AWS Data Comparison

11-27-2023
14:14

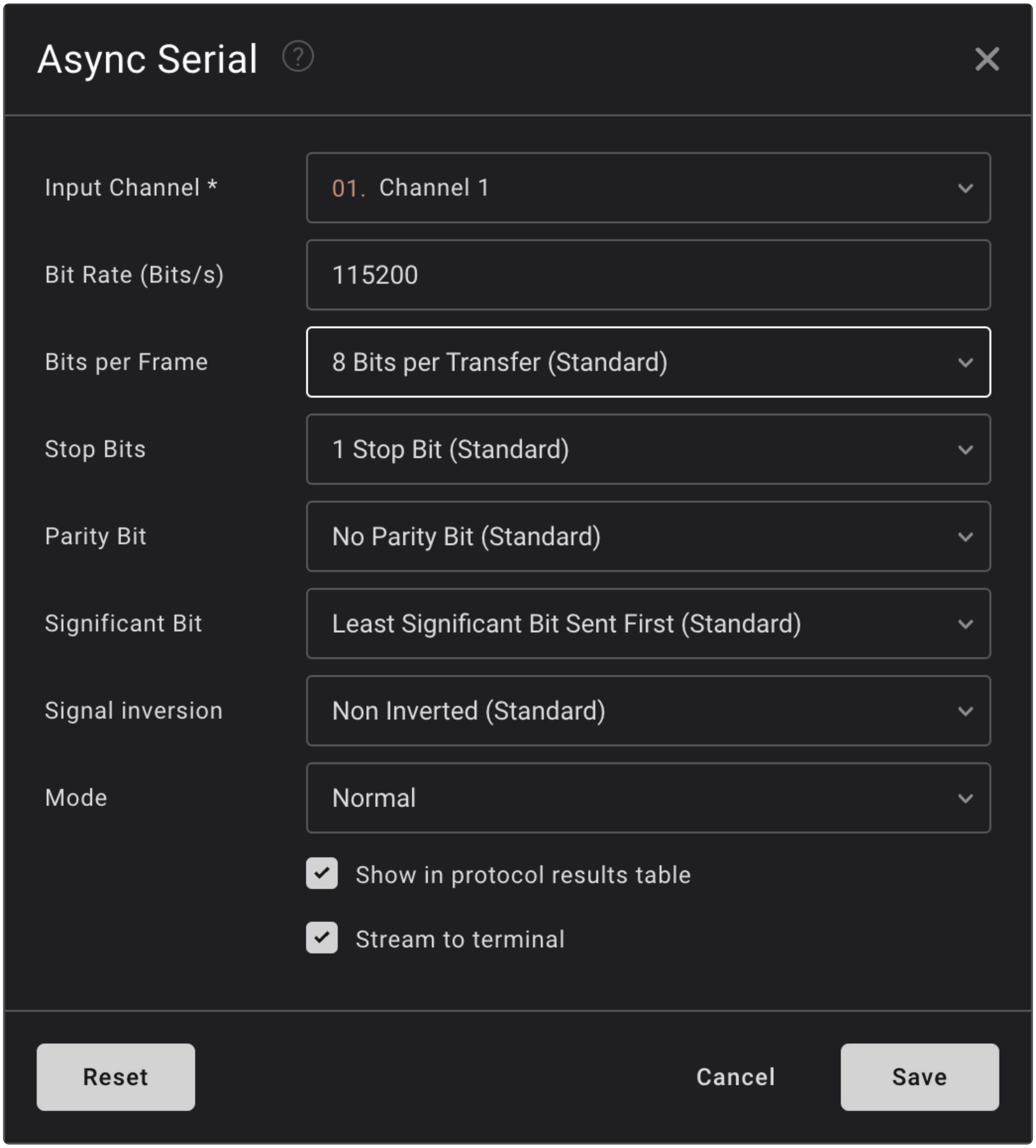
Motivation

For all of our previous tests where we have wanted to record the values that the SAADC outputs while experimenting with the Battery Voltage we have used an AWS Terminal application. It uses LTE to send the values that the SAADC using the MQTT protocol. This application uses the same process to measure the signal voltage as the ADC Terminal application. In AWS I have created DynamoDB databases to store the data from the nRF9160dk. I also use message routing rules to transfer the messages from MQTT to the DynamoDB tables.

We wanted to confirm that the data sent out by the nRF9160dk was not being modified or manipulated by AWS so we needed a way to save the application output. We decided to used the default Serial Wire Debug(SWD) GPIO pin interface along with a signal analyzer to record the data sent out by the nRF9160dk. The AWS Terminal logs the message that will be sent to AWS, meaning that the data will be encoded in the SWD. The signal analyzer was necessary to decode the SWD protocol data into plaintext.
With these two sources we would be able to compare what was recoded and determine if there was any manipulation or modification in AWS

Test

For this test we would use the Saleae Logic 4 analyzer along with the Logic 2 desktop application to record the SWD and decode it into plain text. We would run a shorter version of our ADC Battery Test. We used three Battery Voltages: 3.5V, 3.6V, 3.7V and three weights: 100g, 200g, 340g. The entire device would be powered on for 5 minutes. The Saleae Logic 4 analyzer was connected to P0.28(TX) and P0.29(RX) as well as ground on the nRF9160dk. I then configured the analyzer in the Logic 2 software with the Async Serial Analyzer with input 1 and 2 to match the P0.28(TX) and P0.29(RX).

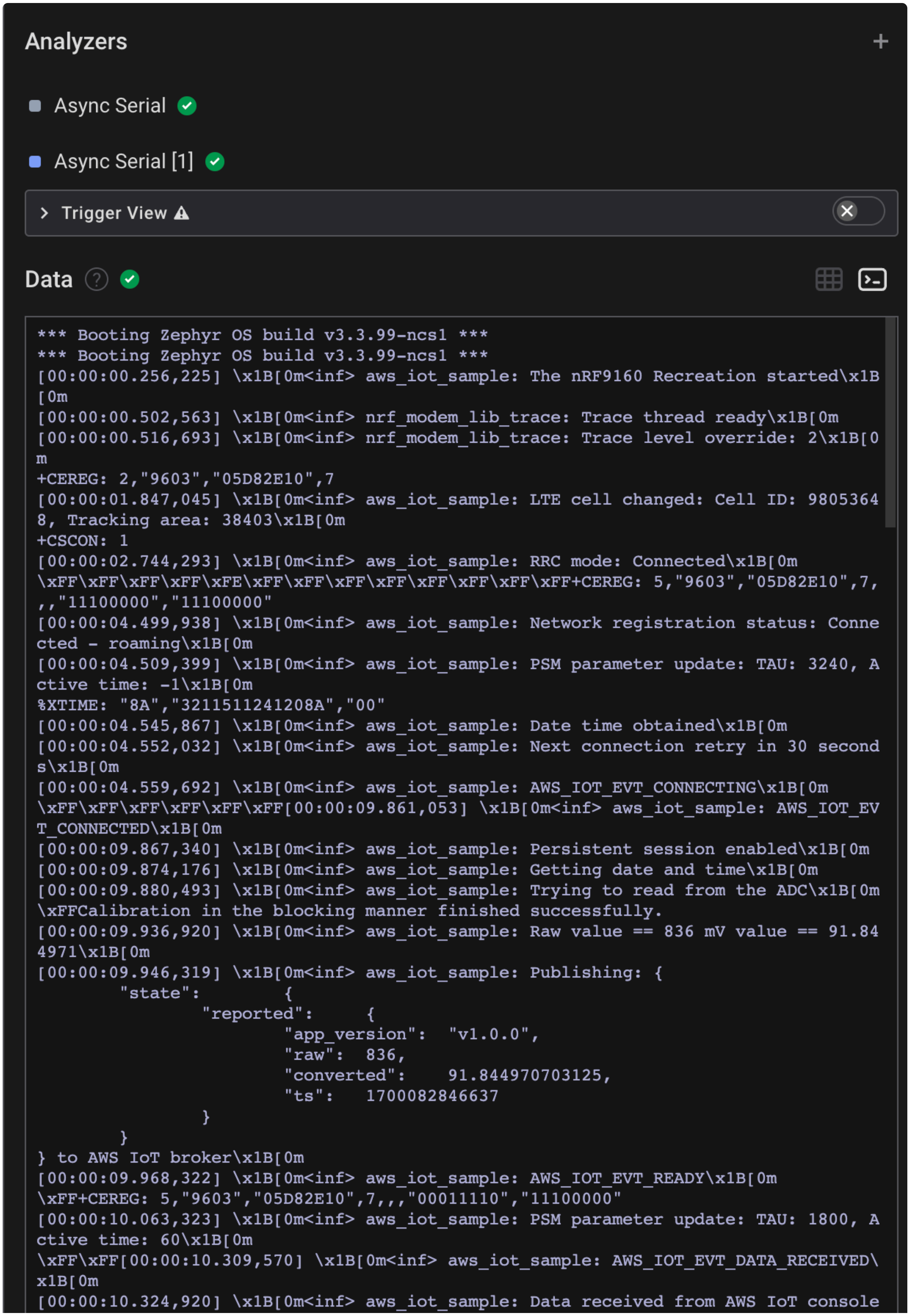

Async Serial

Input Channel *	01. Channel 1
Bit Rate (Bits/s)	115200
Bits per Frame	8 Bits per Transfer (Standard)
Stop Bits	1 Stop Bit (Standard)
Parity Bit	No Parity Bit (Standard)
Significant Bit	Least Significant Bit Sent First (Standard)
Signal inversion	Non Inverted (Standard)
Mode	Normal

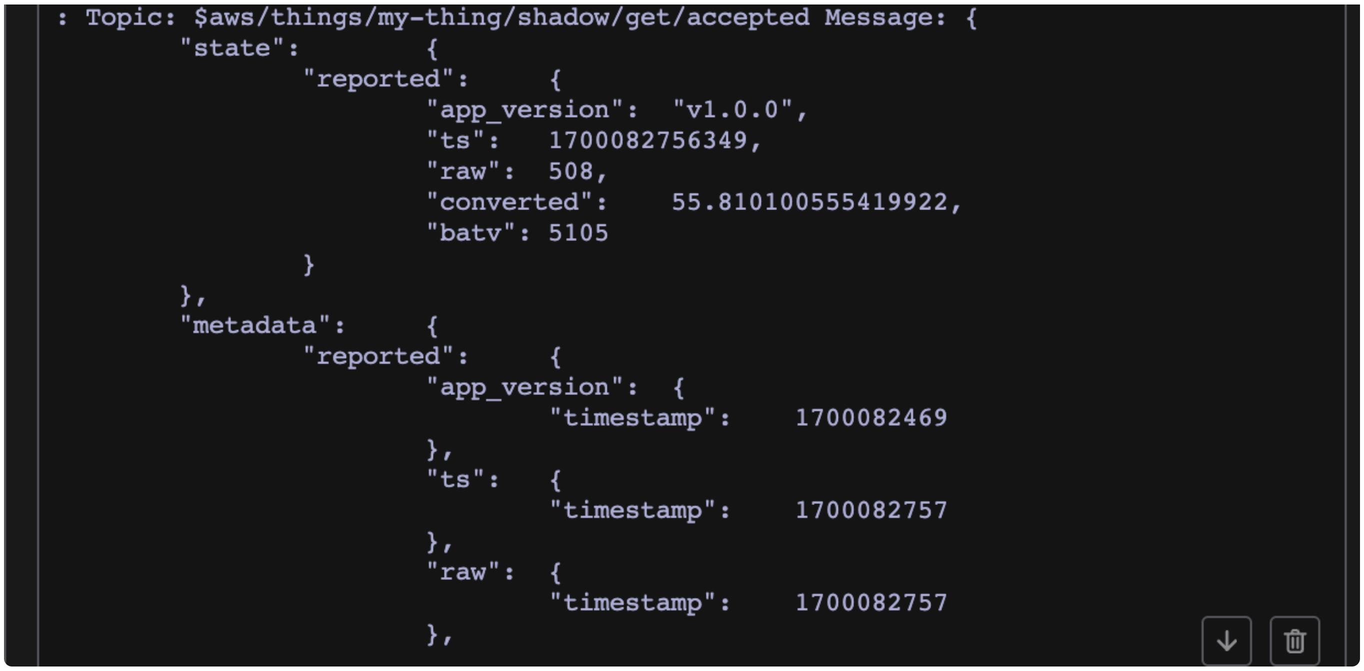
✓ Show in protocol results table

✓ Stream to terminal

After running the test I would save the Logic 2 capture so we could reference it later. We can now use the Async Serial Analyzer to convert the encoded voltage signal from P0.28(TX) and P0.29(RX) into plain text.



```
*** Booting Zephyr OS build v3.3.99-ncs1 ***
*** Booting Zephyr OS build v3.3.99-ncs1 ***
[00:00:00.256,225] \x1B[0m<inf> aws_iot_sample: The nRF9160 Recreation started\x1B
[0m
[00:00:00.502,563] \x1B[0m<inf> nrf_modem_lib_trace: Trace thread ready\x1B[0m
[00:00:00.516,693] \x1B[0m<inf> nrf_modem_lib_trace: Trace level override: 2\x1B[0
m
+CEREG: 2,"9603","05D82E10",7
[00:00:01.847,045] \x1B[0m<inf> aws_iot_sample: LTE cell changed: Cell ID: 9805364
8, Tracking area: 38403\x1B[0m
+CSCON: 1
[00:00:02.744,293] \x1B[0m<inf> aws_iot_sample: RRC mode: Connected\x1B[0m
\xFF\xFF\xFF\xFF\xFE\xFF\xFF\xFF\xFF\xFF\xFF\xFF\xFF+CEREG: 5,"9603","05D82E10",7,
,,"11100000","11100000"
[00:00:04.499,938] \x1B[0m<inf> aws_iot_sample: Network registration status: Conne
cted - roaming\x1B[0m
[00:00:04.509,399] \x1B[0m<inf> aws_iot_sample: PSM parameter update: TAU: 3240, A
ctive time: -1\x1B[0m
%XTIME: "8A","3211511241208A","00"
[00:00:04.545,867] \x1B[0m<inf> aws_iot_sample: Date time obtained\x1B[0m
[00:00:04.552,032] \x1B[0m<inf> aws_iot_sample: Next connection retry in 30 second
s\x1B[0m
[00:00:04.559,692] \x1B[0m<inf> aws_iot_sample: AWS_IOT_EVT_CONNECTING\x1B[0m
\xFF\xFF\xFF\xFF\xFF\xFF[00:00:09.861,053] \x1B[0m<inf> aws_iot_sample: AWS_IOT_EV
T_CONNECTED\x1B[0m
[00:00:09.867,340] \x1B[0m<inf> aws_iot_sample: Persistent session enabled\x1B[0m
[00:00:09.874,176] \x1B[0m<inf> aws_iot_sample: Getting date and time\x1B[0m
[00:00:09.880,493] \x1B[0m<inf> aws_iot_sample: Trying to read from the ADC\x1B[0m
\xFFCalibration in the blocking manner finished successfully.
[00:00:09.936,920] \x1B[0m<inf> aws_iot_sample: Raw value == 836 mV value == 91.84
4971\x1B[0m
[00:00:09.946,319] \x1B[0m<inf> aws_iot_sample: Publishing: {
        "state":        {
                "reported":     {
                        "app_version":  "v1.0.0",
                        "raw":  836,
                        "converted":    91.844970703125,
                        "ts":   1700082846637
                }
        }
} to AWS IoT broker\x1B[0m
[00:00:09.968,322] \x1B[0m<inf> aws_iot_sample: AWS_IOT_EVT_READY\x1B[0m
\xFF+CEREG: 5,"9603","05D82E10",7,,,"00011110","11100000"
[00:00:10.063,323] \x1B[0m<inf> aws_iot_sample: PSM parameter update: TAU: 1800, A
ctive time: 60\x1B[0m
\xFF\xFF[00:00:10.309,570] \x1B[0m<inf> aws_iot_sample: AWS_IOT_EVT_DATA_RECEIVED\
x1B[0m
[00:00:10.324,920] \x1B[0m<inf> aws_iot_sample: Data received from AWS IoT console
```



```
: Topic: $aws/things/my-thing/shadow/get/accepted Message: {
        "state":        {
                "reported":     {
                        "app_version":  "v1.0.0",
                        "ts":   1700082756349,
                        "raw":  508,
                        "converted":    55.810100555419922,
                        "batv": 5105
                }
        },
        "metadata":     {
                "reported":     {
                        "app_version":  {
                                "timestamp":    1700082469
                        },
                        "ts":   {
                                "timestamp":    1700082757
                        },
                        "raw":  {
                                "timestamp":    1700082757
                        },
```

I then copied and pasted this into text files so that I could use a python script to grab the message data and ignore the other console log messages.

ExtractPublishing.py

After isolating the message data, the majority of this code is formatting the the data into a CSV. This is the same format downloaded from AWS.

To analyze this data I created a second python script. This script reads in the CSV files from both AWS and Saleae Logic Analyzer. It uses dictionaries to store the messages data in a 2d array to represent the CSV. We then compare each item in the Saleae data array to those in the AWS array. There are two comparison tables, the first one has tuples in the form of (match, data) and the second just contains whether there was a match. If there is a match then the tables set a 1 for the match variable. This gives us two matching tables, the condensed one is great for getting a overview of what data was is the same, and the longer table allows us to understand which messages did not match between the Saleae Logic 4 and AWS.

Results

Condensed Results, 1 = match 0 = no match

3.5V,100g	3.5V,200g	3.5V,340g	3.6V,100g	3.6V,200g	3.6V,340g	3.7
1	1	1	0	1	1	1
1	1	1	0	1	1	1
1	1	1	0	1	1	1
1	1	1	0	1	1	1
1	1	1	0	1	1	1
1	1	1	0	1	1	1
1	1	1	0	1	1	1
1	1	1	0	1	1	1
1	1	1	0	1	1	1
1	1	1	0	1	1	1
1	1	1	0	1	1	1
1	1	1	0	1	1	1
1	1	1	0	1	1	1
1	1	1	0	1	1	1
1	1	1	0	1	1	1
1	1	1	0	1	1	1
1	1	1	0	1	1	1
1	1	1	0	1	1	1
1	1	1	0	1	1	1
1	1	1	0	1	1	1
1	1	1	0	1	1	1

From the data above we can see that most of the data in theSaleae Logic 4 was found in AWS however we weren't able to find any matches for 3.6V with 100 grams. This is because 3.6V at 100 grams was the first test I did and after finishing I realized that I had forgot to turn on the AWS message routing rule. This means that the messages were received in AWS but weren't forwarded to the DynamoDB table. I reran the test but forgot to save the new capture from the Saleae Logic 2 application. This resulted in AWS having the new messages for 3.6V, 100g and the old Saleae Logic 2 messages. The data supports this since it is the only test that wasn't matched and the entire test is missing matching data.

From this data I am certain that there is no data manipulation happening in AWS. We can repeat this test to verify that the missing data is due to the mentioned mistake above and not any manipulation in AWS.

This test gave us two key results. First that we can trust the data that has been sent to AWS, and second we now have a new way of collecting data from the nRF9160dk without needing to connect to the cloud. That was previously not possible because our old way of connecting to SWD was over a USB connection. This would also provide 5V power source to the nRF9160dk. Providing a 5V power supply to the device would cancel out any Battery Supply Voltage manipulation we performed.