

Configuring and using the DTM example from the Nordic SDK

Jay Tyzzer – Sr. FAE – Nordic Semiconductor – 2017

You have spent a lot of time working on your firmware, tested it out on the Nordic Development board and have laid out your PCBA. It has been reviewed by Nordic Technical Support and has now gone to prototype stage. Your boards have come back from assembly and it is time to test them. How do you go about testing them? This is a question we get a few times a week. There are a few ways to test them to be sure they are functioning. This brief will go over how this can be accomplished using the Nordic tools without spending thousands of dollars on specialized equipment. We will be using the nRF5x Development kits on these projects. Note that there is an emphasis in this paper on how to reconfigure the GPIO / Physical pins on the SoC to fit your needs as the preconfigured serial ports may not be available on your design. One way to do a functional test on the nRF5x device is to program it with one of the standard examples that come with the Nordic SDK for the nRF5x device. An example to use is the Heart Rate Monitor example (HRM). Programming the target board with the compiled hex and Softdevice is easy using the Debug out on the nRF5x-DK. Then using a Smart App like nRFconnect you can see if your product is advertising. This is a fast way of seeing if the product is working. However it does not tell you how well you are transmitting, or the harmonic content etc. To do that you may want to do more robust testing.

Most of the testing done for agency approval is done at the Physical layer level. To enable this we have made available the DTM and Radio Test firmware which is located in “peripheral” section the Nordic SDK. The DTM firmware can be used to check Packet Error Rate (PER) between a “Golden” unit and the unit under test. In the case of full DTM testing, the use of a commercial Tester would be needed. There are many companies that make testers for BTLE and a sample list is available at the end of this paper. For standard FCC testing the “Radio_Test” example would use a spectrum analyzer to examine transmitter and receiver compliance for regulatory agency approval.

To program the nRF5x family using the SWD lines they must be included in the hardware design and made available to the outside world. This is a two wire interface and in production these lines are used just once or twice depending on how the unit is tested and the firmware used. To that end a bed of nails / pogo pin scheme may be used which may be similar to that offered by [TAG-CONNECT](#).

These examples use the UART Interface and those lines must also be made available. This can be either a 2 wire or 4 wire schemes depending on the use of flow control (or not). Too often the availability is not thought out before hand and the I/O in the example has been used for some other function. So this means that the I/O will need to be mapped out to unused pins for test purposes. The NRF5x family has a I/O cross bar multiplexer which allows any GPIO to be remapped to Pin.

Please note that the I/O designator **is not** the same as the Package pin.

The following snippet of code is from the Blinky example.

It can be found in the PCA10028.h file. It shows the GPIO connected to different peripherals and devices such as the UART, Buttons, and the 4 LEDs.

This code configures the 4 LEDs for use with the Dev kit board.

```

// LEDs definitions for PCA10028
#define LEDS_NUMBER    4

#define LED_START      21
#define LED_1          21
#define LED_2          22
#define LED_3          23
#define LED_4          24
#define LED_STOP       24

```

Stock:

```

// LEDs definitions for PCA10028
#define LEDS_NUMBER    4

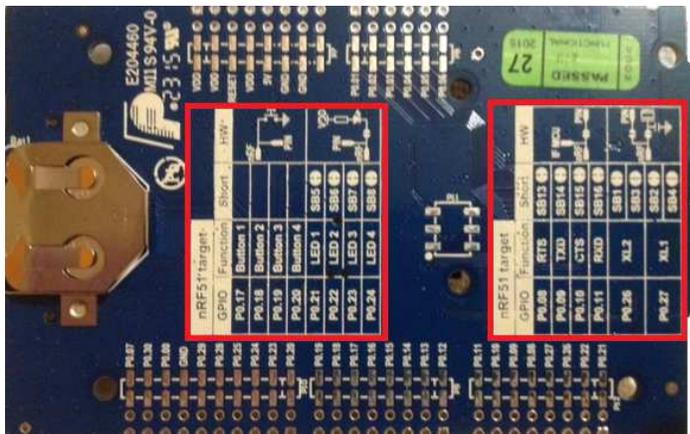
#define LED_START      17
#define LED_1          17
#define LED_2          18
#define LED_3          19
#define LED_4          20
#define LED_STOP       20

```

Modified

The numbers 21, 22, 23 and 24 are the P0.x numbers. Again, this should not be confused with the actual physical pin (or Ball) on the package. This method allows the code to be used on various package types without rewriting the code.

You can see these GPIOs listed on the bottom of the Dev Kit. However if you wanted to swap them around or change one of them to another GPIO you Just need change the number.



In reviewing the PIN – I/O list on the package you can see GPIO # 21, 22, 23, 24 utilize pins 40, 41, 42, 43 respectively. If you wanted to move them to **pins** 25,26,27,28 you would call them out as 17,18,19,20 in the code.

Note: once this is changed in the PCA10028.h file, this will be the default setup for all examples unless changed.

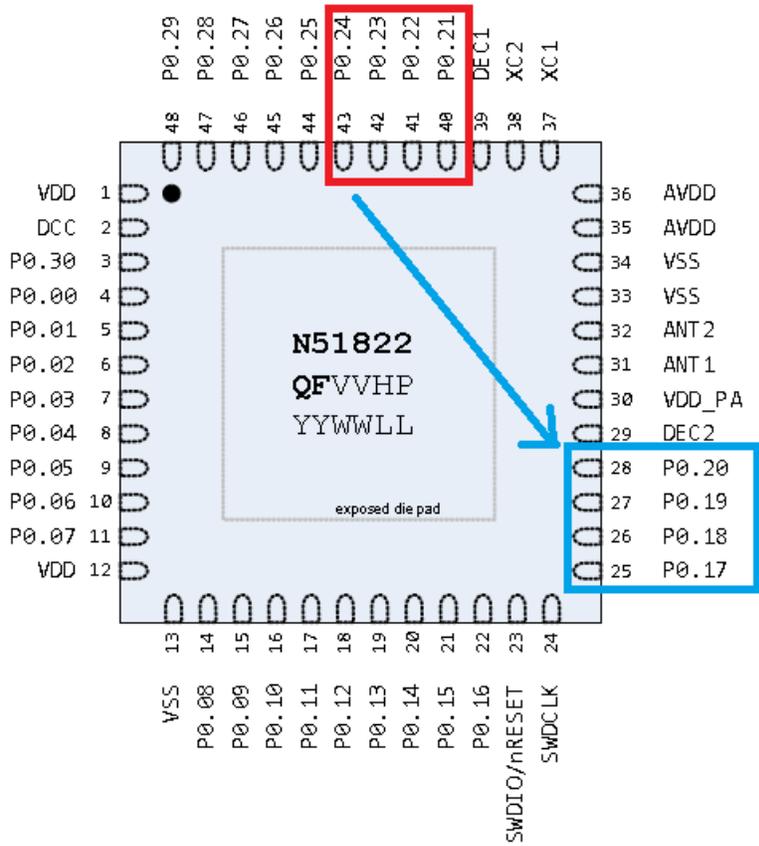


Figure 2 Pin assignment - QFN48 packet

Note: It is up to you to be sure there are no other devices connected the new I/O.

Another example would be the UART interface. This is the one that is used for the DTM and Radio_Test example.

As shown on the bottom of the nRF51-DK kit we use P0.8, 9, 10, 11.

These may need to be moved so you can redefine them to **any unused I/O**.

Note that there may be other defined functions using the same I/O but unless the Example application is using them you don't need to change or delete them.

Again in the PCA10028.h file showing the UART GPIOs.

```

#define RX_PIN_NUMBER 11
#define TX_PIN_NUMBER 9
#define CTS_PIN_NUMBER 10
#define RTS_PIN_NUMBER 8
#define HWFC true

```

Stock:

```

#define RX_PIN_NUMBER 19
#define TX_PIN_NUMBER 20
#define CTS_PIN_NUMBER 21
#define RTS_PIN_NUMBER 22
#define HWFC true

```

Modified:

Shown here are the UART Signals moved to the other side of the chip. Bears repeating that any GPIO that was not being used could have been chosen for the UART.

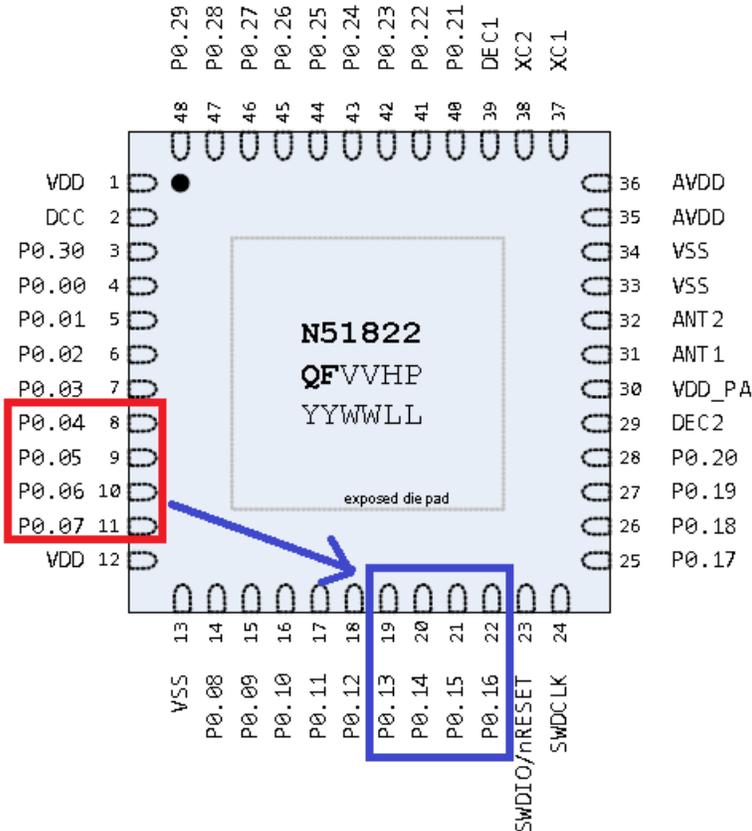


Figure 2 Pin assignment - QFN48 packet

```
C:\nRF52 - SDK\nRF52-SDK_11.0.0 - New\examples\peripheral\radio_test\pca10040\arm5_no_packs\radio_test_pca10040.uvprojx - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
nrf52832_xxaa
Project: radio_test_pca10040
  nrf52832_xxaa
    Application
      main.c
      app_error.h
      app_error_weak.h
      app_uart.h
      app_util.h
      app_util_platform.h
      boards.h
      bsp.h
      cmsis_armcc.h
      compiler_abstraction.h
      core_cm4.h
      core_cmFunc.h
      core_cmInstr.h
      core_cmSimd.h
      nordic_common.h
      nrf.h
      nrf51_to_nrf52.h
nrf52_bitfields.h
pca10040.h
main.c
56 #define BSP_BUTTON_3 BUTTON_4
57
58 #define BSP_BUTTON_0_MASK (1<<BSP_BUTTON_0)
59 #define BSP_BUTTON_1_MASK (1<<BSP_BUTTON_1)
60 #define BSP_BUTTON_2_MASK (1<<BSP_BUTTON_2)
61 #define BSP_BUTTON_3_MASK (1<<BSP_BUTTON_3)
62
63 #define BUTTONS_MASK 0x001E0000
64
65 #define RX_PIN_NUMBER 8 // on CIAA is ball G7
66 #define TX_PIN_NUMBER 6 // on CIAA is ball F7
67 #define CTS_PIN_NUMBER 7 // on CIAA is ball F6
68 #define RTS_PIN_NUMBER 5 // On CIAA is ball E7
69 #define HWFC true
70
71 #define SPIS_MISO_PIN 28 // SPI MISO signal.
72 #define SPIS_CSN_PIN 12 // SPI CSN signal.
73 #define SPIS_MOSI_PIN 25 // SPI MOSI signal.
74 #define SPIS_SCK_PIN 29 // SPI SCK signal.
75
76 #define SPIMO_SCK_PIN 29 // SPI clock GPIO pin number.
77 #define SPIMO_MOSI_PIN 25 // SPI Master Out Slave In GPIO pin numbe
78 #define SPIMO_MISO_PIN 28 // SPI Master In Slave Out GPIO pin numbe
79 #define SPIMO_SS_PIN 12 // SPI Slave Select GPIO pin number.
```

How it looks full page.

Direct Test Mode

Now it is time to start your DTM testing (Direct Test Mode). Very few people own a very expensive piece of equipment like a Litepoint, Rohde-Schwarz, Keysight, etc. To that end Nordic has made available a DTM example in our SDK. This may be accomplished with two of our development kits. The only additional hardware you need is a RF Attenuator. This test may be used with the nRF51 or nRF52 family of products.

The main use of the DTM as we are using it here is to identify units that don't work or have some marginal performance due to assembly.

We can practice this by using the 2 development kits mentioned above. At this time you can hook up your two development kits to your computer for programming... Please note that this is using the SDK version 11.0.0.

1. Download the nAN-34 App note and the python scripts from the product page, using link. https://www.nordicsemi.com/eng/nordic/download_resource/20649/8/14776798

2. Flash both the DUT and the nRF52 DK kit with the DTM Example \11.0.0-2_alpha\examples\dtm\direct_test_mode\pca100xx\blank\arm5_no_packs. Xx is PCA board type

2a. Use SDK version 11.0 for the nRF52-DK and the nRF51-DK

2b. Use the proper project for the -DK version being used. (PCA10028, PCA10036, PCA10040)

2c. compile the projects and flash devices.

3. Open the example.py python script found in the Python folder in the nAn34_v1.01 zip file.

Edit the TestSerialPortName and GoldenSerialPortName to match the COM-ports that the DUT and nRF51 DK are connected to. (Open Device Manager >> Ports to see the COM port number)

4. If you do not have Python, download it from and add python.exe to your Windows path. <https://www.python.org/downloads/release/python-2711/>

5. Unzip attached Serial.zip file into Python directory.
You can also find the serial port extension here
<https://pypi.python.org/pypi/pyserial>

To run the tests just open a terminal window in the nAn34_v1.01/Python folder and type in the following.

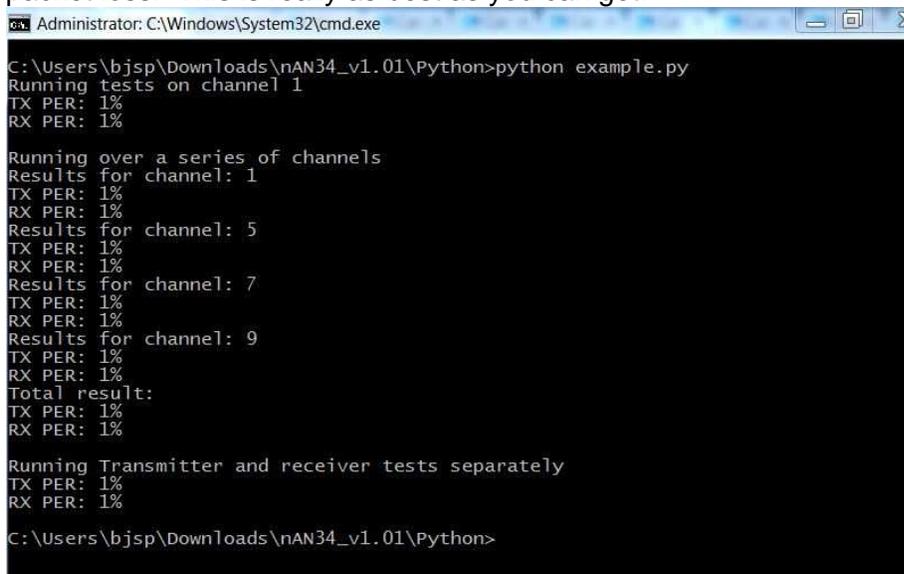
Python example.py

If this does not run either move the items downloaded in step 1 to the Python subdirectory or change the path per the instructions found here.

<http://stackoverflow.com/questions/6318156/adding-python-path-on-windows-7>

Photos below were taken while using the nRF51-DK. The physical test setup is the same for the nRF52-DK

With 0db attenuation on the Golden Unit the output should be like below. As shown there is 1% packet loss. This is really as best as you can get.



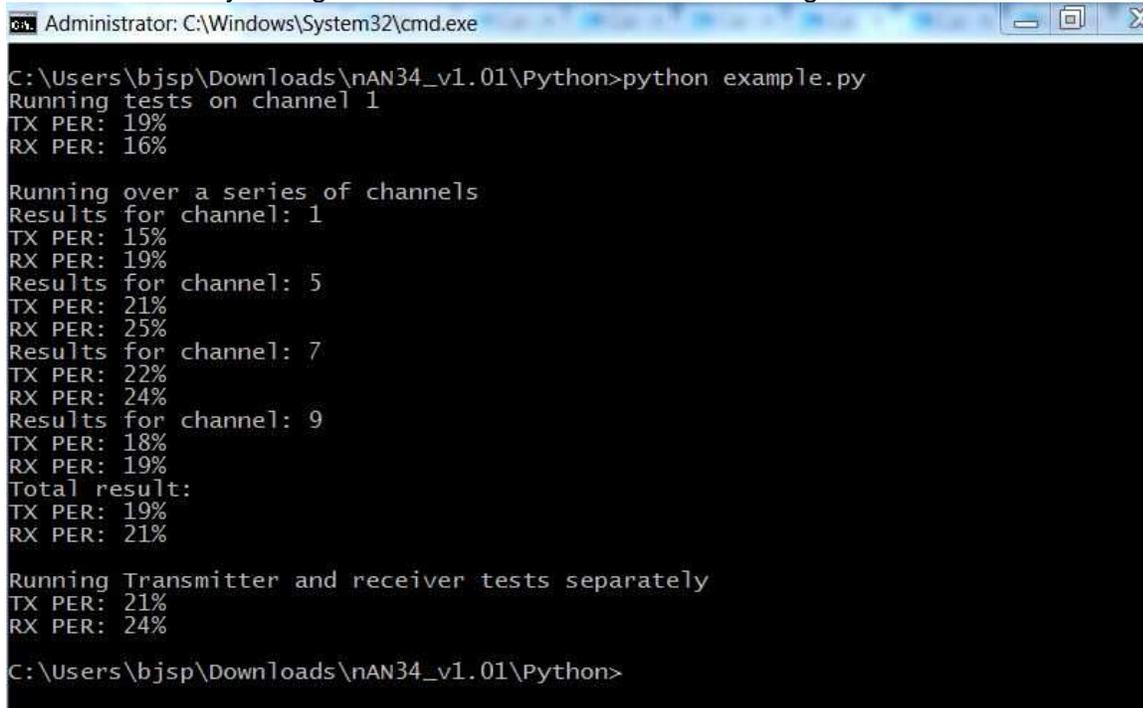
```
Administrator: C:\Windows\System32\cmd.exe
C:\Users\bjsp\Downloads\NaN34_v1.01\Python>python example.py
Running tests on channel 1
TX PER: 1%
RX PER: 1%

Running over a series of channels
Results for channel: 1
TX PER: 1%
RX PER: 1%
Results for channel: 5
TX PER: 1%
RX PER: 1%
Results for channel: 7
TX PER: 1%
RX PER: 1%
Results for channel: 9
TX PER: 1%
RX PER: 1%
Total result:
TX PER: 1%
RX PER: 1%

Running Transmitter and receiver tests separately
TX PER: 1%
RX PER: 1%

C:\Users\bjsp\Downloads\NaN34_v1.01\Python>
```

Adjust the attenuation level so that the packet loss is close to 30%, i.e. the DUT is barely passing. With the nRF51 DK, I had to set the attenuation to 50dB, see the screenshot below. Please note that you might have to set the attenuation level higher or lower.



```
Administrator: C:\Windows\System32\cmd.exe
C:\Users\bjsp\Downloads\nAN34_v1.01\Python>python example.py
Running tests on channel 1
TX PER: 19%
RX PER: 16%

Running over a series of channels
Results for channel: 1
TX PER: 15%
RX PER: 19%
Results for channel: 5
TX PER: 21%
RX PER: 25%
Results for channel: 7
TX PER: 22%
RX PER: 24%
Results for channel: 9
TX PER: 18%
RX PER: 19%
Total result:
TX PER: 19%
RX PER: 21%

Running Transmitter and receiver tests separately
TX PER: 21%
RX PER: 24%

C:\Users\bjsp\Downloads\nAN34_v1.01\Python>
```

If the attenuation level is too high then no packets will go through correctly, i.e. around 100% percent packet loss, for the nRF51 Dk this was 55dB. In this case back off the attenuation until it

passes.

```
Administrator: C:\Windows\System32\cmd.exe
C:\Users\bjsp\Downloads\NaN34_v1.01\Python>python example.py
Running tests on channel 1
Transmitter test Packeterrorrate to high at 98%
TX PER: N/A%
RX PER: N/A%

Running over a series of channels
Transmitter test Packeterrorrate to high at 97%
Results for channel: 1
TX PER: N/A%
RX PER: N/A%
Transmitter test Packeterrorrate to high at 100%
Results for channel: 5
TX PER: N/A%
RX PER: N/A%
Transmitter test Packeterrorrate to high at 99%
Results for channel: 7
TX PER: N/A%
RX PER: N/A%
Transmitter test Packeterrorrate to high at 98%
Results for channel: 9
TX PER: N/A%
RX PER: N/A%
Total result:
TX PER: 100%
RX PER: 100%

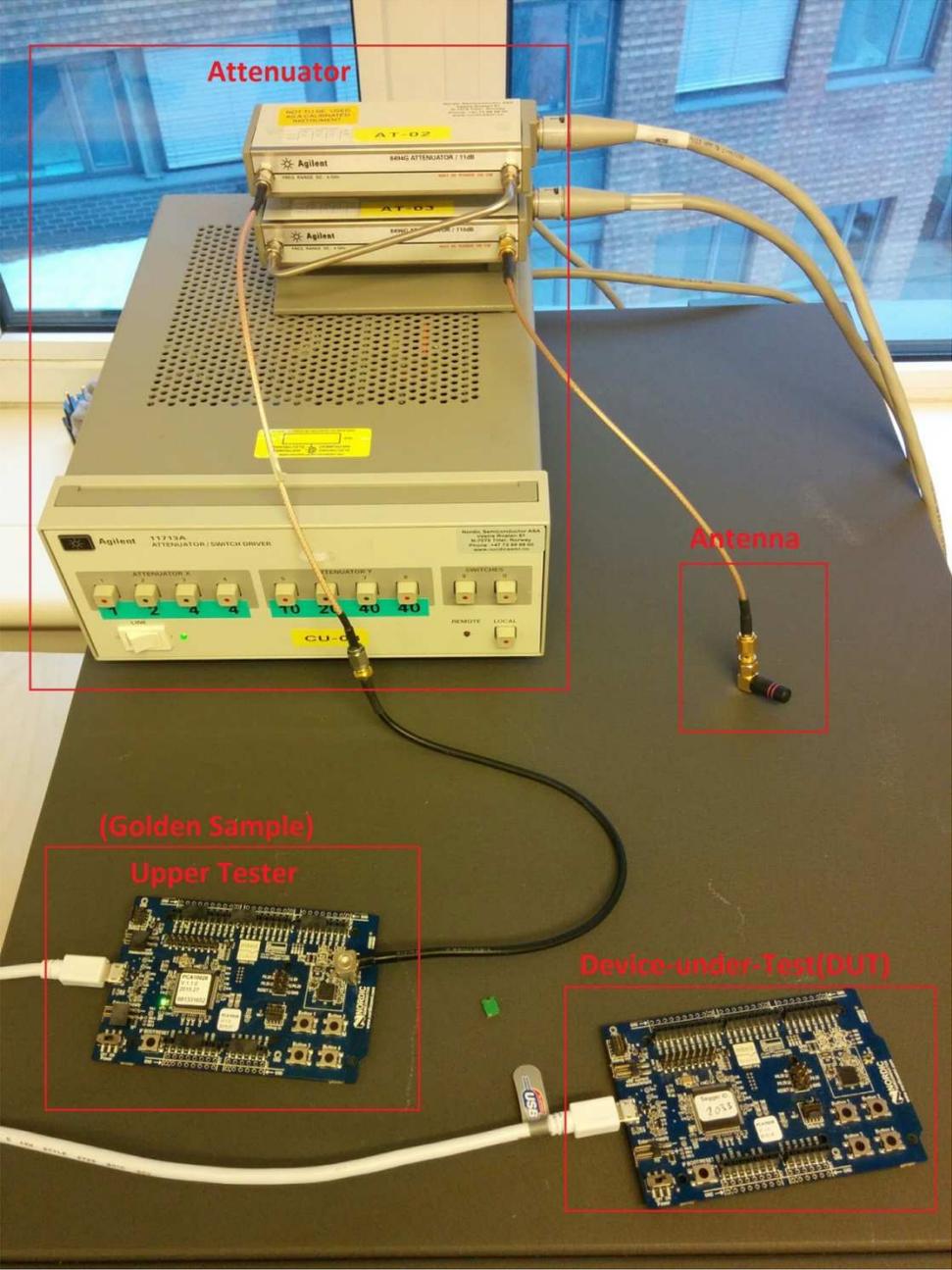
Running Transmitter and receiver tests separately
Transmitter test Packeterrorrate to high at 99%

C:\Users\bjsp\Downloads\NaN34_v1.01\Python>
```

The Golden unit and DUT may be mounted on a Jig that allows repeatability. The amount of Attenuation will need to be adjusted and will be different depending on the antennas and the distance between the two units. Note, as the units are both transmitting and receiving, an attenuator must be used. Just lowering the output power is not satisfactory.

Open air setup with programmable attenuation

Testing in chamber

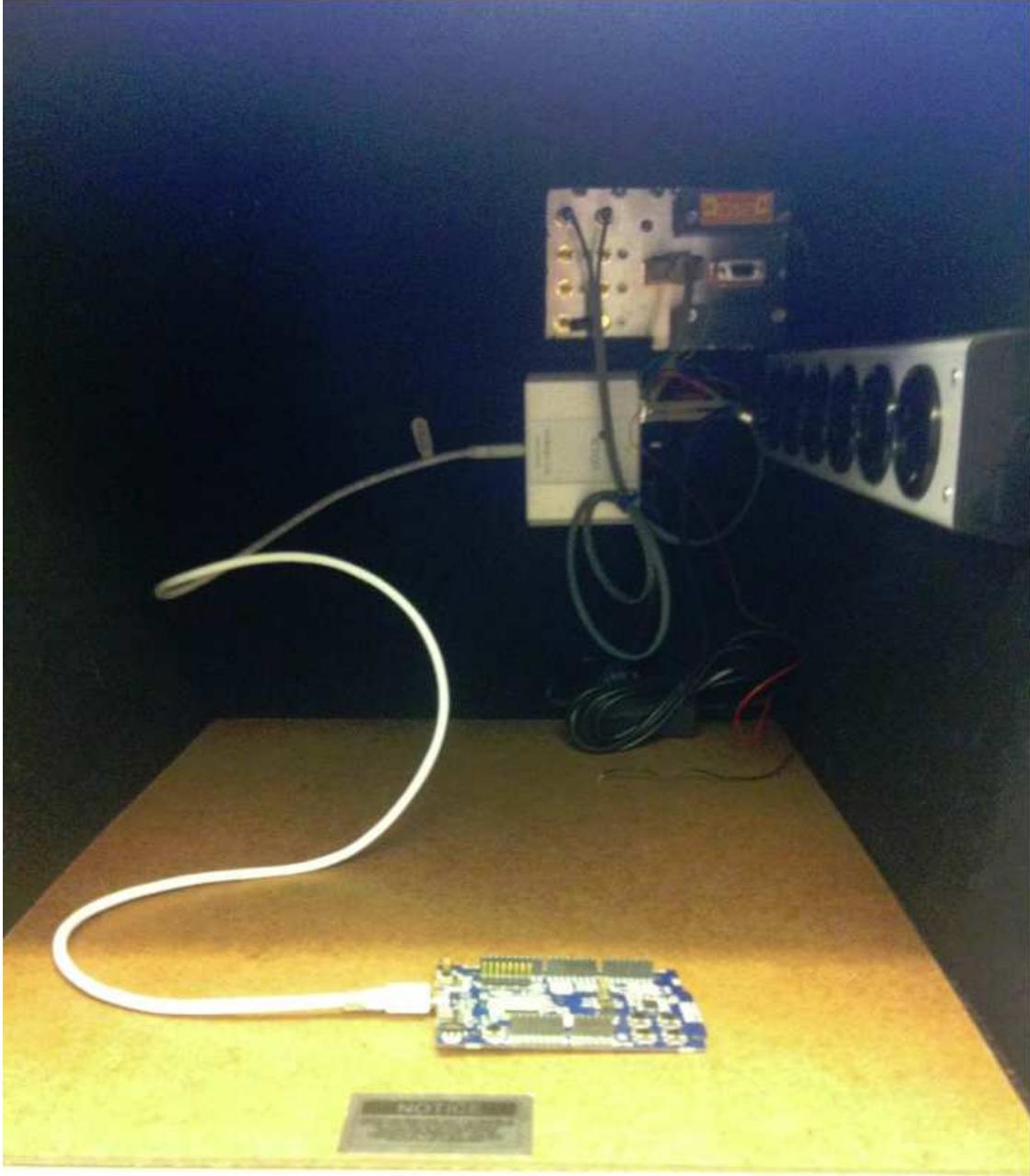


Attenuator

Antenna

(Golden Sample)
Upper Tester

Device-under-Test(DUT)



The author is hopeful that this brief tutorial on GPIO configuration and DTM testing was of value.