Project Progress report: January 2013

# Previous work

My most recent Project Proposal (14/11/2012) outlined the following work that had been carried out on the project:

1. An informal but in-depth code examination has been carried out on the current telemetry device software. The knowledge gained from this will allow for an in-depth review of the software as it stands, giving a benchmark that can be used to measure the performance of the proposed system (exact performance metrics are yet to be established).
2. A review of the current hardware and software functionality has been carried out including cost analysis, vehicle system integration and future-casting.
3. Developed a tool / script in C to analyse CAN data logs from the vehicles. This has given an insight into the behaviour of the CAN bus on live vehicles, but has been configured around a single-message acceptance filter. Further work is required to adapt this simulation to include the multi-filter scheme proposed in this document.

# Lessons and Conclusions from Previous Work

As this is the first progress report I have written for this project, I shall go into a little more detail on the work done prior to this month.

To provide a benchmark for success, a CAN trace was sent to an existing Remote Device (RD) several times to ascertain how many times the each ID was logged. The device firmware was modified to output, to its debug port, the ID of every message it saw on the CAN bus. This output was logged and a simple script was used to count the occurrences of each CAN ID. The results of these tests can be seen in Table 2.

It can be seen that the capture rate per ID is very unpredictable in nature. This is most likely due to the 2 message buffers on the device’s CAN transceivers becoming full, blocking other new messages, before the firmware polls them.

The first CAN analysis script in (3) above, was based around a single-ID filter, and used the time stamp on each message in the CAN trace to predict whether or not that message would have been logged by the dynamic filtering system. This script became over-complicated as I added more and more factors to try and get the most realistic results (source code available on request). It did, however, produce some useful information about the timing of each CAN ID in the CAN trace, including the variation in ‘period’ of each ID (see Table 3).

A logging sequence was found by recording the order in which CAN ID’s first appeared in the trace. The script then operated on the assumption that a message would be ‘caught’ if it fell before or on its target time, and if the previous ID matched the ID that preceded it in the sequence. This target was based on the period of the message. The high level of variation in timing (particularly in the 0x7nn series of ID’s) meant that the single-ID acceptance filter could not be guaranteed to catch as many messages as the existing firmware. For example, ID 0x709 had a capture rate of only 15.89% (Table 3), compared to the 87.75 – 89.63% rate of the existing firmware (Table 2). Although this capture rate was not high enough, the script did demonstrate that, unlike the existing firmware, the messages captured using dynamic filtering would produce the same results every time, bringing predictability to the system.

# Work this month

This month, I have spent time re-working the CAN analysis script from (3) above to simulate the filtering system in my most recent proposal, utilising multiple acceptance filters. As the single-ID filter could not cope well with the varying sequence on the CAN bus, it was hoped that allowing a block of different ID’s at any one time, and replacing those ID’s with new ones as they appeared on the bus, would have a higher success rate.

## Script

The revised script follows process below:

* **Find logging sequence** – Finds all unique ID’s in the CAN trace that are included in the logging list. These ID’s are arranged into a sequence. For now, this is the order that ID’s first appear on the CAN bus.
* **Count sequence** – Counts the number of unique ID’s in the Logging Sequence.
* **Check logability** – This simulates the multi-ID acceptance filter, and performs the main ‘logability’ analysis on the CAN trace. It functions as follows:
  + A time-triggered, periodical logging task is simulated that, in a real embedded system, would read all logged messages from the CAN buffer, and update the acceptance filter. The period of this simulated task is controlled by “LOGGING\_TASK\_PERIOD\_us”.
  + The number of ID’s in the acceptance filter (represented by the array, “acceptanceFilter[]”) is configurable with the argument, “filterSize”.
  + acceptanceFilter[] is loaded from the top of the loggingSequence[] array up to the size, filterSize.
  + A variable, “sequencePointer”, is used to keep track of the location in loggingSequence[].
  + The CAN trace is read, line-by-line, and each CAN ID is extracted.
  + The CAN ID is first checked to see if it falls in the Logging List.
    - If the ID is in the Logging List, acceptanceFilter[] is interrogated to see if the ID is present.
      * If the ID is present in the acceptance filter, the ID has been ‘captured’.
      * A counter relating to the captured ID is incremented, as is a general “IDLogCount”.
      * The ID is marked as ‘logged’ in the acceptance filter.
    - If the ID is not present in the acceptance filter, the ID has been ‘missed’ and IDMissedCount is incremented.
  + The timestamp of each message is interrogated to identify when the simulated logging task should run. When LOGGING\_TASK\_PERIOD\_us has expired, each ID in the acceptance filter that has been marked as ‘logged’ is replaced by the next ID in the logging sequence that isn’t already present in the acceptance filter.

## Tests

Below are descriptions of the tests carried out using the analysis script:

### Initial Tests

Early iterations of the script displayed very poor results, with capture rates of only ~20%. This was found to be due to the logic that replaced logged ID’s in the acceptance filter. Originally, the script would replace any logged ID with the next ID in the sequence, without checking to see if that ID was already in the acceptance filter. This meant that the acceptance filter quickly filled up with repeats of the less frequent ID’s in the trace file, blocking the more frequent ID’s from being captured.

The script was then modified to replace the logged ID’s with the next ID in the logging sequence that *wasn’t already included in the acceptance filter.* This change allowed me to obtain much more encouraging results.

### Filter Size Variation

This test was used to investigate the effect of the number of ID’s loaded into the Acceptance Filter at any one time.

* “LOGGING\_TASK\_us” was kept at a constant arbitrary value of 1000 µs
* “filterSize” was varied from 1 to 32 (the total number of ID’s in the logging sequence).

The results of this test can be seen below:

Table : Logged and missed messages for

varying filter size

|  |  |  |
| --- | --- | --- |
| filterSize | Logged | Missed  Figure : TOTAL MESSAGES LOGGED VS ACCEPTANCE FILTER SIZE |
| 1 | 51895 | 1292354 |
| 2 | 122888 | 1221361 |
| 3 | 224627 | 1119622 |
| 4 | 297623 | 1046626 |
| 5 | 300914 | 1043335 |
| 6 | 397833 | 946416 |
| 7 | 558586 | 785663 |
| 8 | 787651 | 556598 |
| 9 | 921503 | 422746 |
| 10 | 1203899 | 140350 |
| 11 | 1281245 | 63004 |
| 12 | 1297853 | 46396 |
| 13 | 1315810 | 28439 |
| 14 | 1333643 | 10606 |
| 15 | 1339364 | 4885 |
| 16 | 1338773 | 5476  Figure : TOTAL MESSAGES LOGGED VS ACCEPTANCE FILTER SIZE (DETAIL) |
| 17 | 1336931 | 7318 |
| 18 | 1336142 | 8107 |
| 19 | 1335824 | 8425 |
| 20 | 1335596 | 8653 |
| 21 | 1335426 | 8823 |
| 22 | 1334978 | 9271 |
| 23 | 1334455 | 9794 |
| 24 | 1333316 | 10933 |
| 25 | 1331130 | 13119 |
| 26 | 1327875 | 16374 |
| 27 | 1321761 | 22488 |
| 28 | 1316137 | 28112 |
| 29 | 1337313 | 6936 |
| 30 | 1343771 | 478 |
| 31 | 1344235 | 14 |
| 32 | 1344249 | 0 |

The results of this test show an upwards trend in the number of messages logged compared to the size of the acceptance filter (Figure 1). Which is to be expected, when working under the assumption that the more ID’s the acceptance filter recognises, the greater the chance of a message arriving on the CAN bus is going to satisfy the filter.

One aspect of behaviour of this test that was unexpected, however, is shown in Figure 2. When the filter size reaches 15 the capture rate peaks. After this peak, the capture rate reaches an intermediate minimum at a filter size of 28, before climbing steeply to achieve the expected 100% capture rate when the filter size equals the number of ID’s to log. The difference between this intermediate peak and trough is not insignificant (23227 messages) and the same pattern has occurred in other tests. I am yet to explain why this happens and need to review the code in my script to confirm that there are no errors in the logic around the point where the acceptance filter is updated with new ID’s. For the purposes of testing, this peak has been used as a preliminary ‘optimum’ size for the acceptance filter, being small enough for the CAN hardware to contain, and whilst offering the greatest capture rate.

### Task Period Variation

A further test carried out involved keeping the size of the acceptance filter constant, but varying the period of the logging task. From the results of the above test, the optimum filter size of 15 was used, and the logging task period varied in 500 µs increments from 500 to 2500 µs.

The results of this test can be seen in Table 4 and Figures Figure 3, Figure 4, Figure 5 and Figure 6. Here, it is shown that the capture rate increases the more frequently the acceptance filter is updated. This is due to the fact that, when the acceptance filter is updated less frequently, there is a more of a chance that ID’s will be present on the CAN bus that the filter is not looking for; the filter will still be configured to accept messages that have already arrived.

It can also be seen in these results that, compared to the existing algorithm, the spread of capture is much more consistent. For example, there is a difference of 8.027 % with the existing algorithm, compared to only 0.111 % using the script and a task period of 500 µs.

# Conclusions and plan for February

The results from the initial tests run using the simulation script are very promising. The variable acceptance filter, once configured to expect multiple ID’s, appears to offer much more consistent results than the ‘hit and miss’ method currently in use on the telemetry device. There is some unexpected behaviour from the script, namely the trend in the capture rate when varying the filter size, and it is this that I plan on focusing on in February following the steps below:

* Review the script code for errors that could explain the Filter Size Variation results.
* Read around the statistical phenomena / processes that will influence the capture rate and hopefully explain the behaviour seen in the Filter Size Variation tests if no errors are found in the code review.
* Test the effect of varying the number of expected ID’s is varied, related to the total number of unique ID’s in the CAN trace.
* Investigate the effect that the logging sequence order has on the capture rate, and how the order can be optimised either statically or dynamically.
* Further repeats of the above tests are needed using different CAN traces.

**Test Results**

Table : BENCHMARK ID COUNTS FROM EXISITING REMOTE DEVICE FIRMWARE

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CAN ID | Total |  | RD 1 | |  | RD 2 | |  | RD 3 | |  | RD 4 | |
|  |  |  | logs | Percent |  | logs | Percent |  | logs | Percent |  | logs | Percent |
| 0x187 | 45107 |  | 41178 | 91.29 |  | 40042 | 88.77 |  | 40104 | 88.91 |  | 39986 | 88.65 |
| 0x188 | 45111 |  | 38738 | 85.87 |  | 38598 | 85.56 |  | 37962 | 84.15 |  | 38390 | 85.10 |
| 0x189 | 45110 |  | 41442 | 91.87 |  | 40380 | 89.51 |  | 40229 | 89.18 |  | 40119 | 88.94 |
| 0x18A | 45111 |  | 41887 | 92.85 |  | 41006 | 90.90 |  | 40776 | 90.39 |  | 40577 | 89.95 |
| 0x18B | 45111 |  | 40011 | 88.69 |  | 38911 | 86.26 |  | 39544 | 87.66 |  | 38847 | 86.11 |
| 0x18C | 45111 |  | 37954 | 84.13 |  | 37060 | 82.15 |  | 37413 | 82.94 |  | 37426 | 82.96 |
| 0x18D | 45109 |  | 38653 | 85.69 |  | 37049 | 82.13 |  | 37054 | 82.14 |  | 37317 | 82.73 |
| 0x18E | 45111 |  | 39273 | 87.06 |  | 38167 | 84.61 |  | 38391 | 85.10 |  | 38145 | 84.56 |
| 0x207 | 45107 |  | 38858 | 86.15 |  | 38045 | 84.34 |  | 38412 | 85.16 |  | 37880 | 83.98 |
| 0x209 | 45110 |  | 41574 | 92.16 |  | 40486 | 89.75 |  | 40345 | 89.44 |  | 40342 | 89.43 |
| 0x20B | 45111 |  | 41078 | 91.06 |  | 40075 | 88.84 |  | 39746 | 88.11 |  | 40126 | 88.95 |
| 0x20D | 45109 |  | 40332 | 89.41 |  | 39448 | 87.45 |  | 39313 | 87.15 |  | 39707 | 88.02 |
| 0x287 | 45107 |  | 40749 | 90.34 |  | 39843 | 88.33 |  | 39579 | 87.74 |  | 39687 | 87.98 |
| 0x289 | 45110 |  | 41172 | 91.27 |  | 40153 | 89.01 |  | 39989 | 88.65 |  | 40015 | 88.71 |
| 0x28B | 45111 |  | 41575 | 92.16 |  | 40291 | 89.32 |  | 40535 | 89.86 |  | 40459 | 89.69 |
| 0x28D | 45109 |  | 41564 | 92.14 |  | 40757 | 90.35 |  | 40572 | 89.94 |  | 40534 | 89.86 |
| 0x307 | 45107 |  | 39491 | 87.55 |  | 38626 | 85.63 |  | 38820 | 86.06 |  | 38579 | 85.53 |
| 0x309 | 45110 |  | 36911 | 81.82 |  | 36119 | 80.07 |  | 36439 | 80.78 |  | 36505 | 80.92 |
| 0x30B | 45111 |  | 37032 | 82.09 |  | 35878 | 79.53 |  | 36347 | 80.57 |  | 36513 | 80.94 |
| 0x30D | 45109 |  | 37569 | 83.28 |  | 36807 | 81.60 |  | 36789 | 81.56 |  | 36577 | 81.09 |
| 0x385 | 45111 |  | 36647 | 81.24 |  | 33236 | 73.68 |  | 35105 | 77.82 |  | 33872 | 75.09 |
| 0x387 | 45107 |  | 39684 | 87.98 |  | 38817 | 86.06 |  | 38854 | 86.14 |  | 38597 | 85.57 |
| 0x389 | 45110 |  | 41572 | 92.16 |  | 40819 | 90.49 |  | 40338 | 89.42 |  | 40323 | 89.39 |
| 0x38B | 45111 |  | 41846 | 92.76 |  | 40896 | 90.66 |  | 40756 | 90.35 |  | 40688 | 90.20 |
| 0x38D | 45109 |  | 41464 | 91.92 |  | 40242 | 89.21 |  | 40275 | 89.28 |  | 39998 | 88.67 |
| 0x407 | 45107 |  | 39284 | 87.09 |  | 38286 | 84.88 |  | 38127 | 84.53 |  | 37984 | 84.21 |
| 0x409 | 45110 |  | 38438 | 85.21 |  | 37638 | 83.44 |  | 37394 | 82.90 |  | 37617 | 83.39 |
| 0x40B | 45111 |  | 39377 | 87.29 |  | 37911 | 84.04 |  | 38306 | 84.91 |  | 38214 | 84.71 |
| 0x40D | 45109 |  | 39898 | 88.45 |  | 39259 | 87.03 |  | 39021 | 86.50 |  | 39077 | 86.63 |
| 0x707 | 9018 |  | 8051 | 89.28 |  | 7883 | 87.41 |  | 7884 | 87.43 |  | 7885 | 87.44 |
| 0x709 | 9018 |  | 8083 | 89.63 |  | 7914 | 87.76 |  | 8000 | 88.71 |  | 7913 | 87.75 |
| 0x70B | 9018 |  | 7855 | 87.10 |  | 7638 | 84.70 |  | 7682 | 85.19 |  | 7614 | 84.43 |
| 0x70D | 9018 |  | 7959 | 88.26 |  | 7806 | 86.56 |  | 7784 | 86.32 |  | 7837 | 86.90 |

Table : Message Timing and Predicted Capture Rates for single-ID Acceptance Filter

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CAN ID | Total  logs | Total captured | Times on Target | Times before Target | Percent Captured | Times after Target | Max. Jitter (us) | Av. Period (us) | Max. Period (us) | Min. Period (us) | Offset (us) | Dif (us) |
| 0x187 | 45107 | 41249 | 0 | 41249 | 91.45 | 1 | 81726 | 41653 | 100485 | 18759 | 20644 | 7774 |
| 0x188 | 45111 | 42930 | 0 | 42930 | 95.17 | 0 | 81378 | 50328 | 100472 | 19094 | 614 | 340 |
| 0x189 | 45110 | 45057 | 0 | 45057 | 99.88 | 2 | 83770 | 36811 | 100495 | 16725 | 20918 | 274 |
| 0x18A | 45111 | 45108 | 0 | 45108 | 99.99 | 1 | 91752 | 36176 | 100498 | 8746 | 21192 | 274 |
| 0x18B | 45111 | 39027 | 0 | 39027 | 86.51 | 0 | 81458 | 50377 | 100482 | 19024 | 274 | 274 |
| 0x18C | 45111 | 43073 | 0 | 43073 | 95.48 | 0 | 81817 | 47759 | 100496 | 18679 | 21760 | 266 |
| 0x18D | 45109 | 41702 | 0 | 41702 | 92.45 | 1 | 82085 | 45313 | 100498 | 18413 | 21494 | 302 |
| 0x18E | 45111 | 43469 | 0 | 43469 | 96.36 | 1 | 82567 | 42897 | 100499 | 17932 | 22287 | 266 |
| 0x207 | 45107 | 45100 | 0 | 45100 | 99.98 | 1 | 86315 | 42095 | 100499 | 14184 | 22021 | 261 |
| 0x209 | 45110 | 44059 | 0 | 44059 | 97.67 | 1 | 82573 | 41828 | 100499 | 17926 | 22549 | 262 |
| 0x20B | 45111 | 45078 | 1 | 45077 | 99.93 | 7 | 84280 | 26136 | 100474 | 16194 | 22799 | 250 |
| 0x20D | 45109 | 45095 | 0 | 45095 | 99.97 | 13 | 92210 | 23998 | 100354 | 8144 | 3329 | 2715 |
| 0x287 | 45107 | 44863 | 28 | 44835 | 99.46 | 243 | 92192 | 22265 | 100338 | 8146 | 3589 | 260 |
| 0x289 | 45110 | 43467 | 110 | 43357 | 96.36 | 1639 | 91873 | 21045 | 100017 | 8144 | 3847 | 258 |
| 0x28B | 45111 | 43295 | 143 | 43152 | 95.97 | 1815 | 91881 | 20820 | 100009 | 8128 | 4105 | 258 |
| 0x28D | 45109 | 42838 | 403 | 42435 | 94.97 | 2270 | 73451 | 20559 | 81590 | 8139 | 4363 | 258 |
| 0x307 | 45107 | 41716 | 1469 | 40247 | 92.48 | 3390 | 73273 | 20327 | 81412 | 8139 | 4619 | 256 |
| 0x309 | 45110 | 39684 | 3193 | 36491 | 87.97 | 5420 | 52103 | 20202 | 60240 | 8137 | 4875 | 256 |
| 0x30B | 45111 | 39175 | 3279 | 35896 | 86.84 | 5935 | 52105 | 20183 | 60242 | 8137 | 5131 | 256 |
| 0x30D | 45109 | 37838 | 3430 | 34408 | 83.88 | 7270 | 52007 | 20143 | 60141 | 8134 | 5387 | 256 |
| 0x385 | 45111 | 43264 | 0 | 43264 | 95.91 | 0 | 81275 | 54450 | 100490 | 19215 | 140248 | 112880 |
| 0x387 | 45107 | 43565 | 146 | 43419 | 96.58 | 1541 | 92067 | 20887 | 100205 | 8138 | 5645 | 258 |
| 0x389 | 45110 | 44571 | 430 | 44141 | 98.81 | 533 | 92065 | 20887 | 100223 | 8158 | 5903 | 258 |
| 0x38B | 45111 | 44838 | 74 | 44764 | 99.39 | 272 | 92069 | 20880 | 100223 | 8154 | 6169 | 266 |
| 0x38D | 45109 | 44898 | 48 | 44850 | 99.53 | 210 | 92072 | 20863 | 100228 | 8156 | 6437 | 268 |
| 0x407 | 45107 | 44906 | 47 | 44859 | 99.55 | 200 | 91923 | 20841 | 100085 | 8162 | 6863 | 426 |
| 0x409 | 45110 | 35572 | 14953 | 20619 | 78.86 | 9535 | 71987 | 20046 | 80155 | 8168 | 7117 | 254 |
| 0x40B | 45111 | 35424 | 16168 | 19256 | 78.53 | 9685 | 51956 | 20041 | 60126 | 8170 | 7368 | 251 |
| 0x40D | 45109 | 35227 | 17445 | 17782 | 78.09 | 9881 | 51965 | 20035 | 60136 | 8171 | 7616 | 248 |
| 0x707 | 9018 | 1687 | 449 | 1238 | 18.71 | 6494 | 2703 | 99901 | 100498 | 97795 | 12870 | 142 |
| 0x709 | 9018 | 1433 | 304 | 1129 | 15.89 | 6847 | 19855 | 99878 | 100499 | 80644 | 9720 | 2104 |
| 0x70B | 9018 | 2373 | 1030 | 1343 | 26.31 | 5923 | 3267 | 99927 | 100495 | 97228 | 12728 | 3008 |
| 0x70D | 9018 | 5198 | 3798 | 1400 | 57.64 | 3123 | 16252 | 99953 | 100498 | 84246 | 27368 | 4569 |

Table : Logging Analysis Results for Varying logging Task Period

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CAN ID | Total |  | 500 us | | |  | 1000 us | | |  | 1500 us | | |  | 2000 us | | |
|  |  |  | Logged | Missed | Percent |  | Logged | Missed | Percent |  | Logged | Missed | Percent |  | Logged | Missed | Percent |
| 0x187 | 45107 |  | 45101 | 6 | 99.987% |  | 45057 | 50 | 99.889% |  | 44870 | 237 | 99.475% |  | 44243 | 864 | 98.085% |
| 0x188 | 45111 |  | 45107 | 4 | 99.991% |  | 44341 | 770 | 98.293% |  | 44385 | 726 | 98.391% |  | 44236 | 875 | 98.060% |
| 0x189 | 45110 |  | 45075 | 35 | 99.922% |  | 45055 | 55 | 99.878% |  | 44935 | 175 | 99.612% |  | 44393 | 717 | 98.411% |
| 0x18A | 45111 |  | 45107 | 4 | 99.991% |  | 44780 | 331 | 99.266% |  | 44649 | 462 | 98.976% |  | 44238 | 873 | 98.065% |
| 0x18B | 45111 |  | 45067 | 44 | 99.902% |  | 45047 | 64 | 99.858% |  | 44925 | 186 | 99.588% |  | 44352 | 759 | 98.317% |
| 0x18C | 45111 |  | 45107 | 4 | 99.991% |  | 44307 | 804 | 98.218% |  | 44003 | 1108 | 97.544% |  | 44061 | 1050 | 97.672% |
| 0x18D | 45109 |  | 45077 | 32 | 99.929% |  | 45054 | 55 | 99.878% |  | 44938 | 171 | 99.621% |  | 44372 | 737 | 98.366% |
| 0x18E | 45111 |  | 45107 | 4 | 99.991% |  | 44863 | 248 | 99.450% |  | 44566 | 545 | 98.792% |  | 44071 | 1040 | 97.695% |
| 0x207 | 45107 |  | 45101 | 6 | 99.987% |  | 45045 | 62 | 99.863% |  | 44838 | 269 | 99.404% |  | 44202 | 905 | 97.994% |
| 0x209 | 45110 |  | 45069 | 41 | 99.909% |  | 45059 | 51 | 99.887% |  | 44929 | 181 | 99.599% |  | 44383 | 727 | 98.388% |
| 0x20B | 45111 |  | 45098 | 13 | 99.971% |  | 44826 | 285 | 99.368% |  | 44406 | 705 | 98.437% |  | 43442 | 1669 | 96.300% |
| 0x20D | 45109 |  | 44817 | 292 | 99.353% |  | 44453 | 656 | 98.546% |  | 44225 | 884 | 98.040% |  | 43331 | 1778 | 96.058% |
| 0x287 | 45107 |  | 45100 | 7 | 99.984% |  | 44958 | 149 | 99.670% |  | 44835 | 272 | 99.397% |  | 44238 | 869 | 98.073% |
| 0x289 | 45110 |  | 45056 | 54 | 99.880% |  | 44966 | 144 | 99.681% |  | 44749 | 361 | 99.200% |  | 44216 | 894 | 98.018% |
| 0x28B | 45111 |  | 45016 | 95 | 99.789% |  | 44942 | 169 | 99.625% |  | 44794 | 317 | 99.297% |  | 44082 | 1029 | 97.719% |
| 0x28D | 45109 |  | 44936 | 173 | 99.616% |  | 44862 | 247 | 99.452% |  | 44740 | 369 | 99.182% |  | 43961 | 1148 | 97.455% |
| 0x307 | 45107 |  | 45091 | 16 | 99.965% |  | 45042 | 65 | 99.856% |  | 44915 | 192 | 99.574% |  | 44111 | 996 | 97.792% |
| 0x309 | 45110 |  | 45090 | 20 | 99.956% |  | 45032 | 78 | 99.827% |  | 44819 | 291 | 99.355% |  | 44165 | 945 | 97.905% |
| 0x30B | 45111 |  | 45087 | 24 | 99.947% |  | 45048 | 63 | 99.860% |  | 44752 | 359 | 99.204% |  | 44269 | 842 | 98.133% |
| 0x30D | 45109 |  | 45084 | 25 | 99.945% |  | 45034 | 75 | 99.834% |  | 44804 | 305 | 99.324% |  | 44265 | 844 | 98.129% |
| 0x385 | 45111 |  | 45107 | 4 | 99.991% |  | 44883 | 228 | 99.495% |  | 44166 | 945 | 97.905% |  | 44143 | 968 | 97.854% |
| 0x387 | 45107 |  | 45103 | 4 | 99.991% |  | 45077 | 30 | 99.933% |  | 44946 | 161 | 99.643% |  | 44188 | 919 | 97.963% |
| 0x389 | 45110 |  | 45101 | 9 | 99.980% |  | 45077 | 33 | 99.927% |  | 44877 | 233 | 99.483% |  | 44154 | 956 | 97.881% |
| 0x38B | 45111 |  | 45106 | 5 | 99.989% |  | 45084 | 27 | 99.940% |  | 44880 | 231 | 99.488% |  | 44280 | 831 | 98.158% |
| 0x38D | 45109 |  | 45106 | 3 | 99.993% |  | 45083 | 26 | 99.942% |  | 44960 | 149 | 99.670% |  | 44346 | 763 | 98.309% |
| 0x407 | 45107 |  | 45101 | 6 | 99.987% |  | 45078 | 29 | 99.936% |  | 44951 | 156 | 99.654% |  | 44344 | 763 | 98.308% |
| 0x409 | 45110 |  | 45100 | 10 | 99.978% |  | 45077 | 33 | 99.927% |  | 44954 | 156 | 99.654% |  | 44288 | 822 | 98.178% |
| 0x40B | 45111 |  | 45106 | 5 | 99.989% |  | 45083 | 28 | 99.938% |  | 44954 | 157 | 99.652% |  | 44276 | 835 | 98.149% |
| 0x40D | 45109 |  | 45106 | 3 | 99.993% |  | 45079 | 30 | 99.933% |  | 44959 | 150 | 99.667% |  | 44361 | 748 | 98.342% |
| 0x707 | 9018 |  | 9018 | 0 | 100.000% |  | 9018 | 0 | 100.000% |  | 9018 | 0 | 100.000% |  | 9018 | 0 | 100.000% |
| 0x709 | 9018 |  | 9018 | 0 | 100.000% |  | 9018 | 0 | 100.000% |  | 9018 | 0 | 100.000% |  | 9018 | 0 | 100.000% |
| 0x70B | 9018 |  | 9018 | 0 | 100.000% |  | 9018 | 0 | 100.000% |  | 9018 | 0 | 100.000% |  | 9018 | 0 | 100.000% |
| 0x70D | 9018 |  | 9018 | 0 | 100.000% |  | 9018 | 0 | 100.000% |  | 9018 | 0 | 100.000% |  | 9018 | 0 | 100.000% |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Total | 1344249 |  | 1343301 | 948 | 99.929% |  | 1339364 | 4885 | 99.637% |  | 1333796 | 10453 | 99.222% |  | 1317083 | 27166 | 97.979% |

Figure : Capture Rate Comparisson for ID's 0x18n

Figure : CAPTURE RATE COMPARISSON FOR ID'S 0x2nn

Figure : Capture Rate Comparisson for ID's 0X3nN

Figure : Capture Rate Comparisson for ID's 0x40n & 0x70n

# Source Code

Below is the source code for the main analysis portion of the script:

**#define** BUFFERSIZE (81)

**#define** FILTERSIZE (35)

**#define** MAX\_TRACE\_LINES (0) /\* Set to zero to analyse entire trace \*/

**#define** LOGGING\_TASK\_PERIOD\_us (1000)

**#define** FREEZE\_TRIES (0)

**#define** MESSAGE\_TIME\_DELTA\_MAX (100500)

**#define** MESSAGE\_TIME\_DELTA\_MIN (0)

**unsigned** **long** ID, timeDelta;

**int** noIDs;

**unsigned** **int** filterPointer;

**typedef** **struct**

{

**int** canID;

**unsigned** **long** counter;

**unsigned** **long** loggedCounter;

} logging\_Sequence\_t;

**typedef** **enum** {*TRUE*, *FALSE*}flag\_t;

**typedef** **struct**

{

**int** canID;

**int** sequencePointer;

flag\_t loggedFlag;

} filter\_t;

logging\_Sequence\_t loggingSequence[BUFFERSIZE];

filter\_t acceptanceFilter[FILTERSIZE];

**void** **getSimpleCanSequence**(**char** \*filename, FILE \*log);

**void** **checkLogability**(**char** \*filename, FILE \*log, **int** filterSize, **int** sequenceSize);

**void** **orderSequence**(**void**);

**int** **countSequence**(**void**);

**int** **main**(**void**)

{

**char** \*CANlogFile = "Log\_for\_analysis2.asc";

**int** i, sequenceSize;

FILE \*outputFile = **fopen**("output.txt", "w");

FILE \*logFile = **fopen**("CAN\_Logging\_Simple\_29-01-2013\_Timing\_1000us.csv", "w");

noIDs = 0;

**/\* source for getSimpleCanSequence() and countSequence() omitted due to simplicity – available on request \*/**

getSimpleCanSequence(CANlogFile, logFile);

sequenceSize = countSequence();

**printf**("\n\n\n");

**printf**("\n\n %u ID's",sequenceSize);

**printf**("\n\n\nChecking logability...\r\n\n");

**fprintf**(logFile,"\n\n,,Filter Size,Logged,Missed\n");

**for**(i = 1; i <= (sequenceSize-1); i++)

{

checkLogability(CANlogFile, logFile, i, sequenceSize);

}

**fclose**(outputFile);

**fclose**(logFile);

**return** EXIT\_SUCCESS;

}

**void** **checkLogability**(**char** \*filename, FILE \*log, **int** filterSize, **int** sequenceSize)

{

**char** inputStr[200];

**char** canData[200];

**int** i = 0, j = 0, sequencePointer = 0, sequencePointerStart = 0, IDLogCount = 0, IDMissedCount = 0;

flag\_t IDlogged = *FALSE*, IDfound = *FALSE*;

**unsigned** **long** timeNow\_s ,timeNow\_us, timeOrigin, lineCounter;

**int** ID;

/\* open trace file \*/

FILE \*bufferFile = **fopen**(filename, "r");

/\* Put first filterSize CAN ID's from sequence into acceptanceFilter \*/

**for**(i = 0; i < filterSize; i++)

{

acceptanceFilter[i].canID = loggingSequence[i].canID;

acceptanceFilter[i].sequencePointer = i;

acceptanceFilter[i].loggedFlag = *FALSE*;

sequencePointer = i;

}

timeOrigin = 0;

/\* loop through trace lines in file for MAX\_TRACE\_LINES \*/

**while**((**fgets**(inputStr, 190, bufferFile) != NULL) && ((lineCounter++ < MAX\_TRACE\_LINES) || (MAX\_TRACE\_LINES == 0))) {

/\* Extract values from input string \*/

**unsigned** **int** scanReturn = **sscanf**(inputStr, logFormat, &timeNow\_s, &timeNow\_us, &ID, &canData);

**if**(scanReturn == 4) /\* valid line in trace \*/

{

/\* Time calculation from seconds and microseconds \*/

timeNow\_us += (timeNow\_s \* 1000000);

/\* Set time origin on first run \*/

**if**(timeOrigin == 0)

{

timeOrigin = timeNow\_us;

}

/\* Find current time delta from origin \*/

timeDelta = (timeNow\_us - timeOrigin);

/\* Output so I can see it doing something \*/

**printf**("%u %lu\n", filterSize, timeNow\_us);

/\* Logging task has run - replace logged ID’s in filter \*/

**if**(timeDelta >= LOGGING\_TASK\_PERIOD\_us)

{

/\* Iterate through whole acceptance filter \*/

**for**(i = 0; i < filterSize; i++)

{

/\* loggedFlag is set when ID is logged for first time \*/

**if**(acceptanceFilter[i].loggedFlag == *TRUE*)

{

/\* look through logging sequence for next ID not contained in acceptance filter \*/

sequencePointerStart = sequencePointer;

**do**

{

sequencePointer++;

**if**(sequencePointer >= sequenceSize)

{

sequencePointer = 0;

}

IDfound = *FALSE*;

**for**(j = 0; j < filterSize; j++)

{

**if**(acceptanceFilter[j].canID == loggingSequence[sequencePointer].canID)

{

IDfound = *TRUE*;

}

}

}**while**((sequencePointer != sequencePointerStart) && (IDfound == *TRUE*));

/\* ID found not already in acceptance filter - replace Id in filter \*/

**if**(IDfound == *FALSE*)

{

acceptanceFilter[i].canID = loggingSequence[sequencePointer].canID;

acceptanceFilter[i].sequencePointer = sequencePointer;

acceptanceFilter[i].loggedFlag = *FALSE*;

}

}

}

/\* Reset time origin to current time \*/

timeOrigin = timeNow\_us;

}

/\* ID is in logging list \*/

**if**(GetCAN1BufferPointer(ID) == 1)

{

IDlogged = *FALSE*;

i = 0;

/\* look for ID in acceptance filter \*/

**do**

{

**if**(acceptanceFilter[i].canID == ID)

{

/\* ID found, increment counters \*/

IDLogCount++;

loggingSequence[acceptanceFilter[i].sequencePointer].loggedCounter++;

acceptanceFilter[i].loggedFlag = *TRUE*;

IDlogged = *TRUE*;

}

i++;

}**while**((IDlogged == *FALSE*) && (i < filterSize));

/\* ID not found in filter, so would be missed \*/

**if**(IDlogged == *FALSE*)

{

IDMissedCount++;

}

}

}

}

**printf**("filterSize: %u Logged: %u Missed %u\n", filterSize, IDLogCount, IDMissedCount);

/\* Output code commented out when not in use TODO: make these separate functions \*/

/\* Use below to output individual ID counts \*/

// for(i = 0; i < BUFFERSIZE; i++)

// {

// if(loggingSequence[i].canID != 0)

// {

// fprintf(log,",,0x%03X,%lu,%lu,%lu\n",loggingSequence[i].canID, loggingSequence[i].loggedCounter, (loggingSequence[i].counter - loggingSequence[i].loggedCounter), loggingSequence[i].counter);

// }

// }

// fprintf(log,"\n");

/\* Use below to output overall, per filterSize, hit / miss counts \*/

**fprintf**(log,",,%u,%u,%u\n", filterSize, IDLogCount, IDMissedCount);

}