AVT9152 EVB

Software Guide for Building Nordic Sample Projects

v1.0 – December 04, 2020

Document Control

Document Version:	1.0
Document Date:	04 Dec 2020
Document Author:	Jennifer Sy, Alan Low
Document Classification:	Public
Document Distribution:	Public

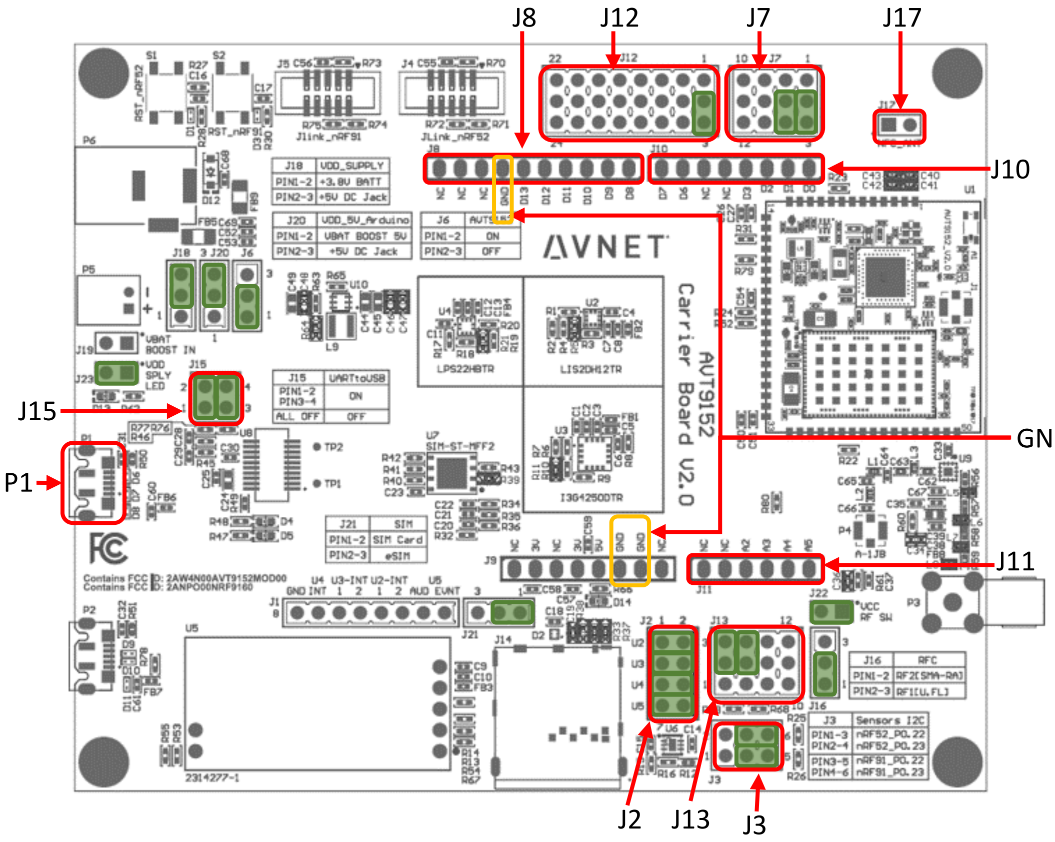
Version History

Version	Date	Comment
1.0	04 Dec 2020	First Release

Contents

1 Introduction

This document provides information on how to adapt and build existing samples from nRF Connect SDK (NCS) v1.4.0 for AVT9152 EVB.



nRF9160 IO	Arduino/Function pinout	Connecting Jumper Header Pin
P0.13	J11-A2	J13 [2-3]
	J11-A4	J13 [8-9]
P0.14	J11-A3	J13 [5-6]
	J11-A5	J13 [11-12]
P0.15	J10-D0	J7 [2-3]
	J10-D2	J7 [8-9]
P0.16	J10-D1	J7 [5-6]
	J10-D3	J7 [11-12]
P0.20	J10-D6	J12 [2-3]
	J8-D10	J12 [14-15]
P0.21	J10-D7	J12 [5-6]
	J8-D11	J12 [17-18]
P0.22	SENS_SCL *	J3 [3-5]
P0.23	SENS_SDA *	J3 [4-6]

P0.24	J8-D8	J12 [8-9]
	J8-D12	J12 [20-21]
	UART_RX **	J15 [1-2]
P0.25	J8-D9	J12 [11-12]
	J8-D13	J12 [23-24]
	UART_TX **	J15 [3-4]

nRF52840 IO	Arduino/Function pinout	Connecting Jumper Header Pin
P0.02	J11-A2	J13 [1-2]
	J11-A4	J13 [7-8]
P0.03	J11-A3	J13 [4-5]
	J11-A5	J13 [10-11]
P0.04	J10-D0	J7 [1-2]
	J10-D2	J7 [7-8]
P0.05	J10-D1	J7 [4-5]
	J10-D3	J7 [10-11]
P0.09	J17-1 (NFC1)	-
P0.10	J17-2 (NFC2)	-
P0.19	J10-D6	J12 [1-2]
	J8-D10	J12 [13-14]
P0.21	J10-D7	J12 [4-5]
	J8-D11	J12 [16-17]
P0.22	SENS_SCL *	J3 [1-3]
P0.23	SENS_SDA *	J3 [2-4]
P0.24	J8-D8	J12 [7-8]
	J8-D12	J12 [19-20]
P1.0	J8-D9	J12 [10-11]
	J8-D13	J12 [22-23]

List of IOs accessible from AVT9152 EVB

* SENS_SCL and SENS_SDA are the I2C pins connected to the on-board sensors.

** UART_RX and UART_TX are connected to the USB-to-UART converter accessible through P1.

2 Pre-requisites

User is expected to have a copy of NCS v1.4.0 and the necessary development environment setup. (Please refer to http://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/gs_assistant.html if you are new to NCS.)

3 Board configuration files

AVT9152 EVB board configuration files can be downloaded from AVT9152_DEMO repo.

They are grouped into 2 folders namely **nrf9160_avt9152** and **nrf52840_avt9152** in *<your work folder>\AVT9152_DEMO\boards\arm\* folder for each chip integrated in the AVT9152 module.

3.1 Alias defined in Device Tree Source (DTS) files

Below is the list of aliases defined in

*<your work folder>\AVT9152_DEMO\boards\arm\nrf9160_avt9152\***nrf9160_avt9152_common.dts** and *<your work folder>\AVT9152_DEMO\boards\arm\nrf52840_avt9152\***nrf52840_avt9152.dts**.

Alias	Define for (chip)	Description
ard_a2	Both	Arduino A2 pin
ard_a3	Both	Arduino A3 pin
ard_a4	Both	Arduino A4 pin
ard_a5	Both	Arduino A5 pin
ard_d0	Both	Arduino D0 pin
ard_d1	Both	Arduino D1 pin
ard_d2	Both	Arduino D2 pin
ard_d3	Both	Arduino D3 pin
ard_d6	Both	Arduino D6 pin
ard_d7	Both	Arduino D7 pin
ard_d8	Both	Arduino D8 pin
ard_d9	Both	Arduino D9 pin
ard_d10	Both	Arduino D10 pin
ard_d11	Both	Arduino D11 pin
ard_d12	Both	Arduino D12 pin
ard_d13	Both	Arduino D13 pin
interconn_c0	Both	Inter-connection C0 pin
interconn_c1	Both	Inter-connection C1 pin
interconn_c2	Both	Inter-connection C2 pin
interconn_c3	Both	Inter-connection C3 pin
interconn_c4	Both	Inter-connection C4 pin
interconn_c5	Both	Inter-connection C5 pin
sens_i2c	Both	I2C master peripheral to communicate with all built-in sensors.
nrf52_reset	nRF9160	nRF52840 reset control pin
debug_uart	nRF9160	UART peripheral for debug print via USB port (P1)
nrf91_coex0	nRF52840	nRF52 pin connected to nRF91 COEX0 pin
nrf91_coex1	nRF52840	nRF52 pin connected to nRF91 COEX1 pin
nrf91_coex2	nRF52840	nRF52 pin connected to nRF91 COEX2 pin
nrf91_reset	nRF52840	nRF9160 reset control pin
pwr_good	nRF52840	PG status from DC-DC

3.2 Peripherals enabled in DTS files

UART_CONSOLE is by default enabled for both nRF9160 and nRF52840 as defined in their respective defconfig files.

It is set to use the uart0 at 115200 baud rate with the following pins:

Chip	Pins
nRF9160	TX=P0.25

	RX=P0.24
nRF52840	TX=P0.24
	RX=P1.0

For nRF52840, P0.18 is by default configured as RESET pin.

3.3 Sample code on using alias

```
#include <zephyr.h>
#include <device.h>
#include <drivers/i2c.h>
#include <drivers/gpio.h>

#define ARD_D0_NODE                 DT_ALIAS(ard_d0)    //Arduino D0 pin.
#define SENS_I2C_NODE               DT_ALIAS(sens_i2c) //I2C master for built-in sensors.

#define ARDUINO_D0_PORT_DEVICE      DT_GPIO_LABEL(ARD_D0_NODE, gpios)
#define ARDUINO_D0_PIN_NUMBER       DT_GPIO_PIN(ARD_D0_NODE, gpios)
#define SENSORS_I2C_DEVICE          DT_LABEL(SENS_I2C_NODE)
#define SENSOR_SLAVE_ADDR           0x20 //7-bit slave address
#define SENSOR_REG1                 0x01 //REG1 address

struct device *sensors_i2c_dev = NULL;
struct device *arduino_d0_port_dev = NULL;

void main(void) {
        //get sensors i2c device
        sensors_i2c_dev = device_get_binding(SENSORS_I2C_DEVICE);
        if (sensors_i2c_dev != NULL) {
                struct i2c_msg msgs[1];
                uint8_t payload[2];

                payload[0] = SENSOR_REG1;
                payload[1] = 0xff;
                msgs[0].buf = payload;
                msgs[0].len = 2U;
                msgs[0].flags = I2C_MSG_WRITE | I2C_MSG_STOP;

                //Send I2C message to sensor device.
                i2c_transfer(sensors_i2c_dev, msgs, 1, SENSOR_SLAVE_ADDR);
        }
        else {
                printk("Unable to get sensors I2C device!\r\n");
        }

        //get arduino D0 pin resource and configure as input pin.
        arduino_d0_port_dev = device_get_binding(ARDUINO_D0_PORT_DEVICE);
        if (arduino_d0_port_dev != NULL) {
                //Configure Arduino D0 pin
                gpio_pin_configure(arduino_d0_port_dev, ARDUINO_D0_PIN_NUMBER, GPIO_INPUT);
        }
        else {
                printk("Unable to get Arduino D0 port!\r\n");
        }
}
```

4 Preparing example from NCS for AVT9152 EVB

4.1 Check for hardware dependencies

Please refer to the documentation specific to the selected example like its README.rst and/or from http://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/examples.html to check for its hardware requirements.

If the example requires only the UART_CONSOLE, you can proceed to Building example from NCS for AVT9152 EVB as this is enabled by default when using the provided AVT9152 EVB board files. **at_client** in *<NCS installation folder>\nrf\samples\nrf9160\* folder is one such example.

Otherwise, we need to see if AVT9152 EVB can be used and make the necessary changes.

4.2 Adapt the example for AVT9152 EVB

With the example of interest requiring more peripherals, we will need to use device tree overlay and/or project configuration file to configure them.

Let us go through on how to modify the **asset_tracker** in *<NCS installation folder>\nrf\applications\* folder to run on AVT9152 EVB instead of nRF9160 DK (PCA10090).

1) Define the buttons and LEDs
    a) Create a new directory **boards** in your *<NCS installation folder>\nrf\applications\asset_tracker\* folder.
    b) Create a **nrf9160_avt9152ns.overlay** file in the newly created **boards** folder with the following contents.
    These are lifted *from <NCS installation folder>\zephyr\boards\arm\nrf9160dk_nrf9160\**nrf9160dk_nrf9160_common.dts*** with gpio(s) changed to those available in AVT9152 EVB.

```
/ {
        leds {
                compatible = "gpio-leds";
                led0: led_0 {
                        gpios = <&gpio0 3 0>;
                        label = "Green LED 1";
                };
                led1: led_1 {
                        gpios = <&gpio0 4 0>;
                        label = "Green LED 2";
                };
                led2: led_2 {
                        gpios = <&gpio0 13 0>;
                        label = "Green LED 3";
                };
                led3: led_3 {
                        gpios = <&gpio0 14 0>;
                        label = "Green LED 4";
                };
        };
        buttons {
                compatible = "gpio-keys";
                button0: button_0 {
                        gpios = <&gpio0 15 (GPIO_PULL_UP | GPIO_ACTIVE_LOW)>;
                        label = "Switch 1";
                };
                button1: button_1 {
                        gpios = <&gpio0 16 (GPIO_PULL_UP | GPIO_ACTIVE_LOW)>;
                        label = "Switch 2";
                };
                button2: button_2 {
                        gpios = <&gpio0 20 (GPIO_PULL_UP | GPIO_ACTIVE_LOW)>;
                        label = "Push button 1";
                };
                button3: button_3 {
                        gpios = <&gpio0 21 (GPIO_PULL_UP | GPIO_ACTIVE_LOW)>;
                        label = "Push button 2";
                };
        };

        /* These aliases are provided for compatibility with samples */
        aliases {
                led0 = &led0;
                led1 = &led1;
                led2 = &led2;
                led3 = &led3;
                sw0 = &button2;
                sw1 = &button3;
                sw2 = &button0;
                sw3 = &button1;
        };
};
```

This will result to the following buttons and LEDs IO mapping.

PCA10090	nRF9160 IO	AVT9152 EVB
Button 1	P0.20	Arduino D6
Button 2	P0.21	Arduino D7
Switch 1	P0.15	Arduino D0
Switch 2	P0.16	Arduino D1
LED 3	P0.13	Arduino A2
LED 4	P0.14	Arduino A3
LED 1	P0.03	-
LED 2	P0.04	-

2) Define required board specific configuration
   a) Create a copy of *<NCS installation folder>\nrf\applications\asset_tracker\***prj.conf*** in the newly created **boards** folder and rename it as **nrf9160_avt9152ns.conf**.
   b) Modify **nrf9160_avt9152ns.conf** to have the following configurations

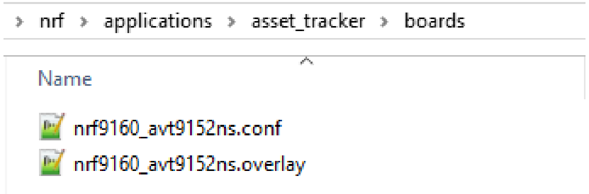
```
CONFIG_FLIP_INPUT=3
CONFIG_TEMP_SIM_BUTTON=2
CONFIG_NRF_CLOUD_CLIENT_ID_PREFIX="avt-"
```

CONFIG_FLIP_INPUT will configure the application to take Arduino D0 as FLIP input.

CONFIG_TEMP_SIM_BUTTON, although is not used, needs to be defined as there is no default value set in asset_tracker's KConfig for custom board.

CONFIG_NRF_CLOUD_CLIENT_ID_PREFIX is to change the MQTT client id prefix as the default "nrf-" is reserved only for official Nordic devices on nRF Cloud.

You should now have 2 files in your *<NCS installation folder>\nrf\applications\asset_tracker\boards* folder as shown below.



nrf9160_avt9152ns.conf
nrf9160_avt9152ns.overlay

3) Setup AVT9152 EVB
Populate the following headers with jumpers.

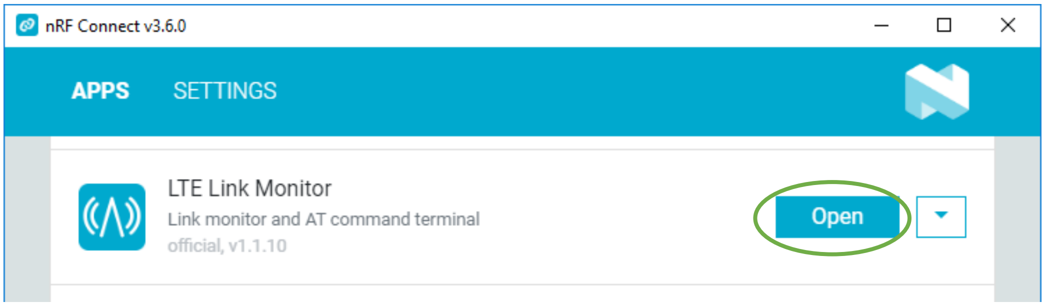
Header [Pin]	Header-Arduino pinout
J12 [2-3]	J10-D6
J12 [5-6]	J10-D7
J7 [2-3]	J10-D0
J7 [5-6]	J10-D1
J13 [2-3]	J11-A2
J13 [5-6]	J11-A3

Insert a jump wire into each corresponding Arduino pinout header with the other end of the wire floating.

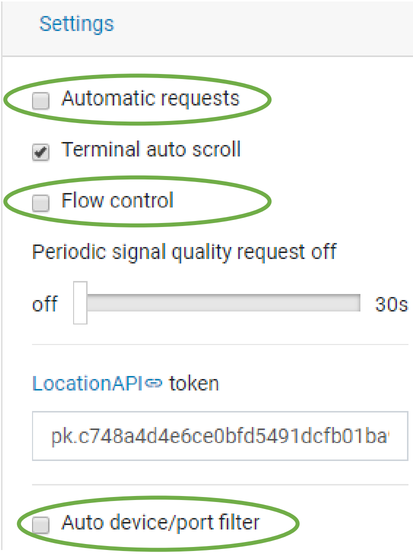
Buttons and LEDs are all active low. You can short the specific Arduino pinout mapped to function as button to the GND to simulate a button press and un-short it to release. You can use a digital analyzer to monitor the state of the pins to function as LED.

4) Install nRF Cloud certificate
Application running on nRF9160 of AVT9152 needs to be able to accept long AT commands. For example, the **at_client** and the pre-flashed AVT9152 demo software when in command mode.

   a) Open nRF Connect for Desktop and launch LTE Link Monitor.



b) In the Settings pane on the lower right, untick **Automatic requests**, **Flow control**, and **Auto device/port filter**.



Settings

□ Automatic requests

☑ Terminal auto scroll

□ Flow control

Periodic signal quality request off

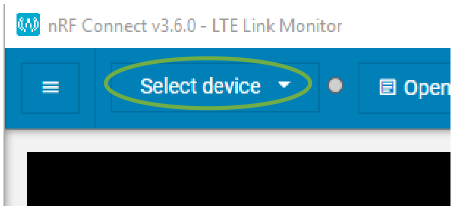
off 30s

LocationAPI token

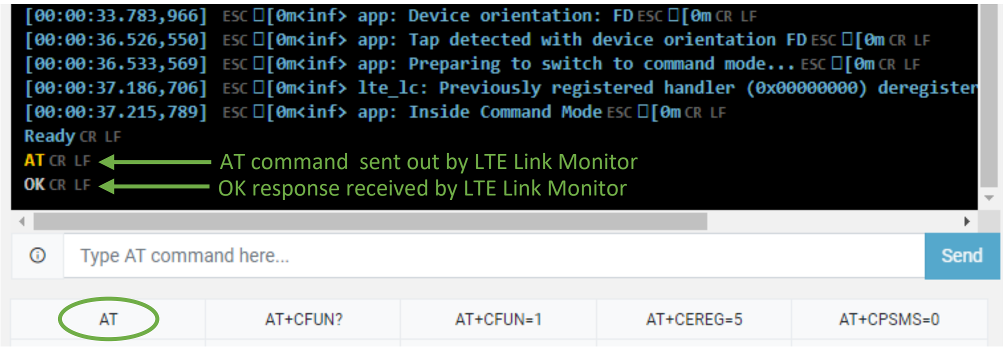
pk.c748a4d4e6ce0bfd5491dcfb01ba

□ Auto device/port filter

c) Select the Virtual COM port associated to AVT9152 EVB P1.



d) Check if target is responsive to commands.



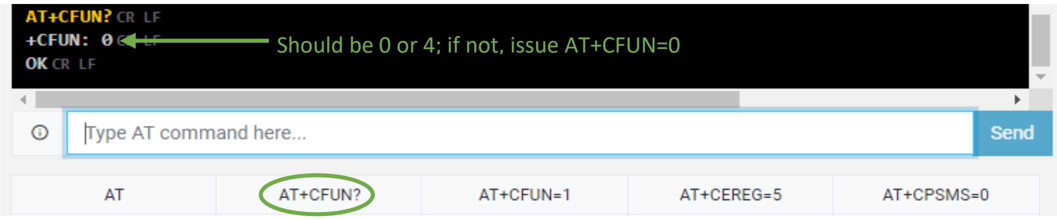
```
[00:00:33.783,966] ESC □[0m<inf> app: Device orientation: FD ESC □[0m CR LF
[00:00:36.526,550] ESC □[0m<inf> app: Tap detected with device orientation FD ESC □[0m CR LF
[00:00:36.533,569] ESC □[0m<inf> app: Preparing to switch to command mode... ESC □[0m CR LF
[00:00:37.186,706] ESC □[0m<inf> lte_lc: Previously registered handler (0x00000000) deregister
[00:00:37.215,789] ESC □[0m<inf> app: Inside Command Mode ESC □[0m CR LF
Ready CR LF
AT CR LF
OK CR LF
```

AT command sent out by LTE Link Monitor

OK response received by LTE Link Monitor

If not, reopen the COM port by repeating step c.

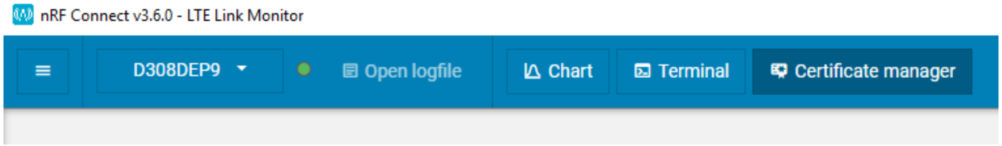
e) Make sure the modem is in offline state.



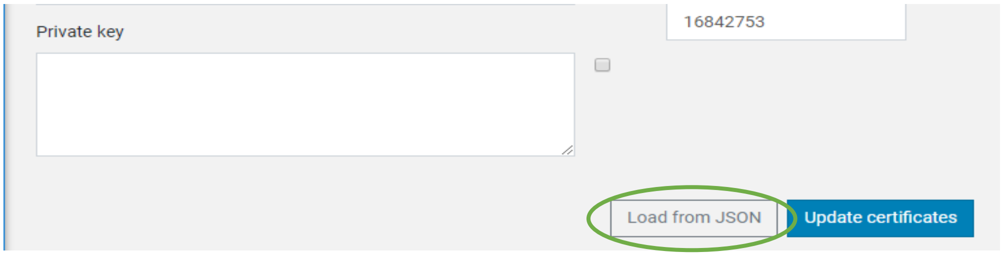
```
AT+CFUN? CR LF
+CFUN: 0 CR LF
OK CR LF
```

Should be 0 or 4; if not, issue AT+CFUN=0

f) Click **Certificate manager** in the navigation bar to switch to the certificate manager view.



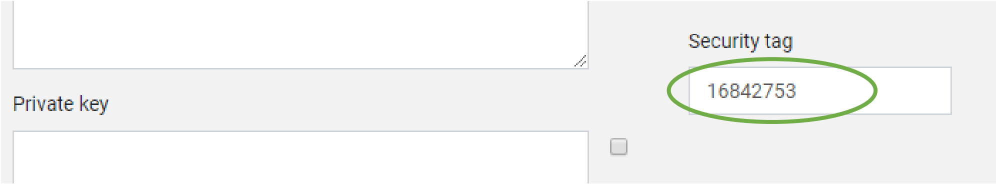
g) Load the downloaded JSON file from nRF Cloud for your device (avt-<imei>).



Private key

16842753

h) Make sure the Security tag is set to **16842753**.

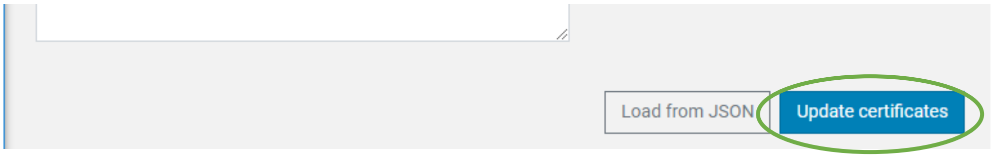


Security tag

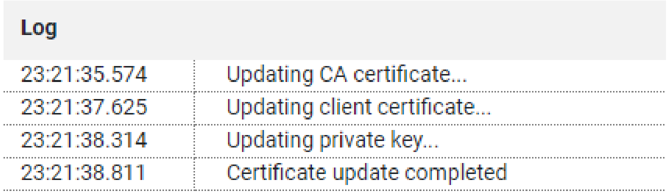
16842753

Private key

i) Click **Update certificates**.



You shall see similar logs in the bottom Log pane.

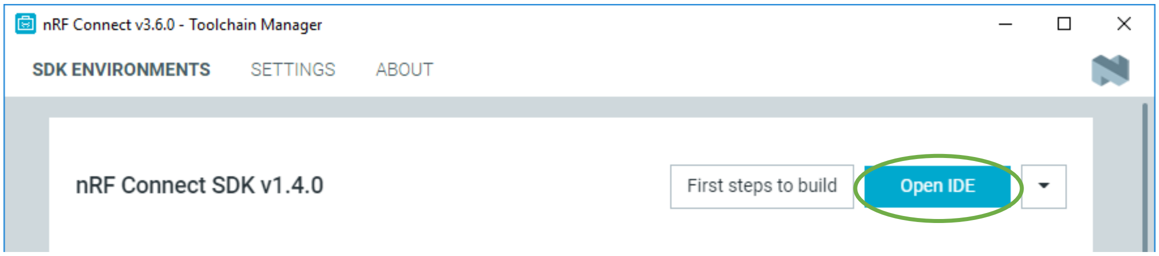


```
Log
23:21:35.574   Updating CA certificate...
23:21:37.625   Updating client certificate...
23:21:38.314   Updating private key...
23:21:38.811   Certificate update completed
```

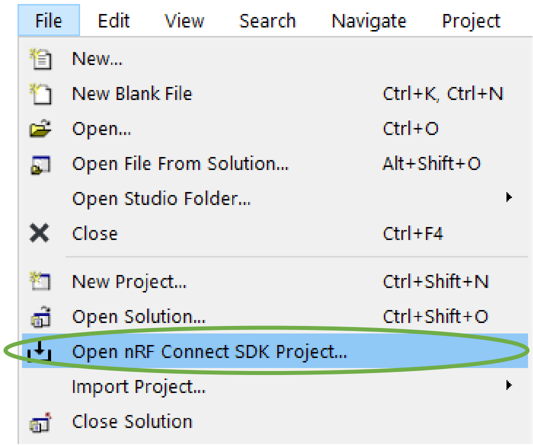
5 Building example from NCS for AVT9152 EVB

5.1 For nRF9160

1) Open SEGGER Embedded Studio for ARM (SES).



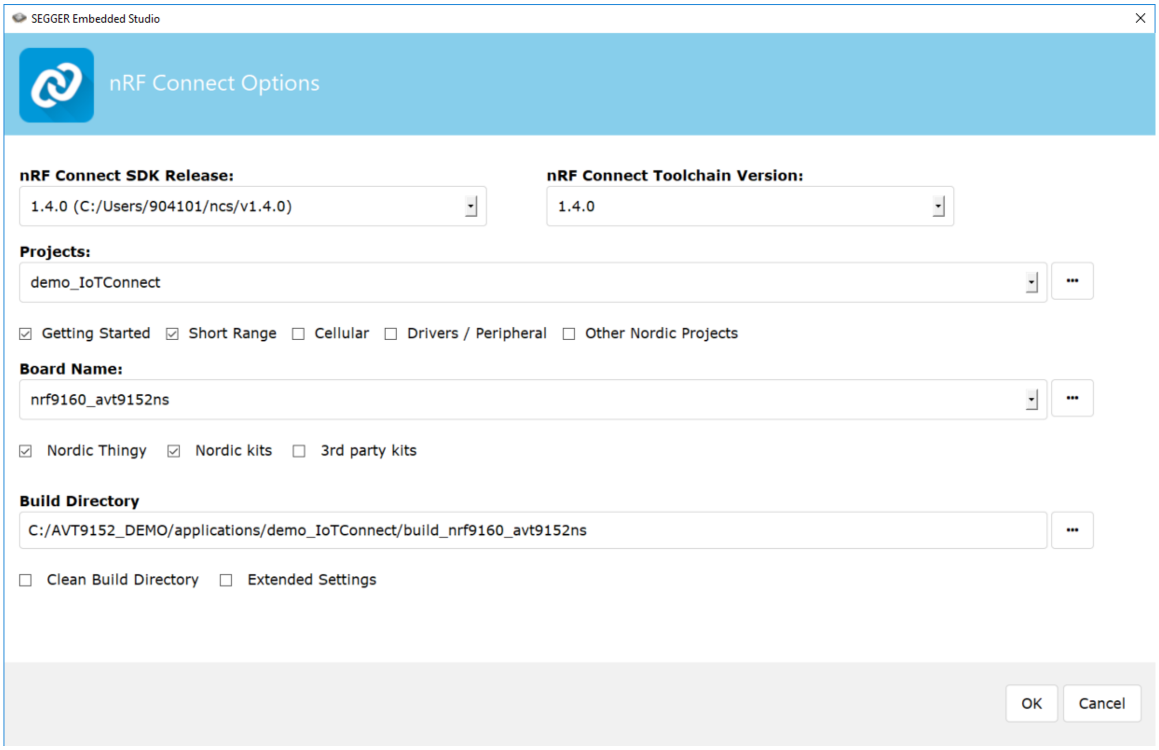
2) Select **File** -> **Open nRF Connect SDK Project…**


File
Edit
View
Search
Navigate
Project
New...
New Blank File Ctrl+K, Ctrl+N
Open... Ctrl+O
Open File From Solution... Alt+Shift+O
Open Studio Folder...
Close Ctrl+F4
New Project... Ctrl+Shift+N
Open Solution... Ctrl+Shift+O
Open nRF Connect SDK Project...
Import Project...
Close Solution

3) Fill in the **nRF Connect Options** dialog box accordingly.

   a. Click the **…** button on the right of the dropdown-list for **Projects** and select the folder where the CmakeList.txts of the sample project of interest is located.

b. Click the **…** button on the right of the dropdown-list for **Board Name** and select *<your working directory>\AVT9152_DEMO\boards\arm\nrf9160_avt9152*.



nRF Connect SDK Release: 1.4.0 (C:/Users/904101/ncs/v1.4.0)

nRF Connect Toolchain Version: 1.4.0

Projects: demo_IoTConnect

☑ Getting Started ☑ Short Range □ Cellular □ Drivers / Peripheral □ Other Nordic Projects

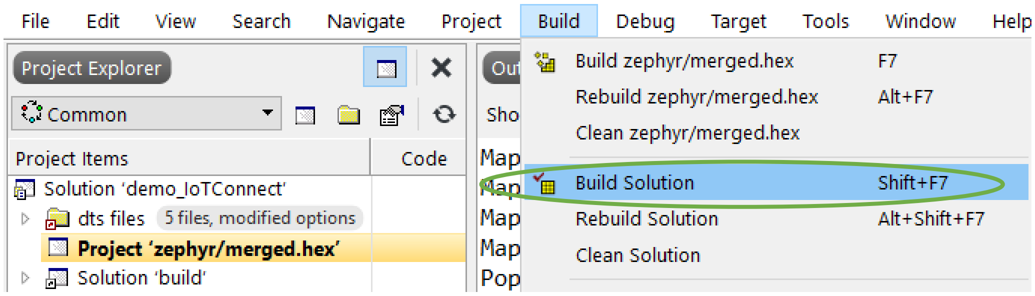
Board Name: nrf9160_avt9152ns

☑ Nordic Thingy ☑ Nordic kits □ 3rd party kits

Build Directory C:/AVT9152_DEMO/applications/demo_IoTConnect/build_nrf9160_avt9152ns

□ Clean Build Directory □ Extended Settings

4) Click **OK** and wait for SES to load the project.
5) Select **Build** -> **Build Solution** or press **Shift+F7** to build the application.



6) Once the build completes, you can find the **merged.hex** in the **zephyr** subdirectory under your build directory.

5.2 For nRF52840

The procedure to build examples for nRF52840 is the same as nRF9160 except for having the

- **Board Directory** set to *<your work folder>\AVT9152_DEMO\boards\arm\**nrf52840_avt9152***
- **Board Name** set to **nrf52840_avt9152**

6 Programming AVT9152

Please refer to AVT9152 EVB Programming Guide.

Note: Serial recovery through MCUboot capability will no longer be available once the target is programmed through SWD with firmware not having this feature. Please go through Enabling MCUboot serial recovery if you want to retain this feature.

7 Enabling MCUboot serial recovery

7.1 For nRF9160

1) Create a **nrf9160_avt9152.conf** file in your <*NCS installation folder*>\ *bootloader**mcuboot**boot**zephyr**boards*\ folder with the following contents.

```
CONFIG_UART_CONSOLE=n

CONFIG_SIZE_OPTIMIZATIONS=y

# MCUBoot settings
CONFIG_BOOT_MAX_IMG_SECTORS=256

# MCUboot serial recovery
CONFIG_MCUBOOT_SERIAL=y
CONFIG_BOOT_SERIAL_DETECT_PORT="GPIO_0"
CONFIG_BOOT_SERIAL_DETECT_PIN=21
CONFIG_BOOT_SERIAL_DETECT_PIN_VAL=0

CONFIG_BOOT_SIGNATURE_TYPE_RSA=y
CONFIG_BOOT_SIGNATURE_KEY_FILE="root-rsa-2048.pem"
```

2) Make sure to have the **CONFIG_BOOTLOADER_MCUBOOT=y** in your project configuration file to include the MCUboot as child image as well as for the build process to also generate the MCUboot compatible binary file **app_update.bin**.
3) Reload and rebuild the solution.

7.2 For nRF52840

1) Create a **nrf52840_avt9152.conf** file in your <*NCS installation folder*>\ *bootloader**mcuboot**boot**zephyr**boards*\ folder with the following contents.

```
CONFIG_UART_CONSOLE=n

# The build won't fit on the partition allocated for it without size
# optimizations.
CONFIG_SIZE_OPTIMIZATIONS=y
CONFIG_PM_PARTITION_SIZE_MCUBOOT=0x12000

# Serial
CONFIG_SERIAL=y
CONFIG_UART_NRFX=y
CONFIG_UART_INTERRUPT_DRIVEN=y
```

```
CONFIG_UART_LINE_CTRL=y

# MCUboot serial recovery
CONFIG_GPIO=y
CONFIG_MCUBOOT_SERIAL=y
CONFIG_BOOT_SERIAL_CDC_ACM=y
CONFIG_BOOT_SERIAL_DETECT_PORT="GPIO_0"
CONFIG_BOOT_SERIAL_DETECT_PIN=21
CONFIG_BOOT_SERIAL_DETECT_PIN_VAL=0

# Required by USB
CONFIG_MULTITHREADING=y

# USB
CONFIG_USB=y
CONFIG_USB_DEVICE_STACK=y
CONFIG_USB_DEVICE_PRODUCT="MCUBOOT"
CONFIG_USB_CDC_ACM=y
CONFIG_USB_COMPOSITE_DEVICE=y
CONFIG_USB_MASS_STORAGE=n
CONFIG_USB_DEVICE_MANUFACTURER="Nordic Semiconductor"
CONFIG_USB_DEVICE_VID=0x1915
CONFIG_USB_DEVICE_PID=0x520F

CONFIG_BOOT_SIGNATURE_TYPE_RSA=y
CONFIG_BOOT_SIGNATURE_KEY_FILE="root-rsa-2048.pem"
```

2) Make sure to have the **CONFIG_BOOTLOADER_MCUBOOT=y** in your project configuration file to include the MCUboot as child image as well as for the build process to also generate the MCUboot compatible binary file **app_update.bin**.
3) Reload and rebuild the solution.