Checking Values of CAN Signals against Default Requirements

2022-05-13
Support Note SN-HND-1-045

Author(s) Rao, Srinidhi
Restrictions Public Document

Table of Contents

1 Overview

This Support Note shows how values of CAN signals arriving first-time after an ECU power-up can be checked against the database specification.

This is common to be done as a test to verify whether the ECU complies with CAN initialization requirements. In this context, this has to be ensured for thousands of signals transmitted by the ECU & it is even more complex in gateway ECUs.

The direct approach is to manually access signals (using `$`) and compare them with respective default values (from a list of declared constants). The challenge here is to compare the default value (as defined in the DBC file) with received value on CAN bus for all signals one-by-one which is tedious.

With no programming construct in CAPL to traverse through the whole list of CAN signals, the following two solutions deal with the signals and the respective default values trickily. The following sections discuss the high-level steps on both the approaches.

2 Different possible approaches

There are two different approaches described in this document:

> Approach 1: Call-in signals by DBC look-up dynamically

> Approach 2: Auto-generate a CAPL code that calls-in signals statically

The choice of solution can be reckoned through the below trade-offs:

- Approach 2 is more performant as the run-time environment is faster, because the signal objects are determined during CAPL code compilation before measurement start. Approach 1 determines the signal objects during run-time as these selectors search the DB, and can affect the performance with larger databases
- Approach 1 is still fair enough despite the performance is influenced, as it runs in test module that does not affect measurement. This is suitable when the number of signals is less than 200 roughly (small to medium DBCs)
- Approach 2 may lack elegance in code-readability as it is auto-generated and flexibility to print-out detailed test reports/logs goes with complexity of code that generates CAPL code

Both approaches come with examples that have been archived in a ZIP archive in the following files:

`Comfort.dbc`	Example DBC used in this article
`ExportOfSignalsList.txt`	Export of signal names as per the DBC
`ExportOfSignalValues.txt`	Export of signal values respective to the signals
`TEST1_CAPL_Test_Module.can`	CAPL Test Module as per approach 1
`TEST2_CAPL_Test_Module.can`	Auto-generated CAPL test module as per approach 2
`Result_TEST1_CAPL_Test_Module.txt`	Result log with respect to `TEST1_CAPL_Test_Module.can`
`Result_TEST2_CAPL_Test_Module.txt`	Result log with respect to `TEST2_CAPL_Test_Module.can`
`CAPL_Code_Generator.can`	CAPL code that generates CAPL test module (i.e. `TEST2_CAPL_Test_Module.can`) as per approach 2

3 Approach 1: Call-in signals by DBC look-up dynamically

Step by Step

Logical steps to be taken

- Step 1: Export the list of signals from DBC to a CSV text file (ExportOfSignalsList.txt) as contained in the ZIP archive (**DBC Editor | View | List | Signals & File | Export**)
- Step 2: Read each line as a string and convert it to a signal-object accessible in CANoe
- Step 3: Look-up on the DBC for the default value of a signal using 'DBC Signal Selectors'
- Step 4: Compare the physical value of DBC data with the physical value of the signal (or raw vs. raw)
- Step 5: Decide the pass/fail status based on the result of the comparison

3.1 Example for Approach 1

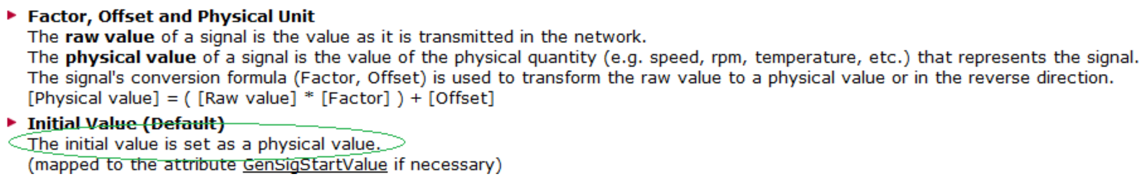
Example

The below code also depicts multiple forms of comparison for additional reference.

3.1.1 TEST1_CAPL_Test_Module.can

```
 dword phys_val(signal * s)
{
   return $s.phys;
}

dword raw_val(signal * s)
{
   return $s.raw;
}

testcase TEST_001()
{
  dword index;

  int var_DBLookupDefaultValue;
  int var_DBLookupGenSigStartValue;
  int var_getSignal;
  int var_phys_val;
  int var_raw_val;

  glbHandle = OpenFileRead("D:\\ExportOfSignalsList.txt", 0);

  while(fileGetStringSZ(parsedSignal, elcount(parsedSignal), glbHandle))
  {
    // In some cases, 'DBLookup().DefaultValue' could be mapped to
'GenSigStartValue' (see footnote 1), if necessary. If so, convert it to
physical value
```

1



- **Factor, Offset and Physical Unit**
  The **raw value** of a signal is the value as it is transmitted in the network.
  The **physical value** of a signal is the value of the physical quantity (e.g. speed, rpm, temperature, etc.) that represents the signal.
  The signal's conversion formula (Factor, Offset) is used to transform the raw value to a physical value or in the reverse direction.
  [Physical value] = ([Raw value] * [Factor]) + [Offset]
- **Initial Value (Default)**
  The initial value is set as a physical value.
  (mapped to the attribute GenSigStartValue if necessary)

```
    // This could be verified by comparing if DBLookup().DefaultValue
returns a zero value
    if ((DBLookup(lookupSignal(parsedSignal)).DefaultValue == 0) &&
(DBLookup(lookupSignal(parsedSignal)).GenSigStartValue == 0))
    {
      // phy = (raw x factor) + offset
      var_DBLookupDefaultValue =
(DBLookup(lookupSignal(parsedSignal)).DefaultValue *
DBLookup(lookupSignal(parsedSignal)).factor) +
DBLookup(lookupSignal(parsedSignal)).offset;
    }
    else
    {
      var_DBLookupDefaultValue =
DBLookup(lookupSignal(parsedSignal)).DefaultValue;
    }

    var_DBLookupGenSigStartValue =
DBLookup(lookupSignal(parsedSignal)).GenSigStartValue;
    var_getSignal = getSignal(parsedSignal);
    var_raw_val = raw_val(lookupSignal(parsedSignal));
    var_phys_val = phys_val(lookupSignal(parsedSignal));

    write("===========================================================
=========================================");
    write("Default/Init/Received Values of '%s':", parsedSignal);
    write("DB value (i.e. physical value) is %d",
var_DBLookupDefaultValue);
    write("DB value (i.e. init value) is %d",
var_DBLookupGenSigStartValue);
    write("Received value is %d (phy: form 1)", var_getSignal);
    write("Received value is %d (phy: form 2)", var_phys_val);
    write("Received value is %d (raw: form 2)", var_raw_val);

    write("===========================================================
=========================================");

    if (TestWaitForSignal(lookupSignal(parsedSignal), 2000) == 1) // to get
rid of false-positive as 'getSignal' returns default if signal is
unavailable
    {
      if(DBLookup(lookupSignal(parsedSignal)) == 1) // if the parsed string
is indeed a signal
      {
        if(var_DBLookupDefaultValue == var_getSignal)
        {
          write("      > PASS: DBC default value & bus-received value of
CAN signal '%s' match\n\n", parsedSignal);
        }
        else
        {
          write("      > FAIL: DBC default value & bus-received value of
CAN signal '%s' do not match\n\n", parsedSignal);
        }
      }
    }
  }
}

// Other possible forms for comparison

// if(DBLookup(lookupSignal(parsedSignal)).GenSigStartValue ==
getSignal(parsedSignal)) --> Raw versus Raw
```

```
        // if(DBLookup(lookupSignal(parsedSignal)).DefaultValue ==
phys_val(lookupSignal(parsedSignal))) --> physical versus physical
        // if((DBLookup(lookupSignal(parsedSignal)).DefaultValue -
DBLookup(lookupSignal(parsedSignal)).offset)/DBLookup(lookupSignal(parsedSi
gnal)).factor == getSignal(parsedSignal))
         // --> Raw versus Raw (converted)
        // if((DBLookup(lookupSignal(parsedSignal)).DefaultValue -
DBLookup(lookupSignal(parsedSignal)).offset)/DBLookup(lookupSignal(parsedSi
gnal)).factor == raw_val(lookupSignal(parsedSignal))         // --> Raw
versus Raw (converted)
```

4 Approach 2: Auto-generate a CAPL code that calls-in signals statically

Step by Step

Logical steps to be taken

- > Step 1: Export the list of signals and corresponding default values from DBC to a CSV text file (`ExportOfSignalsList.txt` & `ExportOfSignalValues.txt`) as contained in the ZIP archive (**DBC Editor | View | List | Signals & File | Export**)
- > Step 2: Read each line from both the files hand-in-hand and write them to a file as is in the format of a CAPL Test Module (CAPL Test Header <> Signal access ($) <> Compare & Report P/F)
- > Step 3: Attach the auto-generated CAPL Test Module to a CAPL Test Node
- > Step 4: Resolve the signal ambiguities, if any (like signal names are not unique or an invalid signal read out of the string)
- > Step 5: Compile & run the CAPL Test Module

In all, this approach flattens the CAPL Test Module that loops and searches DBC on the list of signals, (as in chapter 3) into separate entities accessible statically → 'Loop unrolling'.

4.1 Example for Approach 2

Example

4.1.1 TEST2_CAPL_Test_Module.can

Only a few signals are displayed. Refer to `TEST2_CAPL_Test_Module.can` for full list of signals

```
testcase TEST_002()
{

  if($Active == 0)
  {
     write("> PASS: DBC default value & bus-received value of CAN signal
'Active' match\n");
  }
  else
  {
     write("> FAIL: DBC default value & bus-received value of CAN signal
'Active' do not match\n");
  }
  if($CarSpeed == 0)
  {
     write("> PASS: DBC default value & bus-received value of CAN signal
'CarSpeed' match\n");
  }
  else
  {
     write("> FAIL: DBC default value & bus-received value of CAN signal
'CarSpeed' do not match\n");
  }

  if($WN_right_up == 0)
  {
     write("> PASS: DBC default value & bus-received value of CAN signal
'WN_right_up' match\n");
  }
  else
  {
     write("> FAIL: DBC default value & bus-received value of CAN signal
'WN_right_up' do not match\n");
  }

}
```

4.1.2 CAPL_Code_Generator.can

```
variables
{
  int signalCount = 0;
  char parsedSignalName[1000][100];
  char parsedSignalValue[1000][5];

  char temp[1000];
  int index;

  dword glbHandle = 0;
  dword glbHandle2 = 0;
  dword writeHandle = 0;

  char CAPLtestNodeStarter1[100] = "void MainTest()\n{\n\n";
  char CAPLtestNodeStarter2[100] = "
TestModuleTitle(\"TEST2_CAPL_Test_Module\");\n";
  char CAPLtestNodeStarter3[100] = "
TestModuleDescription(\"TEST2_CAPL_Test_Module\");\n";
  char CAPLtestNodeStarter4[100] = "  TestGroupBegin(\"TEST_GROUP2\",
\"TEST2_CAPL_Test_Module\");\n\n";
  char CAPLtestNodeStarter5[100] = "  TEST_002();\n\n";
  char CAPLtestNodeStarter6[100] = "  TestGroupEnd();\n\n}\n\n\n";

  char CAPLtestHeader[100] = "testcase TEST_002()\n{\n\n";
}

on start
{
  setWritePath("D:");
  writeHandle = openFileWrite("TEST2_CAPL_Test_Module.can", 1);

  glbHandle = OpenFileRead("D:\\ExportOfSignalsList.txt", 0);
  glbHandle2 = OpenFileRead("D:\\ExportOfSignalValues.txt", 0);

  signalCount = 0;
  while(fileGetStringSZ(temp, elcount(temp), glbHandle))
  {
    snprintf(parsedSignalName[signalCount], 100, temp); // Compile the name
of signals
    signalCount++;
  }

  signalCount = 0;
  while(fileGetStringSZ(temp, elcount(temp), glbHandle2))
  {
    snprintf(parsedSignalValue[signalCount], 5, temp);  // Compile the
value of signals
    signalCount++; // Find out the number of signals
  }
}

on key '9'
{
  snprintf(temp, elcount(temp), "%s%s%s%s%s%s%s", CAPLtestNodeStarter1,
CAPLtestNodeStarter2, CAPLtestNodeStarter3, CAPLtestNodeStarter4,
CAPLtestNodeStarter5, CAPLtestNodeStarter6, CAPLtestHeader);

  filePutString(temp, elcount(temp), writeHandle); // CAPL testcase format

  for(index = 0; index < signalCount; index++)
  {
    snprintf(temp, elcount(temp), "");
    snprintf(temp, elcount(temp), "\n");
    snprintf(temp, elcount(temp), "  if($%s == %s)\n  {\n     write(\">
PASS: DBC default value & bus-received value of CAN signal '%s'
match\\n\");\n  }\n", parsedSignalName[index], parsedSignalValue[index],
parsedSignalName[index]);
    filePutString(temp, elcount(temp), writeHandle);

    snprintf(temp, elcount(temp), "");
    snprintf(temp, elcount(temp), "  else\n  {\n     write(\"> FAIL: DBC
default value & bus-received value of CAN signal '%s' do not match\\n\");\n
}\n\n", parsedSignalName[index], parsedSignalValue[index],
parsedSignalName[index]);
    filePutString(temp, elcount(temp), writeHandle);
  }

    snprintf(temp, elcount(temp), "");
    snprintf(temp, elcount(temp), "}");
    filePutString(temp, elcount(temp), writeHandle);

}
```

5 Important notes to both approaches

Note
Please read carefully before using example implementation

> The time-out of 2000 ms is arbitrarily chosen, it is recommended to use respective cycle time of signal/message

> This code is to be executed immediately after power-up before any CAN signal is modified (or else respective CAN signal will fail)

- In approach 2, care shall be taken in cases where `DBLookup().DefaultValue` could be mapped to `GenSigStartValue` unlike it is handled in Approach 1 (refer 1)
- Example code to auto-generate the CAPL test module is written to demonstrate approach 2 and user shall refine to suit it to own requirements
- CAPL code that generates a CAPL test module shall be run in a “dummy CANoe simulation” and further shall be associated with “target CANoe simulation”. **CAPL is only used as an example here. You may use any programming or scripting environment, e.g. .NET, Python or VB**. `CAPL_Code_Generator.can` exemplifies the code design and could be referred to as a blueprint.

6 Associated files

A ZIP archive with the code example files you can download from the connected KnowledgeBase article (KB0012440).

7 Contacts

For support related questions please address to the support contact for your country https://www.vector.com/int/en/company/contacts/support-contact/.