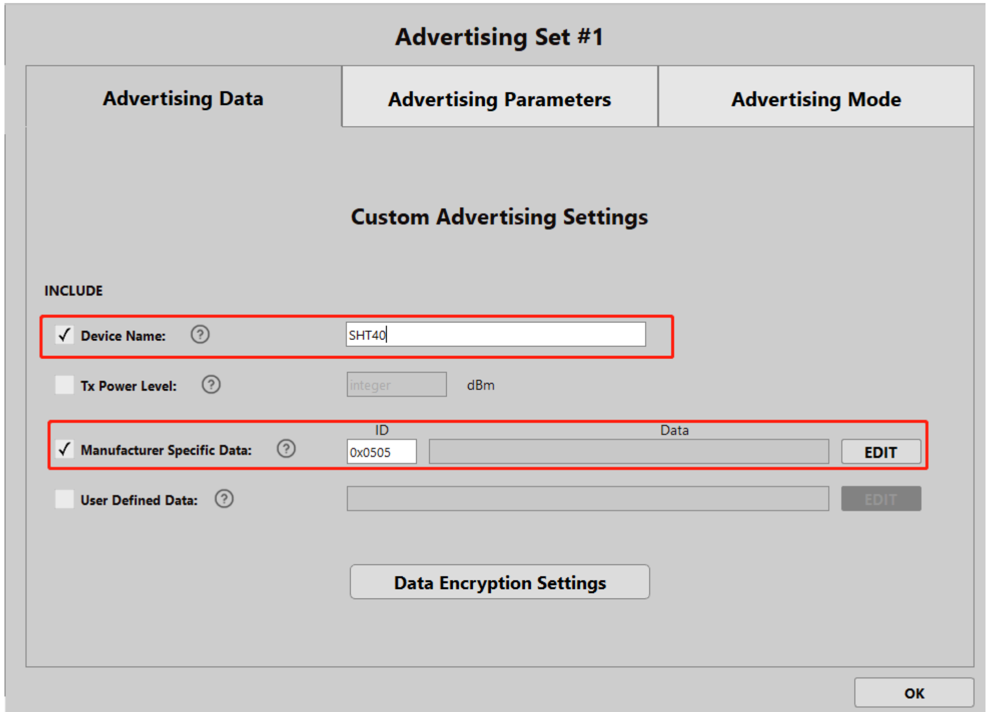
  Three data formats can be set: iBeacon, Eddystone and Custom. In the tutorial, we will mainly use Custom.



- **4.Advertising Set#1 - Edit - Avdertising Data - Custom Settings**

  Tick "Device Name", enter "SHT40", the name can be arbitrary, it is recommended that the length of 5 characters or less, the name is too long will occupy the data bits, mobile phones and ESP32 scanning can be directly based on the name of the screening

  Tick "Manufacturer Specific Data" and click "EDIT" to configure the data.



- **5.Advertising Set#1 - Edit - Avdertising Data - Custom Settings - EDIT**

Only one I2C data is configured here, select "I2C Slave #1 Read Data" in the drop-down box.

If you are using an I2C sensor that returns six or more bytes of I2C data frames in a single pass. You need to match Offset and byte for byte selection and rounding.

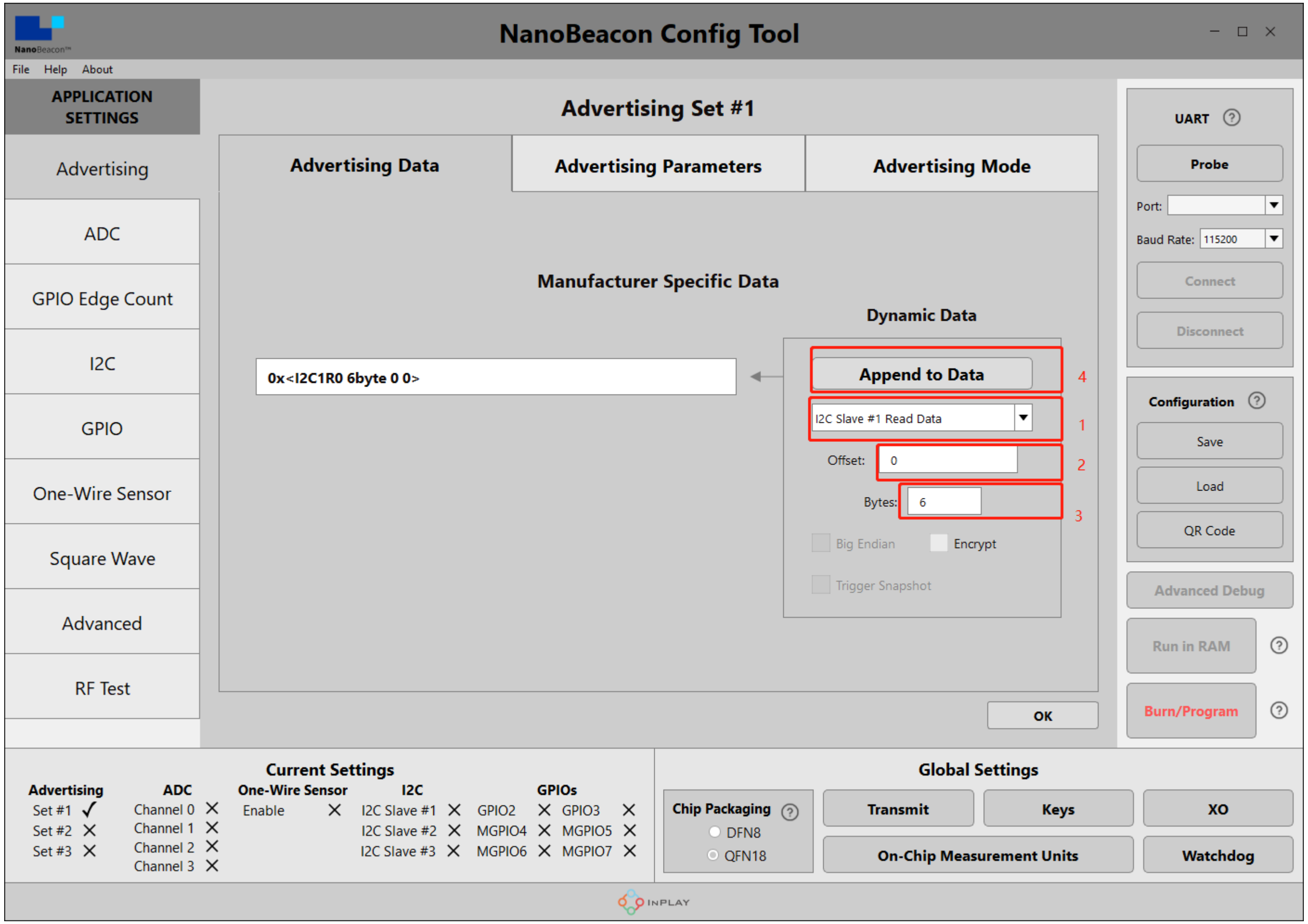
"Offset" Explanation:

Set the byte bias of the sensor I2C feedback, since the I2C feedback is a string of bytes. Sometimes the first few bytes of sensor feedback can be discarded to save the on-board IN100 data buffer.

If the sensor feedback: 00 00 06 FF, at this time the first two bytes do not have any significance, you can set Offset to 2 and discard the first two bytes.

"Bytes" Explanation：

Since the built-in IN100 chip's buffer uses a ring queue for data acquisition, while the buffer stores a maximum of 5 bytes at the same time. Therefore, the number of bytes read into the buffer is limited to 5 at a time. Then, click Append to Data at the top to see "0x<I2C1R0 5byte 0 0>" in the window, click OK to exit.

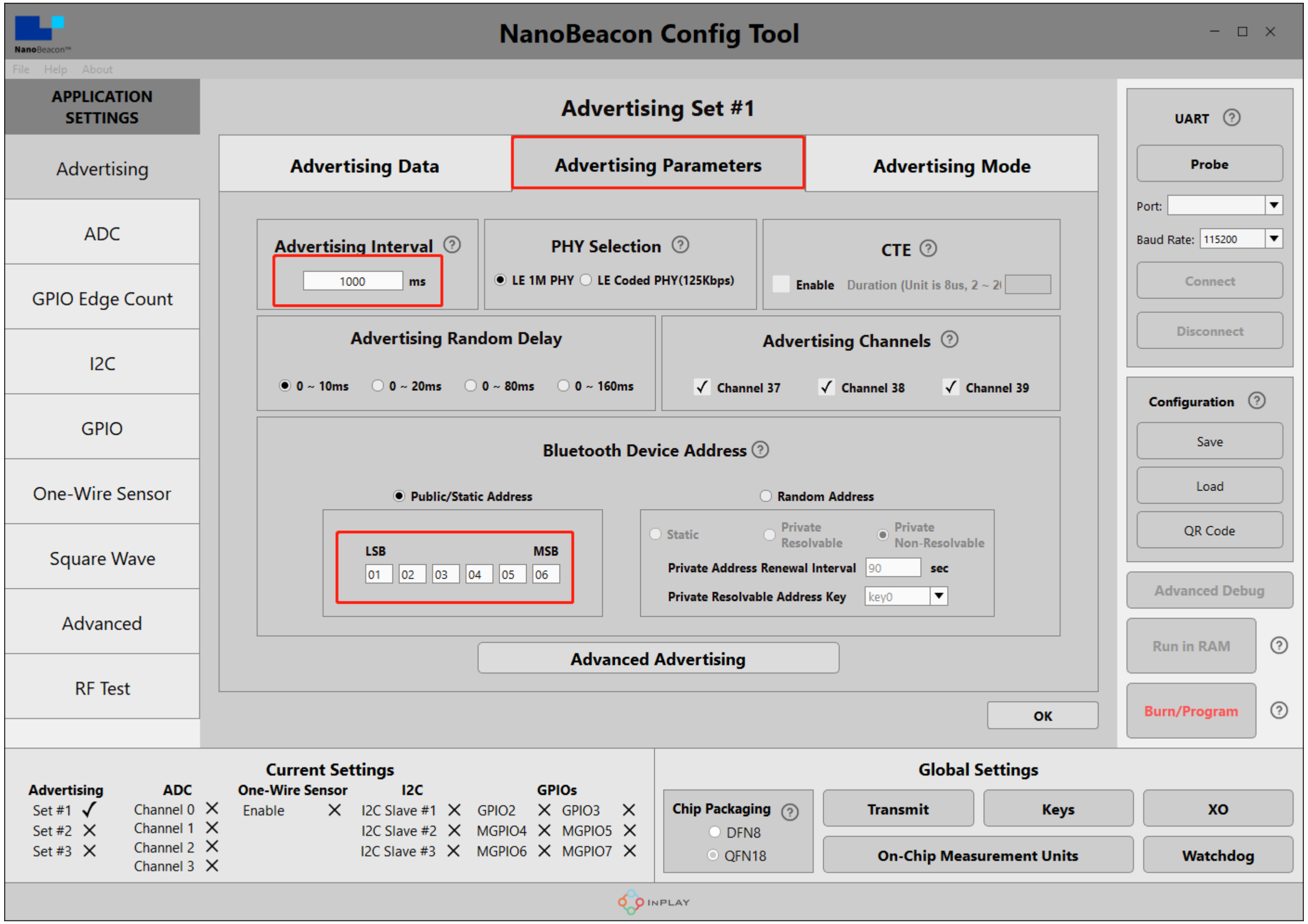


- **6.Advertising Set#1 - Edit - Avdertising Parameters**

Here set the broadcast interval and address, modify as needed, OK to exit when finished. Here the broadcast interval is set to 1000ms, that is, the module will broadcast data at 1S/time.

Advertising Interval is the time interval of auto broadcast.

Bluetooth Device address can set the address of Fermion: BLE sensor beacon. (LSB is the least significant bit, MSB is the most significant bit)



- **7.Communication setup ---- pre-testing with Arduino**

Since the I2C configuration of the Fermion: BLE Sensor Beacon can only be burned once, it is recommended to first test the I2C communication control in a controller such as an Arduino for correctness using code programming.

Use the Arduino UNO to upload I2C to read the code and confirm that the sensor is working correctly Wire diagram can refer to SHT40（SEN0428）wiki (hhttps://wiki.dfrobot.com/SHT40_Humidity_and_Temperature_Sensor_SKU_SEN0428#target_5)

Code is as followed：

```
#include <Wire.h>

#define SHT40_ADDRESS 0x44 // I2C address of the sensor, here 0x44 for SHT40
int l = 5;// Read Byte Length

void setup(){
  Serial.begin(115200);
  Wire.begin();
}

void loop() {

  Wire.beginTransmission(SHT40_ADDRESS);
  Wire.write(0xFD);//i2ctx:3
  Wire.endTransmission();//i2c null

  delay(10); // The programme waits 10ms for the SHT40 to get ready

  Wire.requestFrom(SHT40_ADDRESS, 6);
  if (Wire.available() >= l) {
    byte data[l];
    for (int i = 0; i < l; i++) {//Read the data output from the I2C sensor
      data[i] = Wire.read();
      Serial.print(data[i], HEX);
      Serial.print(" ");
    }
  Serial.println();
  }
  Wire.endTransmission();//i2c null
  delay(1000);
}
```

When a value appears in the serial monitor, it means that the SHT40 sensor is outputting normally and you can continue with the following configuration.

Code Explanation: In common I2C communication sensors, the communication flow tends to be:

Step 1, The master writes a read command to the I2C of the sensor ——>the master reads data to the sensor

It is the master control that writes 0xFD byte to the SHT40 sensor. Checking the datasheet shows that the 0xFD byte is an instruction for the SHT40 sensor to measure temperature and humidity.

Command bin	Command hex	Response length incl. CRC (bytes)	Description
1111 1101	FD	6	measure T & RH with high precision (high repeatability)

Step 2, the master waits for some time while the sensor prepares the data delay(10)； Indicates that the master waits for 10ms, check the datasheet to find out: SHT40 recommends a minimum wait time of 1ms, but here we usually set the wait time to 10ms to be on the safe side.

Waiting time	*tw*	between I2C commands	1	-	-	ms	minimal waiting time for I2C communication

Step three, the master requests data from the sensor Wire.requestFrom(SHT40_ADDRESS, 6);

It is the master requesting the SHT40 to read 6 bytes, checking the datasheet shows that this frame data: The 1st and 2nd bytes are temperature data, and the 3rd bit is CRC check. The 4th and 5th bytes are humidity data, and the 6th bit is CRC checksum.

4.2 Data type & length

I2C bus operates with 8-bit data packages. Information from the sensor to the master has a checksum after every second 8-bit data package.

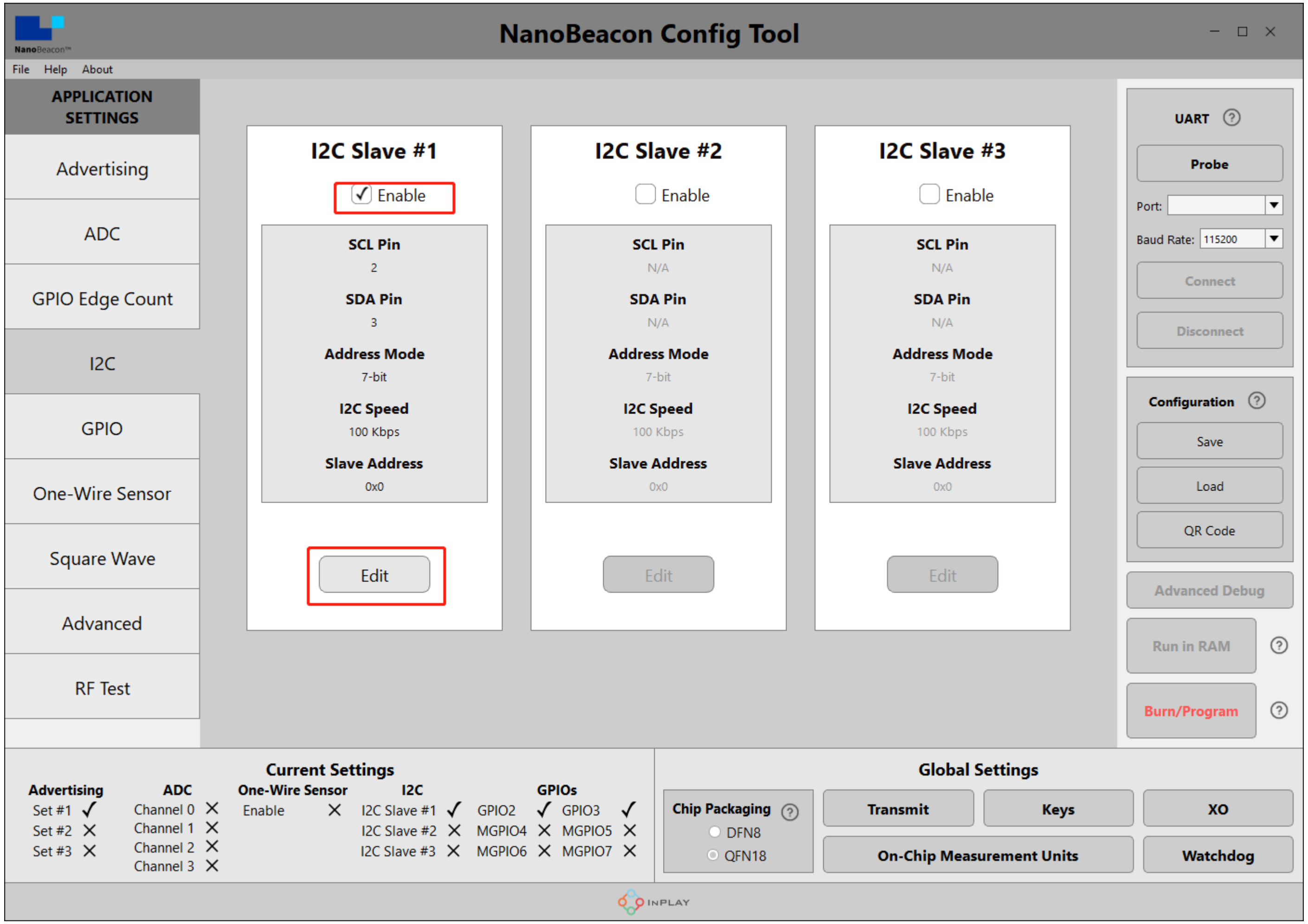
Humidity and temperature data will always be transmitted in the following way: The first value is the temperature signal (2 * 8-bit data + 8-bit CRC), the second is the humidity signal (2 * 8-bit data + 8-bit CRC).

- **8.I2C Communication Settings —— I2C Parameter Configuration**

  Next, I2C-related configuration is performed to enable the Fermion: BLE Sensor Beacon to acquire I2C sensor data.

  Once we are back in the Nano Beacon Config Tool and have selected the I2C tab on the left.

  There are three channels for I2C data acquisition, here we select channel 1 for I2C sensor configuration. Enable "I2C Slave#1" in the I2C interface and click Edit to configure.



According to the pin layout of the module, select SCL as MGPIO7 and SDA as GPIO3 in PIN Select. the connection between the BLE beacon and SHT40 will be more reasonable as shown in the figure below after this layout.

Slave Address is 0x44（SHT40 I2C Address）

Address Mode is 7 bit（I2C standard mode）

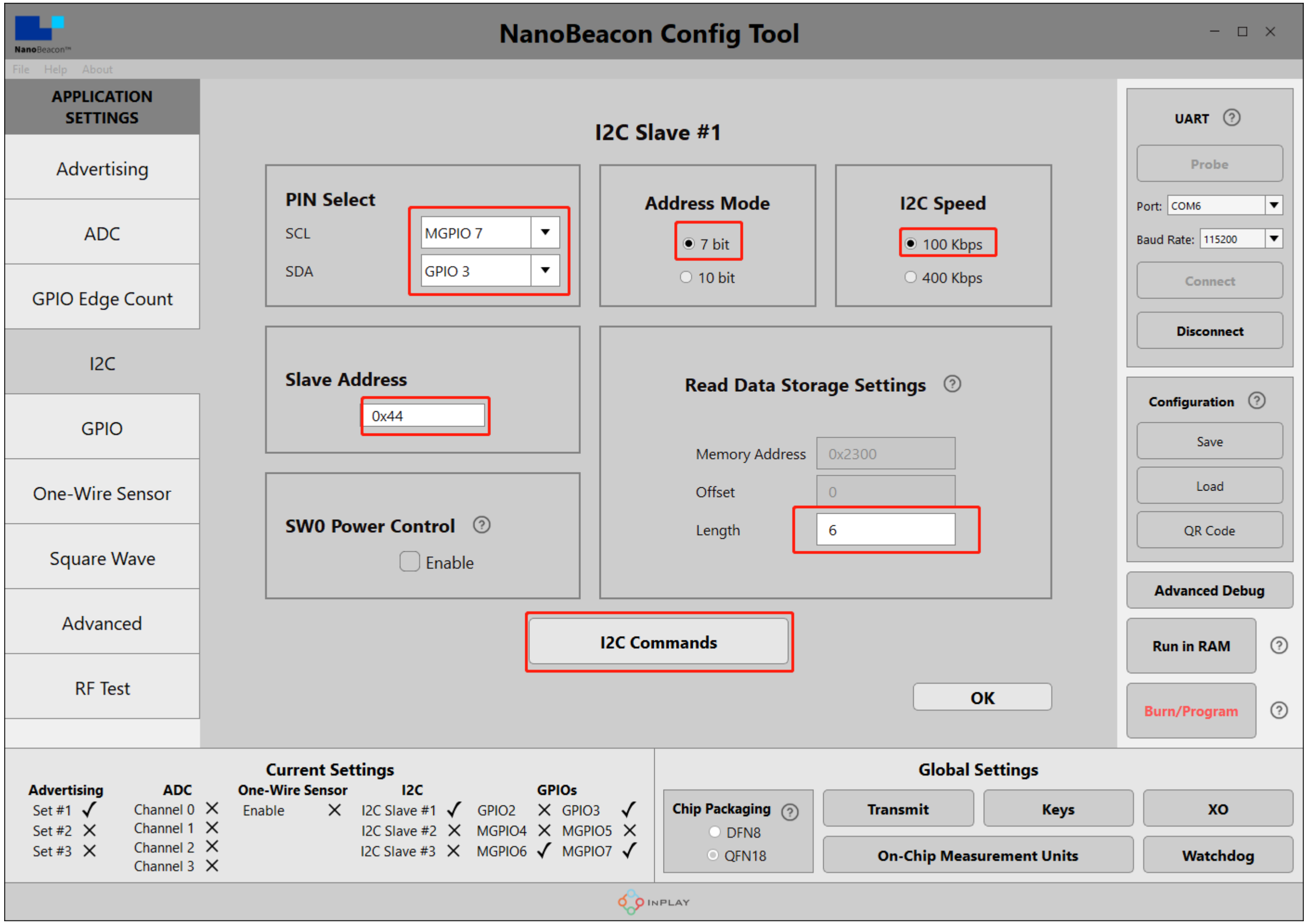
I2C Speed 为100kps（I2C standard speed）

Read Data Storage Settings set Length to 5. Here is the ring buffer mechanism, when read data length > Length, it will wrap back automatically. By looking at the SHT40 datasheet (https://sensirion.com/media/documents/33FD6951/640B22DB/Datasheet_SHT4x.pdf) it can be seen that when the master sends the command 0xFD, the sensor will return: [2 * 8-bit T-data; 8-bit CRC; 2 * 8-bit RH-data; 8-bit CRC], a total of 6 bytes.

However, due to the buffer settings of the beacon's on-board chip, **Fermion: Sensor beacons can only receive a maximum of 5 consecutive bytes after sending 1 command**，Combined with the data returned by SHT40, the last bit is the humidity check bit, which can be discarded, so the data obtained from SHT40 has a total of 5 bits, so Read Data Storage Settings sets Length to 5.

- **9.I2CCommunication Settings —— I2C Parameter Configuration**

Click I2C Commands to set commands.

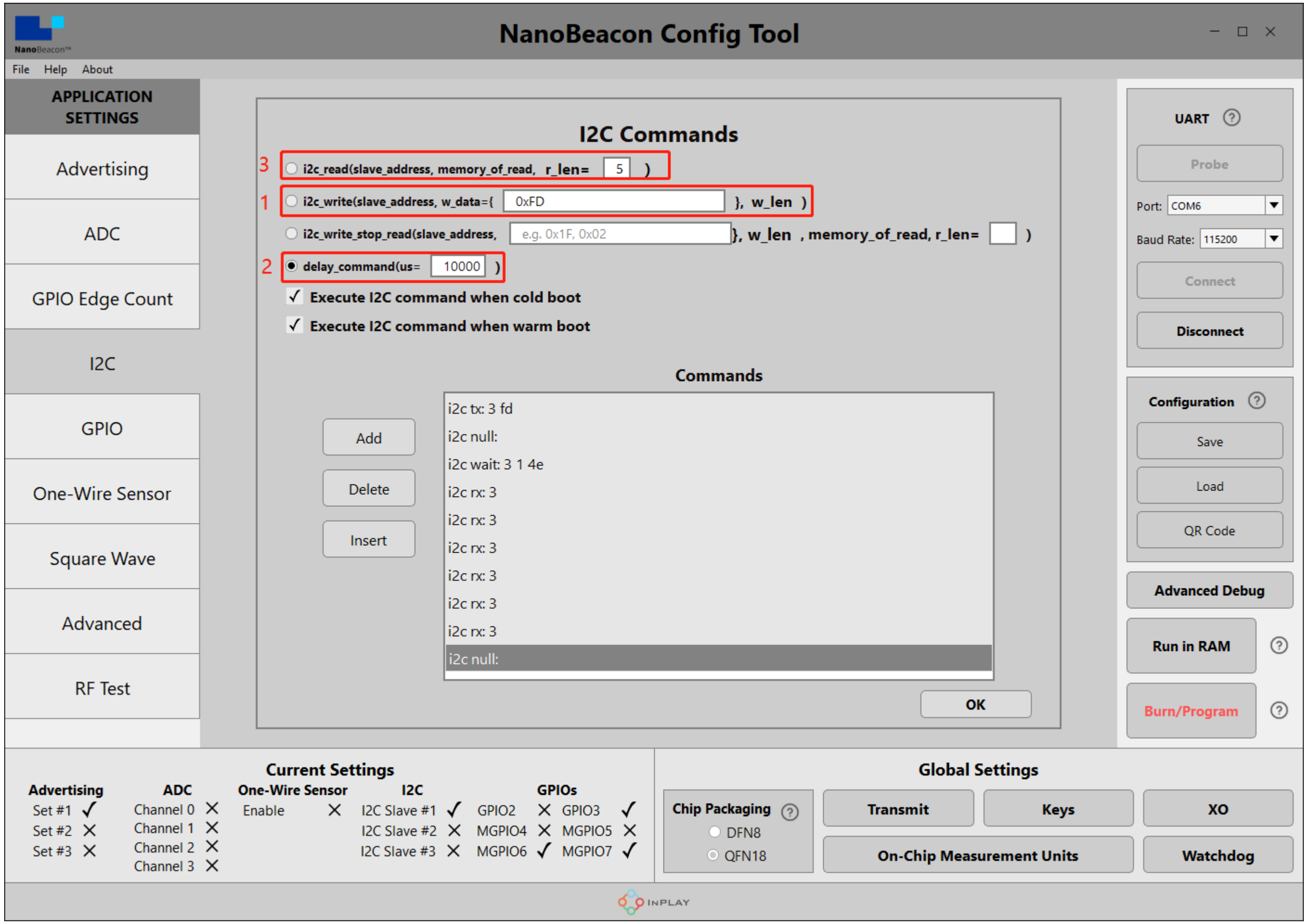


According to the I2C communication instructions in step 7, we first need to have the sensor beacon send the read command (0xFD) to the SHT40 chip: Tick Execute I2C command when cold boot as well as Execute I2C command when warm boot to select the line for i2c_write, fill in the command 0xFD that will read the SHT40 sensor, and then click Add to add it to the command list.

Next, we need a 10ms delay to wait for the SHT40 to prepare the data: We wait 10ms to select the line for delay_command, fill in the delay of 10000us, and click Add to add to the list of commands.

Finally, we need to read the temperature and humidity data from the SHT40: Select the line of i2c_read, fill in the length of data to be read as 5 (IN100 reads up to 5 bytes at a time), and then click Add to add to the command list.

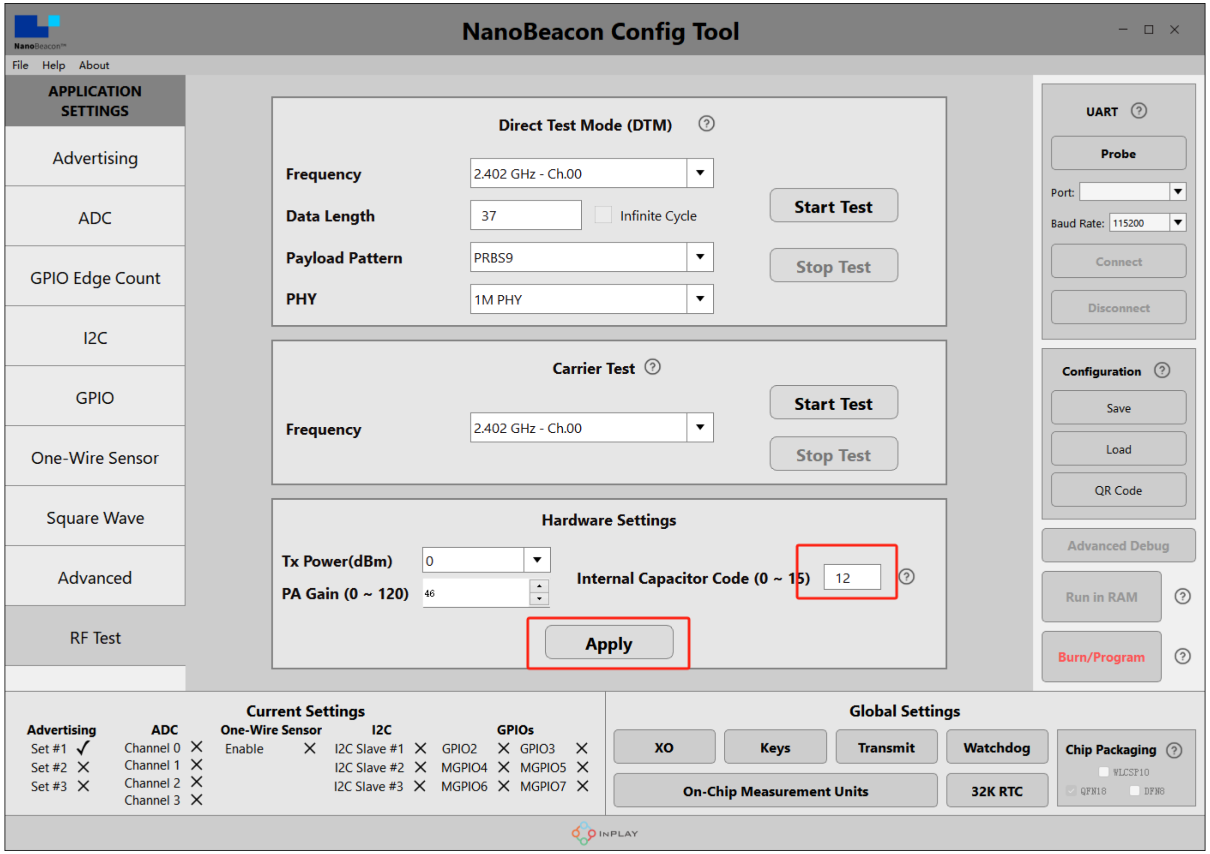
When the configuration is complete, the format should be consistent with the figure below.



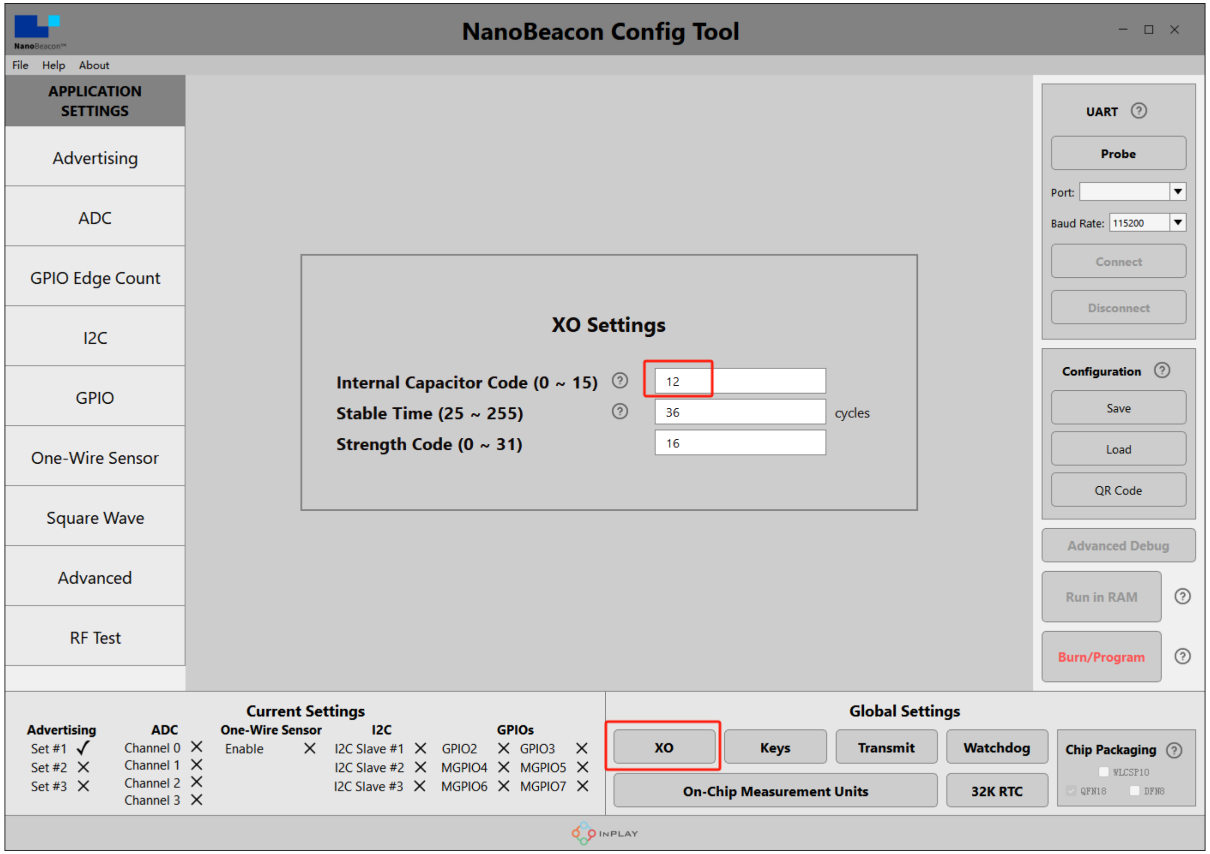
The above SHT40 samples can be downloaded and loaded directly：Fermion: BLE Beacon (https://github.com/DFRobot/DFRobot_FermionBLE)

- **10.Crystal Capacitance Matching**

The NanoBeaconConfig Tool can be set to match the crystal capacitance, and in conjunction with our circuit, in order to keep the frequency bias at an optimal level, we recommend that you change the following two parameters to 12.



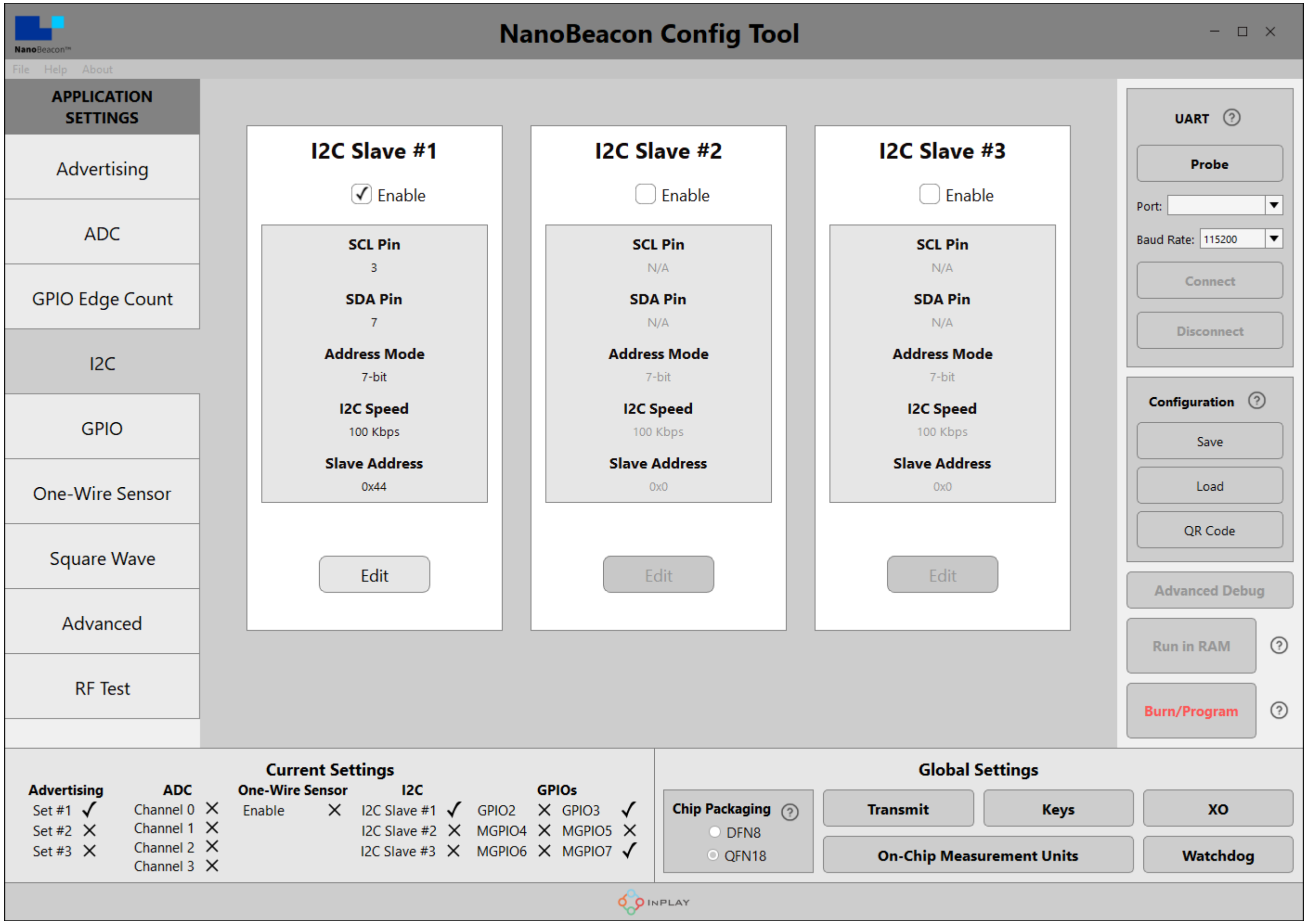
(https://img.dfrobot.com.cn/wiki/62b2fb5caa613609f271523c/6b45f4a6dbe685ae2392096d4813002d.png)



(https://img.dfrobot.com.cn/wiki/62b2fb5caa613609f271523c/602428247045a63ea5ac83d26b44f4d0.png)

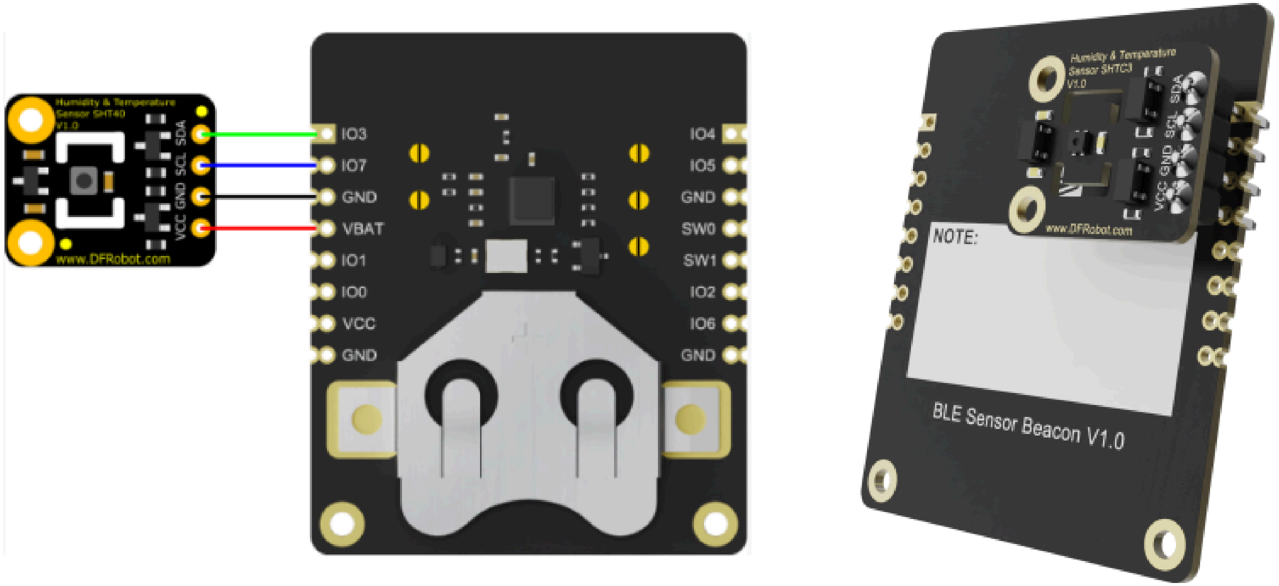
- **11.Check Configuration**

  In the bottom left corner of the software you can see that we have enabled Set #1, I2C Slave #1, GPIO3 and MGPIO7.



- **12.Connecting modules to PC ---- hardware**

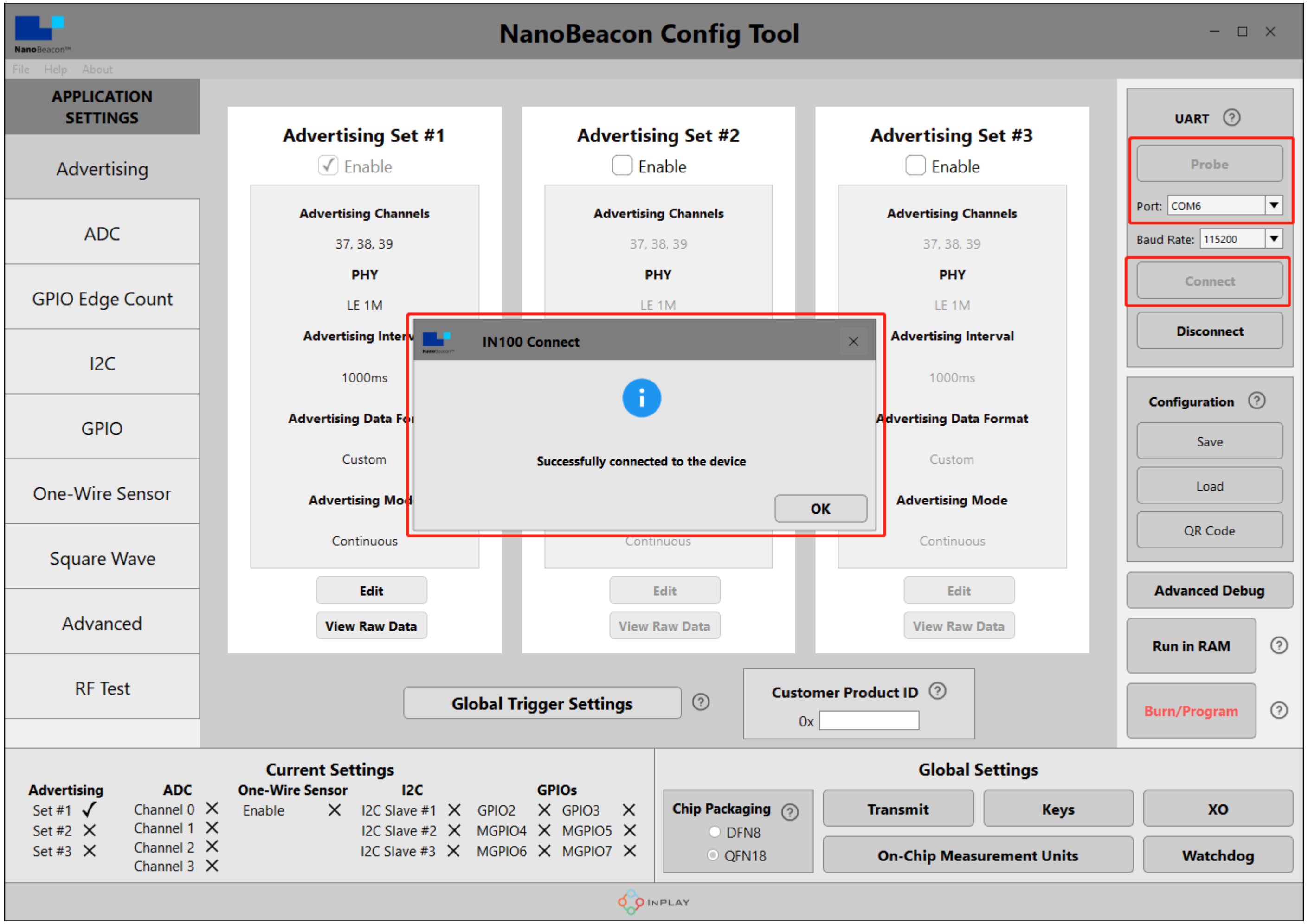
Hardware connections according to the wiring diagram



(https://img.dfrobot.com.cn/wiki/62b2fb5caa613609f271523c/3d9a0b5d6fdaa60a04fdda308ebc42a.png)

- **13.Connecting modules to PC ---- software**

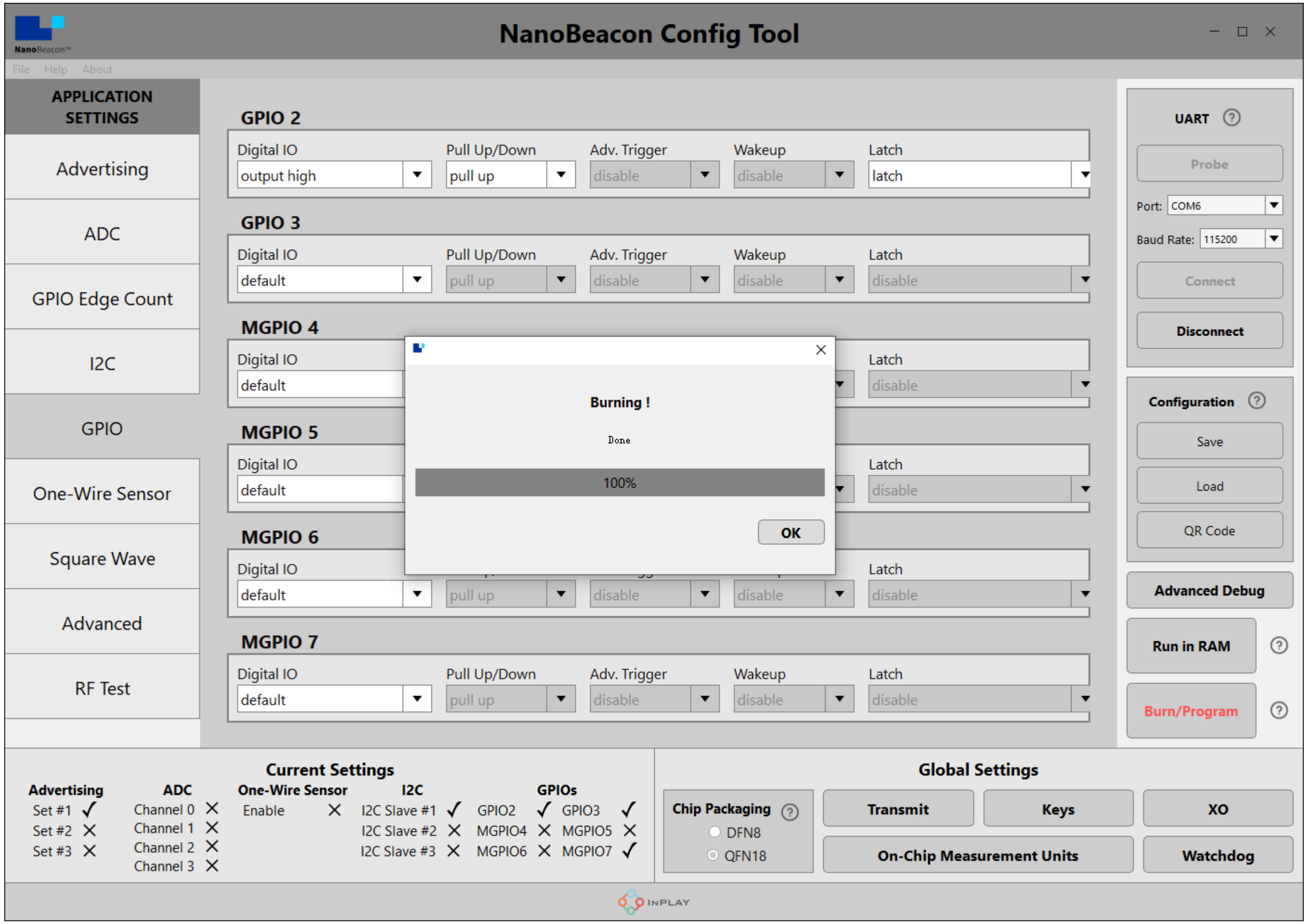
  In the upper right corner of the software, click "Probe" to refresh the port, after refreshing, select the corresponding port, click "Connect", there will be a pop-up window after successful connection.



- **14.Burn Configuration**

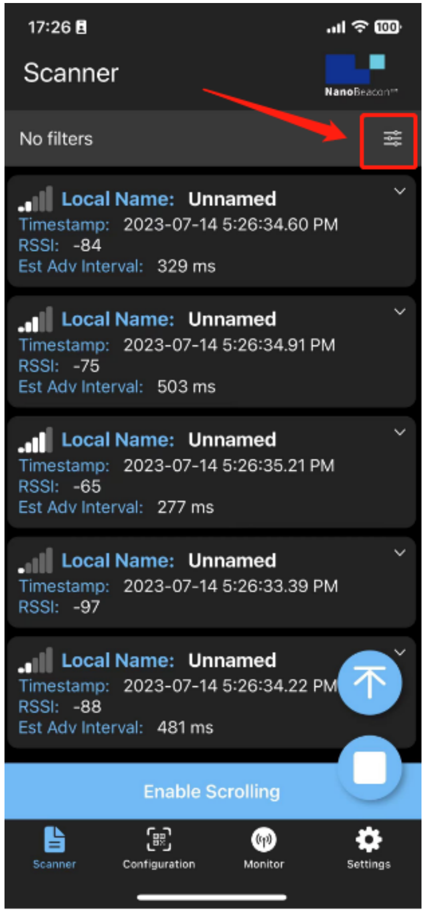
Click on "Burn/Program" and there will be a pop-up when it's done.

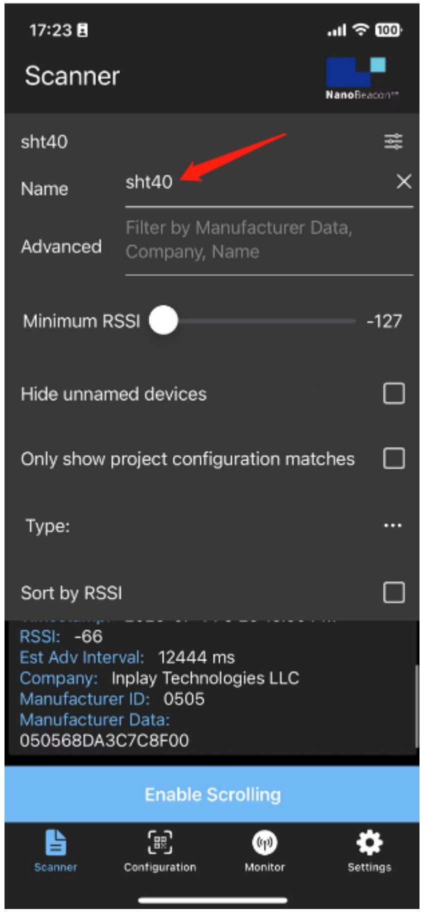
*Note: The module can only be burned once in I2C configuration, please check in detail whether the commands in the above process are correct before burning.！！！



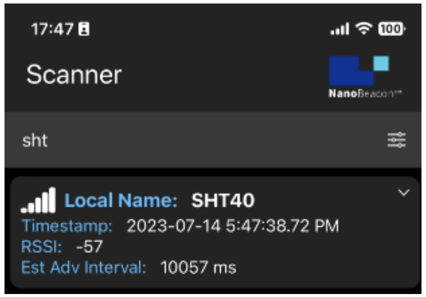
3. Mobile app to get data

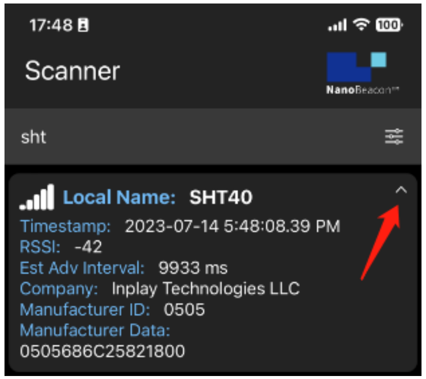
- 1.Take an IOS device for example, AppStore install and open InPlay
- 2.If there are too many other beacons in the neighbourhood to find our device, you can enter the device name of the beacon in the filter. In the tutorial for configuring sensor beacons in step 4, we named the Device Name "SHT40".





- 3.You can see that only "SHT40" remains in the menu, click on it to see the details.





- **4.Data interpretation**

"SHT40" is the Device Name set in step 4 of the sensor beacon configuration.

"686C25 821800" is the I2C acquisition data set in step 5 of Configuring Sensor Beacons

- **5.Sensor Data Calculation**

The current known sensor data captured by the beacon is "686C25 821800", which is 0x68 0x6C 0x25 0x82 0x18 0x00.

By querying the SHT40 datasheet, it is clear that when 0xFD is written to the sensor, the sensor will reply 6 bytes of data to the I2C host:

Command (hex)	Response length incl. CRC (bytes)	Description [return values]
0xFD	6	measure T & RH with high precision (high repeatability) [2 * 8-bit T-data; 8-bit CRC; 2 * 8-bit RH-data; 8-bit CRC]

Since IN100 can only receive 5 bytes of data in one tx instruction, but the SHT40 will spit out 6 bars of data when it receives 0xFD, the sixth bit of data can't be read, i.e., it is the default value of 0x00.

Take, for example, our reading of 686C25 821800, viz:

Temperature value two bytes are: 0x68,0x6C, the original value is 0x686C = 26732

Temperature value CRC checksum is: 0x25 Humidity value two bytes are: 0x82,0x18, the original value is 0x8218 = 33304 Humidity data CRC cannot be read due to hardware limitations.

Then go through the calculation formula in the SHT40 datasheet:

4.5 Conversion of Signal Output

The digital sensor signals correspond to following humidity and temperature values:

$$RH = \left(-6 + 125 \cdot \frac{S_{RH}}{2^{16}-1}\right) \%\mathrm{RH} \qquad (1)$$

$$T = \left(-45 + 175 \cdot \frac{S_T}{2^{16}-1}\right) {}^{\circ}\mathrm{C} \qquad (2)$$

$$T = \left(-49 + 315 \cdot \frac{S_T}{2^{16}-1}\right) {}^{\circ}\mathrm{F} \qquad (3)$$

Can be derived: Temperature = -45 + (175*26732/65535)* ≈ *26.38°C Humidity = -6 + (125*33304/65535) ≈ 57.52 per cent

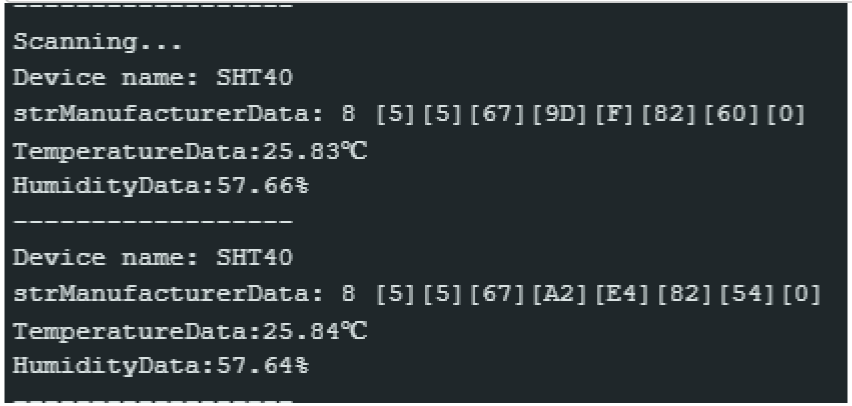
4. ESP32 acquiring data

- Prepare the Arduino IDE & Done ESP32 setup： FireBeetle_ESP32_E Set up tutorial (https://wiki.dfrobot.com/FireBeetle_Board_ESP32_E_SKU_DFR0654#target_7)
- Upload the following program for ESP32

```
/*
   Based on Neil Kolban example for IDF: https://github.com/nkolban/esp32-snippets/blob/maste
   Ported to Arduino ESP32 by Evandro Copercini
   Changed to a beacon scanner to report iBeacon, EddystoneURL and EddystoneTLM beacons by be
*/

#include <Arduino.h>
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEScan.h>
#include <BLEAdvertisedDevice.h>
#include <BLEEddystoneURL.h>
#include <BLEEddystoneTLM.h>
#include <BLEBeacon.h>
#define ENDIAN_CHANGE_U16(x) ((((x)&0xFF00) >> 8) + (((x)&0xFF) << 8))

float TemperatureData,HumidityData;
float Temperature,Humidity;

//Setting up ESP32 to scan for Bluetooth devices once every 5 seconds
int scanTime = 5; //In seconds
BLEScan *pBLEScan;

class MyAdvertisedDeviceCallbacks : public BLEAdvertisedDeviceCallbacks
{
    void onResult(BLEAdvertisedDevice advertisedDevice)
    {
      if (advertisedDevice.haveName())
      {
        if(String(advertisedDevice.getName().c_str()) == "SHT40")//Scan for a Bluetooth devic
        {
          Serial.print("Device name: ");
          Serial.println(advertisedDevice.getName().c_str());
          std::string strManufacturerData = advertisedDevice.getManufacturerData();
          uint8_t cManufacturerData[100];
```

```
          strManufacturerData.copy((char *)cManufacturerData, strManufacturerData.length(), 0
          Serial.printf("strManufacturerData: %d ", strManufacturerData.length());

          for (int i = 0; i < strManufacturerData.length(); i++)
          {
            Serial.printf("[%X]", cManufacturerData[i]);
          }

          //Getting raw data from SHT40
          TemperatureData = int(cManufacturerData[2]<<8 | cManufacturerData[3]);
          HumidityData = int(cManufacturerData[5]<<8 | cManufacturerData[6]);

          //Convert raw data into temperature and humidity data
          Temperature = (175 * TemperatureData/65535) - 45;
          Humidity = (125 * HumidityData/65535) - 6;

          Serial.println();
          Serial.print("TemperatureData:");Serial.print(Temperature);Serial.println("℃");
          Serial.print("HumidityData:");Serial.print(Humidity);Serial.println("%");
          Serial.println("------------------");
        }
      }
    }
};
void setup()
{
  Serial.begin(115200);
  Serial.println("Scanning...");

  BLEDevice::init("");
  pBLEScan = BLEDevice::getScan(); //create new scan
  pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
  pBLEScan->setActiveScan(true); //active scan uses more power, but get results faster
  pBLEScan->setInterval(100);
  pBLEScan->setWindow(99); // less or equal setInterval value
}
```

```
void loop()
{
  // put your main code here, to run repeatedly:
  BLEScanResults foundDevices = pBLEScan->start(scanTime, false);
  pBLEScan->clearResults(); // delete results fromBLEScan buffer to release memory
  delay(2000);
}
```



```
Scanning...
Device name: SHT40
strManufacturerData: 8 [5][5][67][9D][F][82][60][0]
TemperatureData:25.83℃
HumidityData:57.66%
-------------------
Device name: SHT40
strManufacturerData: 8 [5][5][67][A2][E4][82][54][0]
TemperatureData:25.84℃
HumidityData:57.64%
```

- This programme is modified from the BLE_Beacon_Scanner that comes with the ESP32, and can be modified as needed.

Dynamic Power Control

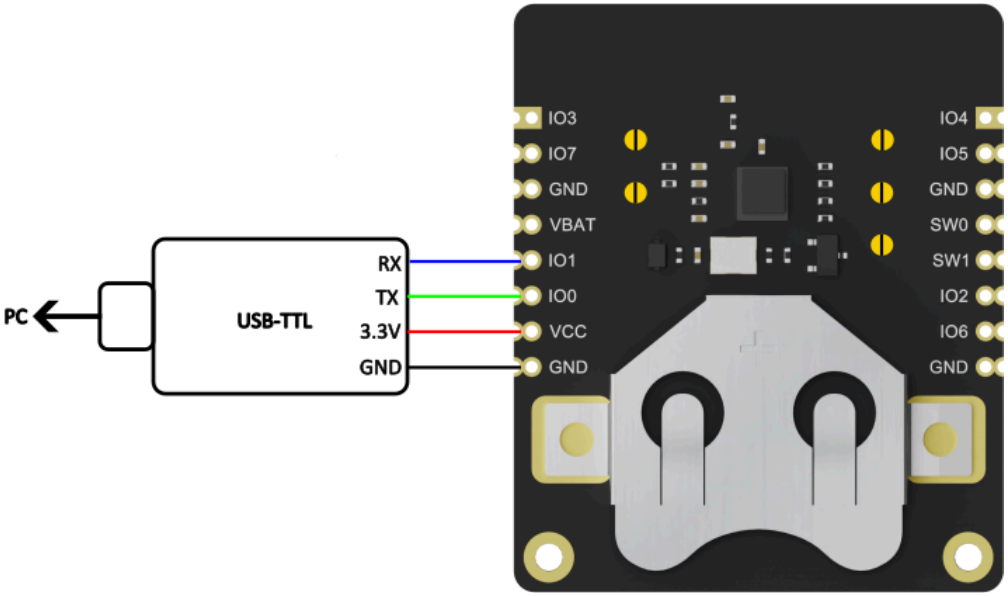
APP pop-up alerts

Fermion:Sensor beacons support APP pop-up alerts, you can set the threshold to trigger the mobile phone alerts, please see IN100 official tutorial (https://inplay-tech.com/blog/nanobeacon-ble-scanner-tutorial-part-3) for details

NanoBeacon Config Tool Instructions for Use

For more information on how to use the NanoBeacon Config Tool, see the software's user guide：
NanoBeacon Config Tool User Guide EN.pdf
(https://img.dfrobot.com.cn/wiki/5cabf4771804207b131ae8cb/dcff0894b62d849acedf1e9f70d37778.pdf)

The user guide uses the "Beacon development kit", when using the Fermion: BLE Sensor Beacon, just use the 3.3V USB-TTL tool:



FAQ

The Github repository (https://github.com/DFRobot/DFRobot_FermionBLE) holds sample code and configuration files for the sensors we have tested. You are also welcome to contact us if you have a need for a new sensor adaptation.

For other questions, please see our FAQ about Fermion: BLE sensor Beacon (https://www.dfrobot.com/forum/topic/334032) topic in Forum.

More

- IN100 datasheet (https://img.dfrobot.com.cn/wiki/5cabf4771804207b131ae8cb/408248e7f253c36c33ac92612a73cb74.pdf)
- Schematic (https://img.dfrobot.com.cn/wiki/62b2fb5caa613609f271523c/9a645e5e7e39197a8b1fe5a661f2a277.pdf)
- Dimension (https://img.dfrobot.com.cn/wiki/5cabf4771804207b131ae8cb/42f3402c04866820dba4b568ff2b5a66.pdf)
- NanoBeacon Config Tool User Guide EN (https://img.dfrobot.com.cn/wiki/5cabf4771804207b131ae8cb/dcff0894b62d849acedf1e9f70d37778.pdf)
- Sample Sensor Profiles Github Repo (https://github.com/DFRobot/DFRobot_FermionBLE)

Get Fermion: BLE Sensor Beacon (https://www.dfrobot.com.cn/goods-3799.html)