1. Developement Environment Building

1 Download Contiki Source Code (under Linux):

Firstly, clone the Contiki repository:

git clone https://github.com/contiki-os/contiki .git

Secondly, use the following command to update the firmware and every dependency about cc26xx(sensrortag):

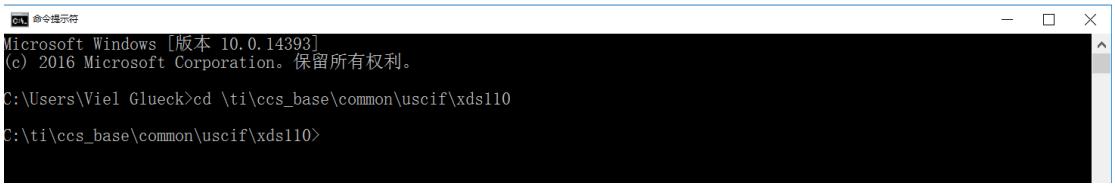
Thirdly, **Under the just downloaded Contiki repository file** and give in the following command:

git sudmodule --init –recursive

2 Update the Debugger DevPack firmware (under windows):

Firstly, download the XDS Emulation Software Package and install it.

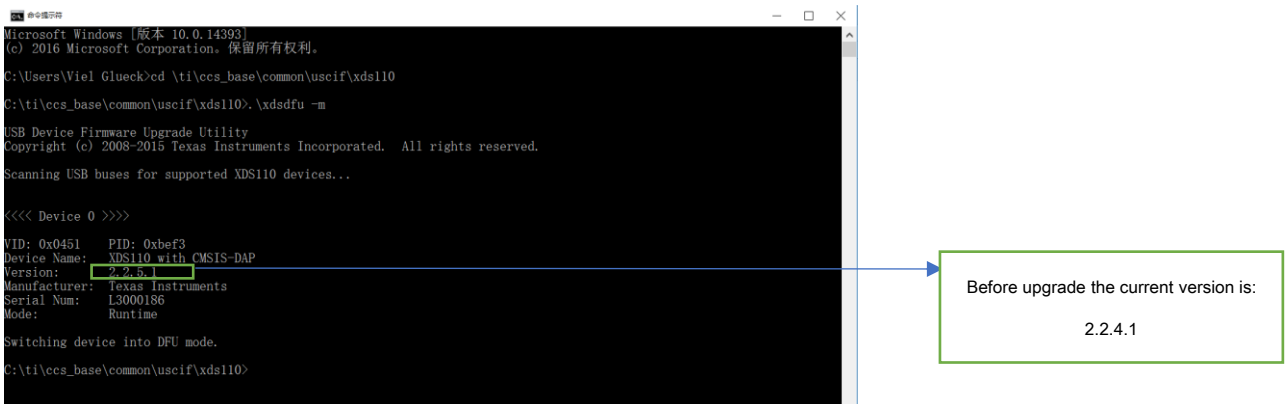
Secondly, under Windows using the bash command to get into the right path:



Thirdly, use the following command:

.\ xdsdfu -m

This command is used to see the firmware version of the Debugger.



Fourthly, using the following command to upgrade it:
.\xdsdfu -f firmware.bin -r
.\xdsdfu -m

Now the current version is 2.2.5.1, the upgrade is done!
Note: When upgrade the debugger must be disconnected from sensortag!

3 A simple RPL-UDP example:

In this example two cc2650 sensortag will build a UDP connection using RPL protocol. The client will send ‘Hello’ to the server. We can see the result through the UART on the Debugger to see the Result.

Firstly, we compile the source code under Linux. Go to the right path of the example:

cd /Desktop/contiki/example/ipv6/rpl-udp

Then we build the server code and the client code separately:
Take client for example:

sudo make TARGET=srf06-cc26xx BOARD=sensrotag/cc2650 udp-client.bin

Before we compile the server code, we must clean the middle files of the previous project:

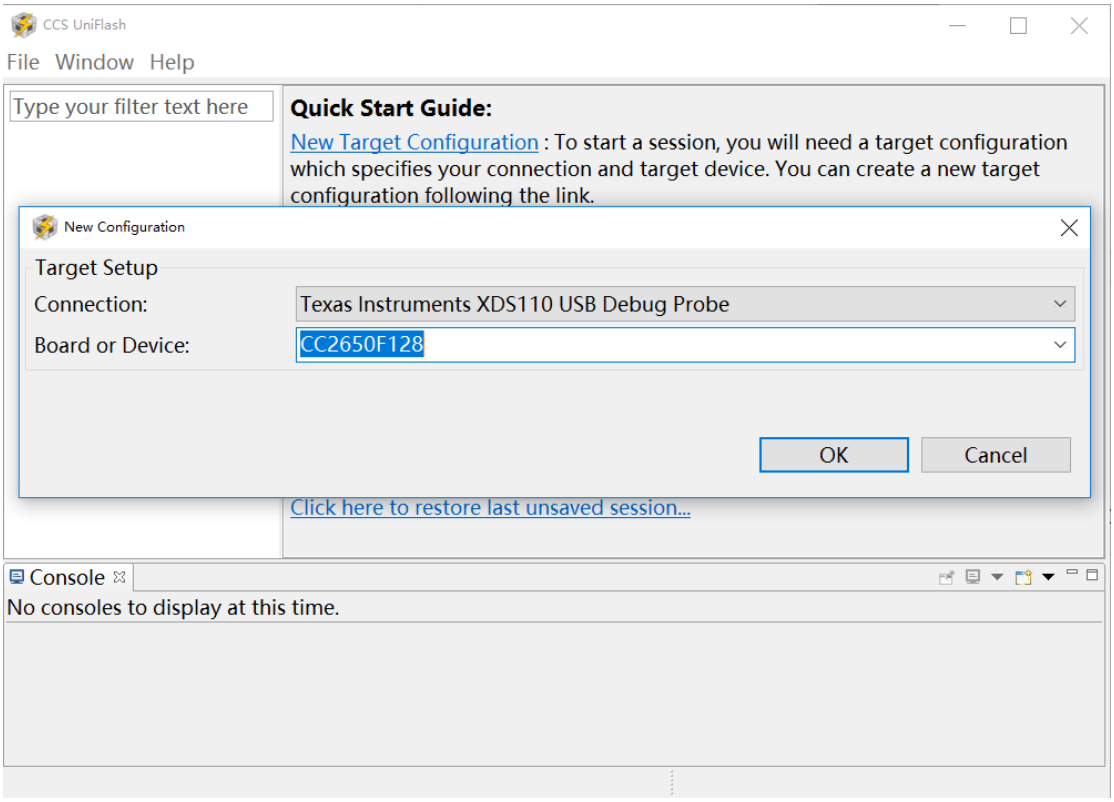
sudo make TARGET=srf06-cc26xx BOARD=sensrotag/cc2650 udp-client.bin clean

Then we can compile the server code:

sudo make TARGET=srf06-cc26xx BOARD=sensrotag/cc2650 udp-server.bin

We can find the output files are:
udp-client.bin
udp-client.elf
udp-client.bin
udp-client.elf

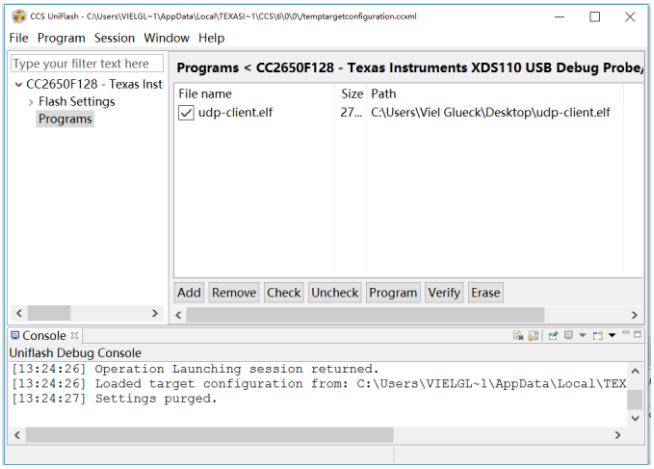
Secondly, we need to flash it into our sensortag. It is recommended to flash it under Windows OS. We need to download the Uniflash and install it. When open the Uniflash software, we create a new file and set as following:



Select on **Flash Setting** menu and click on **Erase Entire Flash.** It'll tell you that you are not running the latest version and it needs to be upgrade. Just click the upgrade and wait.

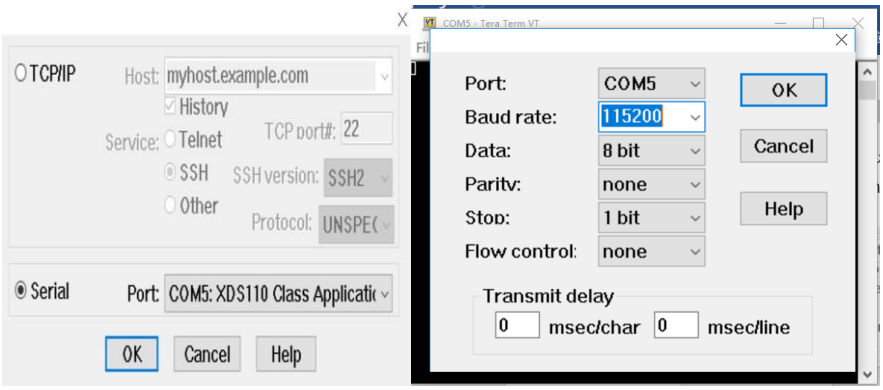
The above console output tells you it is been upgraded.

Thirdly, flash it with your code: click **programs table** and **add .elf** file, click **program.**

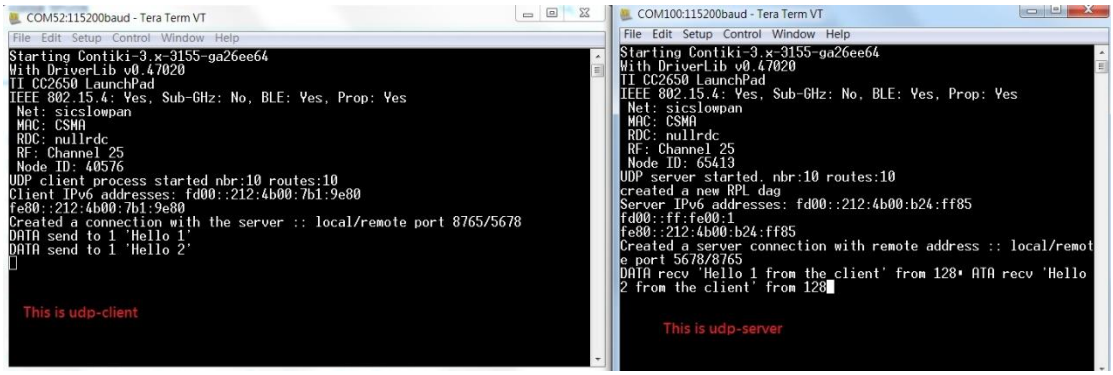


4 Debugger results:

Firstly, it is recommended to use an open source software called **Tera Term** to monitor the **com port** on your computer. **Note: The Baud rate of the UART is 115200.**



Secondly. We can see the client is sending hello to server:



2. Hardware AES:

According to the latest updated cc26xxware, there is a ROM AES API:

AES_ECB_EncryptData(uint8_t *text, uint16_t textLen, uint8_t *aesKey);

which can be called by the user application. The discussion on the TI forum shows that, there is a initial process before the above API is called, this initial process is allocating a absolute RAM scratch area for the ROM AES stack remapping. This remapping is done at :

https://github.com/contiki-os/cc26xxware/tree/master/utils/rom_crypto

According the code document, it should be called before the API usage. So my code is like:

```
StandaloneRomCryptoInit();
AES_ECB_EncryptData(uint8_t *text, uint16_t textLen, uint8_t *aesKey);
```

and also there is a strange symbol in the standalone_rom_crypto.c :

commonROMScratchArea[53] @0x20004F2C;

i guess this is for specific compiler, so i change this to:

uint32_t __attribute__ ((section(".COMMENROMSCRATCHAREA"))) commonROMScratchArea[53] ;

here the .COMMENROMSCRATCHAREA refers to the absolute address: 0x20004F2C

the according linkerfile has been added .COMMENROMSCRATCHAREA like followings:

```
.COMMENROMSCRATCHAREA 0x20000754:
        {

      KEEP(*(.COMMENROMSCRATCHAREA))

        } > SRAM
```

after all this after compiling contiki ,the c2650 is stuck at the API:

AES_ECB_EncryptData(uint8_t *text, uint16_t textLen, uint8_t *aesKey);

The test application is rpl-udp in directory Contiki/example/ipv6/rpl-udp. One senortag uses the ROM aes to send encrypted data to another sensortag.

However this AES part seems not working.