HYBRID COMMERCIAL VEHICLE
DELIVERABLE D6200.4

IMPLEMENTATION OF EXTENDED COMMUNICATION PROTOCOL IN THE SIMULATION PLATFORM
1. Summary

This report (D6200.4) Implementation of extended communication protocol in the simulation platform, gives the detailed overview of the procedure for implementation of communication protocol of hybrid powertrains defined in WP6320 into the high fidelity simulation environment AVL InMotion. The primary goal of this task is to integrate and validation of communication protocol on simulations and test beds.
## 2. Table of contents

1. Summary .................................................................................. 2
2. Table of contents ........................................................................ 3
3. Introduction ................................................................................ 4
4. Technical description .................................................................. 5
   4.1 Database file ......................................................................... 5
       4.1.1 Creating a new database file ........................................... 6
       4.1.2 Creating and modifying objects ...................................... 7
   4.2 MIO — M-module Input/output ............................................. 9
       4.2.1 Programming M-module I/O .......................................... 9
       4.2.2 Initialization and configuration ...................................... 10
       4.2.3 M51: Quadruple CAN interface .................................. 13
   4.3 CANiogen-CANdb import tool ............................................ 15
       4.3.1 Using CANiogen ......................................................... 18
       4.3.2 Importing Electronic Control Units (ECUs) .................. 18
   4.4 Integration into AVL InMotion/HIL ..................................... 20
       4.4.1 Adaptation of Makefile ................................................ 20
       4.4.2 Adaptation of IO.c ..................................................... 22
       4.4.3 Adaptation of User.c .................................................. 24
5. Results and discussion ................................................................ 25
6. Conclusion ................................................................................. 39
3. Introduction

The Hybrid Commercial Vehicle (HCV) project aims to develop urban buses and delivery vehicles with advanced second generation of energy efficient hybrid electric power train. The final result will be the demonstration of a passenger bus and a delivery truck with this advanced technology.

The WP6200 aims to deliver the secure testing standards for high voltage components, to develop the environment for secure exchange of simulation models on different simulation systems, integration and validation of communication protocols on simulators and test beds and finally to develop a new methodology and tools for consistent use of virtual drive cycle for hybrid commercial vehicles in all development stages (design, integration and testing).

The particular task 6230 (implementation of extended communication protocol), the standard communication protocol for hybrid powertrains developed in WP6320 is integrated into the simulation platform. The following actions were scheduled and successfully implemented.

- Creating a database file based on the D6300.2 (communication protocol for 2nd generation HD hybrids).
- Parameterizing the MIO module. In this task, we used M51 module for developing the CAN interface with the simulation platform.
- Using the CANiogen, a database import tool used to import the control units into the simulation platform.
- Finally with the required modifications, integrating into the AVL InMotion/HIL.
4. Technical description

The aim of the deliverable D6200.4 is to implement the communication protocol defined in the WP 6320. The project partner AVL received the report D6300.2 in a text version for the parameters of the communication database.

4.1 Database file

The primary step in the process of implementing the communication protocol into the simulation environment is to create a dbc file, which is then integrated into the simulation environment AVL InMotion. To create the dbc file, we used a database editor named CANdb++. All communication-relevant data that are processed in a networked CAN bus system as well as their interrelationships are usually administered in a central communications database. The figure 4.1 shows the CANdb++ as a central tool for managing communication data.

![Diagram of CANdb++ as a central tool for managing communication data.

Figure 4.1: Central tool for managing communication data.
4.1.1 Creating a new database file
Within CANdb++, we can set up CAN databases of the following types:

- **CANdb++ databases** (*.mdc)
- **CANdb network file** (*.dbc)

The below mentioned procedure is followed to create a new database file.

![Overview of CANdb++ editor](image)

**Figure 4.2: Overview of CANdb++ editor**

1. Choose the **File|Create Database** command. First, the **Template** dialog is opened, as shown in figure 4.2.

![Selection of database template](image)

**Figure 4.3: Selection of database template**

2. There we can choose a template for a **CANdb network** (.dbc) or for a **CANdb++ database** (.mdc).
3. Next we can select one of the available templates as shown in figure 4.3, after selection of the required template, the **New Database File** dialog is opened, in which the memory location, file name and file type can be defined for the CAN database to be created.

![Figure 4.4: Overview of the database editor](image)

**4.1.2 Creating and modifying objects**

The objects that can be created in the CANdb++ editor are vehicles (coupe), networks (body, powertrain), control units (combi, driver control, ECU_motor), environment variables, network nodes (body_gateway, display, motor), messages (driver info, engine control), signals (display temp, Gear select).

To create a new object, the procedure followed is given below:

- Select the object type as shown in figure 4.4, and then choose the **Edit|New** command. This opens a new window as shown in the figure 4.5.

![Figure 4.5: defining a new object in the database window.](image)
• Change the values of system parameters in the object dialog and press the [OK] button. The newly created object appears in the Overview window. Appearing in the table columns on the right side of the Overview Window are the values of the object’s system parameters.

Now based up on the deliverable D6300.2 (Communication protocol for 2nd generation HD Hybrids), by the project partner TNO, a new dbc file is created which will be implemented in to the simulation platform AVL InMotion.

**SPN X+1  ESS Momentary max. Discharge Current**
Comment: Maximum value for the ESS current at which the ESS can be discharged at current moment.
SLOT Name: SAEec01
Data Length: 2 bytes
Resolution: 0.05 A/bit, 0 A offset
Data range: -1600 to 1612.75 A  Operational range 0 to 1612.75 A
Type: Status
PGN reference: 
Refresh rate: 100 ms

**SPN X+2  ESS Actual Voltage**
SLOT Name: SAEev05
Data Length: 2 bytes
Resolution: 0.05 V/bit, 0 V offset
Data range: 0 to 3212.75 V
Type: Measured
PGN reference: 
Refresh rate: 100 ms

**SPN X+3  ESS State Of Charge (SOC)**
Comment: SOC is given as a percentage of the actual electric capacity. This capacity may change over time, so absolute available amount of energy in ESS cannot be determined by SOC alone.
SLOT Name: SAEpc03
Data Length: 1 byte
Resolution: 0.4 %/bit, 0 offset
Data range: 0 to 100%
Type: Status
PGN reference: 
Refresh rate: 5 s

**Figure 4.6: Part of communication protocol in the deliverable D6300.2**
By the above given procedure of creating a new database file, a dbc file is created with reference to the D6300.2. The figure 4.7 shows the created dbc file that is implemented into the simulation platform.

Figure 4.7: dbc file created with reference to the deliverable D6300.2

4.2 MIO—M-module Input/output

4.2.1 Programming M-module I/O

The initialization process for the MIO software module depends on the hardware configuration. First, we have to initialize the MIO software module in general. Then, depending on the installed M-Module hardware, we initialize the installed M-Modules by calling the appropriate MIO_Mxx_Config() functions. A flow chart of the initialization process is shown in figure 4.8.

To simplify this process, the MIO software module logs errors to the session log and informs about with negative return values to MIO initialization functions. However, since the AVL InMotion Log module allows to check the error counter, it is not necessary to check every single call to MIO_xx() functions.
4.2.2 Initialization and configuration

The first step to initialize the MIO hardware is the initialization of the MIO software module in general, by calling the function `MIO_Init()`. This function initializes the carrier boards and reads the module type of all installed M-Modules.

Now, we have to configure the installed M-Modules, each with the appropriate `MIO_Mxx_Config()` function. Each `MIO_Mxx_Config()` function requires at least one argument, the slot number of the M-Module.

When registering a module, the MIO software module checks, if there is a M-Module installed at the given slot, and if the type of the installed module matches the selected `MIO_Mxx_Config()` function. If a mismatch is detected, an error is logged to the session log and a negative value is returned. However, in some circumstances a module mismatch
cannot be ruled out completely: some older M-Modules do not have an Id EEPROM, so the MIO software module cannot determine the type of installed module and has to accept the module type, passed by the MIO_Mxx_Config() function.

The slot numbers of the installed M-Modules depend on the hardware configuration, i.e. on the order in which the BIOS and the XENO kernel recognize and initialize PCI devices. After all M-Modules have been registered, and prior to any access to the modules, we have to check again if there was an error during the MIO hardware initialization. If we do not handle errors at this point of time, the application might crash later on when accessing the MIO hardware, because of bus errors or segmentation violations. The following figure 4.9, shows the MIO initialization of typical AVL InMotion/HIL MIO hardware configuration.

```c
int IO_Init (void)
{
    Log("/O Configuration: %s\n", IO_ListNames(NULL, 1));

    /* hardware configuration "none" */
    if (IO_None)
        return 0;

    /* hardware configuration "demoapp" */
    if (IO_DemoApp) {
        int nErrors = Log_nError;

        /*** MIO initialization **/
        if (MIO_Init(NULL) < 0) {
            LogErrF(EC_General, "MIO initialization failed. I/O disabled (1)\n");
            IO_SelectNone();
            return -1;
        }
        // MIO_ModuleShow () ;

        /* ModuleSetUp */
        MIO_M5x_Config (Slot_AD); /* 12bit Analog In, (16 Channels) */
        MIO_M62_Config (Slot_DA); /* 12bit Analog Out, (16 Channels) */
        MIO_M5x_Config (Slot_CAN, -1 /* default IRQlevel*/);

        /* check for errors */
        if (nErrors != Log_nError) {
            LogErrF(EC_General, "MIO initializatiopn failed. I/O disabled (2)\n");
            IO_SelectNone();
            return -1;
        }

        MIO_M5x_SetComsParam (Slot_CAN, 0, 500000, 70, 2, 0);
        // configure acceptance filter for Rx CAN messages (here: accept all) */
        MIO_M5x_EnableIDS (Slot_CAN, 0, 0 /* start ID */ , 2048 /* num IDs */);
        // when working with extended CAN messages, activate transparent mode */
        // MIO_M5x_SetTransMode (Slot_CAN, 0, 1);
    }
    return 0;
}
```

**Figure 4.9:** Initialization of MIO hardware.
A brief explanation of the required initialization functions of MIO hardware are given below:

**MIO_Init ()**

```c
int MIO_Init(void *IO_Space)
```

This function initializes the MIO module. It has to be called once at program start and prior to any other MIO function. The registration table of M-Modules will be cleared, the configuration of all modules will get lost.

**MIO_GetModuleType ()**

```c
int MIO_GetModuleType(int Slot)
```

This function returns the module type (M-Module number) of the M-Module in the given Slot.

**MIO_RegisterModule ()**

```c
int MIO_RegisterModule(int Slot, int Type)
```

Before a M-Module can be used, it has to be registered first, by calling the function MIO_RegisterModule(). The registration of M-Modules is done automatically by the MIO_Mxx_Config() function.

**MIO_ResetModules ()**

```c
void MIO_ResetModules(void)
```

This function scans all configured M-Modules and calls the specific MIO_Mxx_Reset() functions, in order to (re-)initialize all configured M-Modules.
4.2.3 M51: Quadruple CAN interface

In the simulation environment AVL InMotion, the module M51 is used for the CAN interface. The block diagram of the connector assignment is shown in figure 4.10.

![Connector assignment diagram](image)

Figure 4.10: Connector assignment.

An overview of the necessary functions of the M51 module is given below.

**MIO_M51_Config ()**

```c
int MIO_M51_Config(int Slot, int IRQlevel)
```

By calling the function `MIO_M51_Config()` the configuration of the I/O hardware, i.e. the allocation of a M-Module card location with a module of type M51 is specified. This function must be called one time, before any attempted access to the module. `IRQlevel` specifies the priority that the interrupts will be handled.

**MIO_M51_Reset ()**

```c
void MIO_M51_Reset(int Slot)
```

The function resets all CAN interfaces back to the initial state (see `MIO_M51_Config()`). A running transmission, if necessary, is stopped and the messages in the send and receive buffers are lost. The transmission parameters and mailboxes must be reconfigured thereafter.

**MIO_M51_Delete ()**

```c
void MIO_M51_Delete(int Slot)
```
Deactivates the module located in Slot and frees memory and resources allocated to the module. All mailboxes for the module will be deleted and the resources deallocated. After the call to MIO_M51_Config() the function MIO_M51_Config() must be called before the module can be used again.

MIO_M51_SetBufSize()

void MIO_M51_SetBufSize(int Slot, int Ch, int RxSize, int TxSize)

The size of send and receive buffers can be changed by calling MIO_M51_SetBufSize(). Slot and Ch give the location and channel number of the CAN interface. RxSize and TxSize indicate the number of messages which receive and/or send buffers can hold. The following applies to both parameters: minimum buffer size is 2, the maximum size is not limited, and the default value is 64. Real time conditions will not be met when resizing the buffer to a larger size. Changing the buffer size to something smaller, however, will meet real-time conditions.

MIO_M51_SetCommParam()

int MIO_M51_SetCommParam(int Slot, int Ch, int Baudrate, int SamplePoint, int SynchJumpWidth, int Sample3)

With the function MIO_M51_SetCommParam() the CAN communication parameters are specified. Baudrate is the data transmission rate.

MIO_M51_EnableIds()

void MIO_M51_EnableIds(int Slot, int Ch, int StartId, int n)

It enables the receipt of the specified IDs, beginning with StartId and ending with StartId + (n-1). Slot and Ch give the location and channel number of the CAN interface.

MIO_M51_SetTransMode()

void MIO_M51_SetTransMode(int Slot, int Ch, int OnOff)

With MIO_M51_EnableIds() it is possible to freely determine the messages which can be received. Normally with the help of the acceptance mask of the interface module the messages are pre-selected (hardware function, no CPU load).
4.3 CANiogen-CANdb import tool

Connecting an ECU to the simulation environment AVL InMotion, normally requires to code all CAN messages that have to be transferred between the ECU and the simulation program. In the input functions, all the CAN signals are extracted from received CAN messages and its values are assigned to associated input quantities. Also, values of output quantities are coded into CAN messages, which are transmitted to the ECU.

However, coding the handling of CAN messages by hand takes a lot of time and is error prone. If you have a vector CAN data base, where the ECU is modeled within its surrounding CAN network, it is possible to automate this step with CANiogen.

CANiogen is a command line tool, which is intended to be called automatically during the compilation process of AVL InMotion. This guarantees, that changes to the CANdb and/or the simulation program will affect the behavior of simulation environment. The import of CANdb into the simulation environment AVL InMotion can be seen in the figure 4.1.

As input, CANiogen gets a vector CANdb file. This file models a CAN network with all its network nodes (ECU) by defining CAN messages and CAN signals, which are transferred between the ECUs. Depending on the test stand, there are ECUs which are included as hardware components. Others might be realized in software, which means that their behavior is simulated by AVL InMotion.
Now we have the possibility, to either call CANiogen with a list of ECUs which should be served by AVL InMotion. CANiogen will then assume, that these ECUs build the real part of the CAN network, and will import all information from the CAN data base which is necessary to simulate the rest of the CAN network. Or, we can give a list of simulated ECUs, that tells CANiogen, which ECUs build the simulated part of the CAN network.

CANiogen will then assume all other ECUs to be realized in hardware. In substitution for all ECUs, located in the simulated part of the CAN network, AVL InMotion will handle all CAN messages (both, reception and transmission), which these ECUs exchange with real networknodes. It is also possible to decide about receive or send additional single messages, as well as to choose to receive additional single signals, which are not automatically included by CANiogen. All these information are entirely extracted from the given CAN data base.

By default, CANiogen generates C-code as output, which is recommended for most applications. This C-code is capable of handling all CAN traffic, as configured with command line arguments.
Each CAN message has a unique message identifier and may contain up to 8 data bytes. The sender of a message codes CAN signals into the data bytes, to supply information to other ECUs. The CAN signals may vary in size, i.e. the data width need not to be a multiple of full bytes. Even worse, a CAN signal may be placed at any bit position within the data bytes, and can be stored in Motorola or in Intel byte order.

As a result, it is difficult to decode and encode the CAN signals by hand when programming the CAN traffic for an AVL InMotion application, and bugs are usual. By automating the following steps, the CANiogen helps in the following:

1. Manages transmission and reception of CAN messages for a given ECU, as specified in a CAN data base.
2. Generates CAN I/O variables for any signal of all imported CAN messages and ECUs.
3. Generates Data Dictionary entries for all generated I/O variable (Data Dictionary entries can be omitted generally or for single I/O variables).
4. Imports additional CAN messages and signals of special interest.
5. Retrieves information about how and how often to send a CAN message (cyclic, spontaneous,) from the CAN data base.
6. Fully supports extended CAN messages with 29-bit identifiers.
7. Is able to deal with multiplexed CAN signals.

**C-Generator mode**

In this mode, the output generated by CANiogen are isolated C-modules, which can be easily integrated in AVL InMotion/HIL. It is only needed to add about 5 lines of source code in the module `IO.c`, to activate the generated code. Additional lines need to be adapted for proper assignment between the generated I/O quantities and their associated model variables. Compared to the conventional method – coding everything by hand – the amount of code the user is responsible for, stays much more manageable.

The C-modules generated by CANiogen (typically called `IO_CAN`), will be represented through the files `IO_CAN.c`, `IO_CAN.h`, `IO_CAN_User.c`, `IO_CAN_User.h` and `IO_CAN_VarList.txt`. The header file `IO_CAN.h` defines a set of macros and fast inline functions to encode/decode signals of CAN messages and declare several public functions that have to be called in `IO.c`. Each imported CAN signal, will be presented by a generated
I/O variable. These variables will be added to the Data Dictionary, too. To simplify the use of generated I/O variables in AVL InMotion/HIL, the file IO_CAN_VarList.txt lists all generated I/O variables (CAN signals, CAN message timings, length of CAN messages), together with the data type and full variable name. The C-module file IO_CAN.c contains the generated functions. If not yet existing, CANiogen will also create a C-module called IO_CAN_User. This framework – represented by the files IO_CAN_User.c and IO_CAN_User.h – should be used to customize and refine the interface between model and CAN I/O. These files will never be overwritten by CANiogen.

4.3.1 Using CANiogen
To import a vector CAN data base, CANiogen requires the file name of a CANdb file (normally provided with the extension .dbc) and at least one of the options -simECU, -srvECU, -rcvMsg, -sndMsg or -rcvSig. If none of these options are given, CANiogen cannot import any data due to lack of information and exits with an error message:

```
> CANiogen CANdb.dbc
    CANiogen 1.2.2
    I/O generator for vector CAN data bases
    ERROR: please specify at least:
    - one ECU to handle receipt and transmission of CAN messages and/or
    - one message to receive from CAN bus and/or
    - one signal to receive from CAN bus and/or
    - one message to send on CAN bus
```

The options -simECU and -srvECU cannot be used simultaneously.

4.3.2 Importing Electronic Control Units (ECUs)

CANiogen offers two ways to import ECUs as a whole from a CAN data base: with the option - simECU and with the option -srvECU. These two options have oppositional effects, so they cannot be used simultaneously.

Simulating ECUs (-simECU option)

With the option -simECU we give a list of all ECUs in the CAN data base, which send CAN messages to the real ECUs of the test stand, or receive CAN messages from them. CANiogen accepts a comma separated list of ECUs, as named in the CAN data base.
Unknown ECU names will result in an error message. For this set of ECUs, CANiogen generates data structures and C code to manage all the CAN messages and signals, which are transferred to and from other ECUs in the same CAN network. Typically, the import list of ECUs includes all these ECUs of the CAN data base, which are simulated by AVL InMotion/HIL, except the ones which are part of the test stand hardware.

**Serving ECUs (-srvECU option)**

This option causes right the opposite to the option –simECU. Instead of specifying a list of ECUs to be simulated by AVL InMotion/HIL, we give a list of ECUs which are part of the real CAN network. The remaining part of the CAN network is to be considered as virtual.

CANiogen then generates C code to supply these ECUs with CAN messages and CAN signals from the virtual part of the CAN network. On the other hand, the information which the real ECUs provide to the rest of the CAN network will be read back, too. Typically, the import list of ECUs names all the ECUs, which are part of the test stand. The rest of the CAN network will be taken to be simulated by AVL InMotion/HIL.

**Excluding ECUs (-exeECU option)**

It is also possible that, a CAN data base might include even much more information than needed for the CAN network simulation of a specific test stand. There are often ECUs defined which can be ignored completely.

It’s the option –excECU, that allows you to give a list of ECUs which should be ignored completely when generating the data structures and C code for AVL InMotion/HIL.

**Sending arbitrary CAN messages**

Suppose, you have an ECU (ECU1) connected to your test stand, running a preliminary firmware version with limited operability. Another ECU (ECU2), which is also connected to the test stand, expects to receive a special CAN message (0x180) from ECU1. But, ECU1 which should send this message does not implement this function in its early version. So, we have to simulate this CAN message by software, in order to complete the virtual environment.
This is done with the option `-sndMsg`:

> CANiogen -srvECU ECU1,ECU2 -sndMsg 0x180 CANdb.dbc

The option `-sndMsg` tells CANiogen to generate all necessary I/O variables, timing data structures and C-code to send CAN messages, even if they are not part of the virtual CAN network.

### 4.4 Integration into AVL InMotion/HIL

This section applies only to CANiogen, if used in the C-code generator mode. The output of CANiogen are two largely independent C modules `IO_CAN` and `IO_CAN_User`, which are represented through the files `IO_CAN.c`, `IO_CAN.h`, `IO_CAN_User.c` and `IO_CAN_User.h`. While the files `IO_CAN.c` and `IO_CAN.h` should always be left unchanged, the files `IO_CAN_User.c` and `IO_CAN_User.h` can be customized by the user. The integration into a AVL InMotion/HIL, is done in three steps. First, we have to modify slightly the existing C module files `IO.c` and `User.c` to enable the CAN I/O part. Then build up the connection between parameters of the simulation models and the generated I/O variables, which is done in `IO_CAN_User.c`.

CANdb files (*.dbc) cannot be integrated to the project as such but have to be converted into C-code first. This is done automatically by the tool CANiogen which is called during the project compilation process. For CANiogen generate the code, information has to be provided in the file Make file in the `src` folder of the project and also the functions of the generated code have to be integrated into the files named `IO.c` and `User.c` of the project.

#### 4.4.1 Adaptation of Makefile

First, an object for caniogen has to be defined. The objects name will be the name of the files that are generated by CANiogen. (e.g. `IO_CAN.o` will generate `IO_CAN.c`, `IO_CAN.h`, etc.) Additionally, flags can be defined with which the functions that will be integrated in the code can be encapsuled. This makes (de-)activation of the code segments easier. Here, `-DXXX is the same than #define XXX in a c-file`.

For the generation of the code, the CANdb file for the given object has to be defined. Also, CANiogen needs to know for which node of the specified the communication has to be
generated (in respect to sending and receiving messages). The figure 4.12, shows the
definition of *Makefile* variables for IO_CAN modules.

The easiest way is to use the automatically generated rules for CANiogen. Perform the
following steps:

- **add object** `IO_CAN.o` **to the Makefile variable** `OBS CanIOGEN`
- **define preprocessor macro** `WITH_IO_CAN` **by adding** `-DWITH_IO_CAN` **to**
  the *Makefile* variable `DEF_CFLAGS`
- **create the variable** `IO_CAN_DB` **and assign the path to the CAN data base**
- **create the variable** `IO_CAN_FLAGS` **to hold the command line options for**
  CANiogen

```bash
### I/O generator for CANdb files
#
#/ ---
OBS_CANIOGEN := IO_CAN.o
DEF_CFLAGS += -DWITH_IO_CAN
#/ ---
#
# to include CANdb files:
# --------------------------
# - chose new icModule name (IO_XX)
# - append IO_XX.o to OBS_CANIOGEN
# - add -DWITH_IO_XX to DEF_CFLAGS
# - set IO_XX_DB variable to CANdb file
# - set IO_XX_FLAGS variable with desired command line parameters for CANiogen
# - modifications to IO.c:
#  * include IO_XX.h in IO.o
#  * call IO_XX_Init_First() in IO_Init_First()
#  * call IO_XX_Init() in IO_Init()
#  * call IO_XX_Recv(), IO_XX_RecvLoop() or IO_XX_RecvLoopSw() in IO_Recv()
#  * call IO_XX_Send() or IO_XX_SendSw() in IO_Send()
# - modifications to User.c:
#  * include IO_XX_User.h in User.c
#  * call IO_XX_User_Init() in User_Init()
#  * call IO_XX_User_Out() in User_Out()
# - use preprocessor define WITH_IO_XX to encapsulate IO_XX specific code in
#  all source and header files
#
#/ ---
# I/O Module IO_CAN.o
# IO_CAN_DB = ../CanDb/HCV.db
# IO_CAN_FLAGS = -simECU E55
#/ ---
#
### END (I/O generator for CANdb files)
```

**Figure 4.12:** Defining *Makefile* variables for **IO_CAN** modules.
4.4.2 Adaptation of IO.c

For the function calls within IO.c, the corresponding header file needs to be included. Also, for easier configuration, the used CAN card slot and the used CAN line are parameterized here.

```c
#if defined(WITH_IO_CAN)
 # include "IO_CAN.h"
  int IO_CAN_Slot = 0;
  int IO_CAN_Ch   = 2;
#endif

int
IO_Init_First (void)
{
  #if defined(WITH_IO_CAN)
    IO_CAN_Init_First();
  #endif
  ...
}
```

Please note that the numeration the CAN lines starts with 0. The figure 4.13 shows the numbering of the CAN lines.

![CAN Lines Numbering](image)

**Figure 4.13: Numbering of CAN lines**

In the function `IO_Init_First`, the data structures of the CAN communication has to be initialized. This is done by the generated.

In the function `IO_Init`, the parameterization of the used CAN line is done

```c
int
IO_Init (void)
{
  ...
  #if defined(WITH_IO_CAN)
    MIO_M51_SetCommParam(
      IO_CAN_Slot, //used CAN slot
      IO_CAN_Ch,   //used CAN line
      1000000,     //Baudrate
      70,          //Sample point
      2,           //SynchJumpWidth
      0);          //Sample3
    MIO_M51_EnableIds(
      IO_CAN_Slot, //used CAN slot
```
The initialization of the CAN communication is done as follows,

```c
#if defined(WITH_IO_CAN)
    IO_CAN_Init();
#endif
```

The reception of CAN messages is handled in `IO_In`. The generated code provides a function (`IO_CAN_RecvLoop`) that handles the CAN messages given in the `IO_CAN` module.

```c
void
IO_In (unsigned CycleNo)
{
    ...
#if defined(WITH_IO_CAN)
    IO_CAN_RecvLoop(
        IO_CAN_Slot, //used CAN slot
        IO_CAN_Ch,   //used CAN line
        CycleNo);    //current cycle number
#endif
    ...
}
```

If the sending of messages is also required, the according function (`IO_CAN_Send`) has to be added to `IO_Out`. The reception of CAN messages is handled in `IO_In`. The generated code provides a function that handles the CAN messages given in the `IO_CAN` module.

```c
void
IO_Out (unsigned CycleNo)
{
    ...
#if defined(WITH_IO_CAN)
    IO_CAN_Send(
        IO_CAN_Slot, //used CAN slot
        IO_CAN_Ch,   //used CAN line
        CycleNo);    //current cycle number
#endif
    ...
}
```
4.4.3 Adaptation of User.c

CANiogen also generates *_User-files where user defined adaptations to the CAN communication can be made, e.g. calibration or assignment to simulation variables. (Please note, that those files are also regenerated during re-compilation of the CANdb code, so changes there have to be redone or copied from the automatically generated backup).

For the function calls, the corresponding header files need to be included.

```c
#if defined(WITH_IO_CAN)
# include "IO_CAN.h"
# include "IO_CAN_User.h"
#endif
```

After having received new values via CAN, these values have to be assigned to the appropriate simulation quantity. If this was done in IO_CAN_User.c, than the according IO_CAN_User_In function has to be called. This is done in the function User_In.

```c
void
User_In (const unsigned CycleNo)
{
    ...
    #if defined(WITH_IO_CAN)
        IO_CAN_User_In(CycleNo);
    #endif
    if (SimCore.State != SCState_Simulate)
        return;
    ...
}
```

Before being sent to the hardware, the output signals have to be updated with the simulation result. If this was again done in IO_CAN_User.c, than the according IO_CAN_User_Out function has to be called in User_Out.

```c
void
User_Out (const unsigned CycleNo)
{
    ...
    #if defined(WITH_IO_CAN)
        IO_CAN_User_Out(CycleNo);
    #endif
    if (SimCore.State != SCState_Simulate)
        return;
```
5. Results and discussion

As described in section 4.3, C- generated mode - after successful integration of the database file, the generated files are given below which shows the implementation of communication protocol developed in WP6320

- **APP_CAN_2_VarList.txt**

  It contains all generated I/O variables (CAN signals, CAN message timings, length of CAN messages), together with the data type and full variable name.

- **APP_CAN_2.c**

  The C-module file **IO_CAN.c** contains the generated functions.

- **APP_CAN_2.h**

  This header file defines a set of macros and fast inline functions to encode/decode signals of CAN messages and declare several public functions that have to be called in **IO.c**

---

Figure 5.1: compilation of Communication protocol into the simulation platform
Figure 5.2: Integrating into the simulation platform AVL InMotion/HiL
### APP_CAN_2_VarList.txt

This file lists all I/O variables, generated by CANiogen 2.0.6

#### Created: 2013/05/02 13:09:53

<table>
<thead>
<tr>
<th>signal</th>
<th>data type</th>
<th>full variable name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGN001_DLC</td>
<td>int</td>
<td>APP_CAN_2.ESS.PGN001_DLC</td>
</tr>
<tr>
<td>Pos_temp_1</td>
<td>char</td>
<td>APP_CAN_2.ESS.PGN001.Pos_temp_1</td>
</tr>
<tr>
<td></td>
<td>char</td>
<td>APP_CAN_2.Raw.ESS.PGN001.Pos_temp_1</td>
</tr>
<tr>
<td></td>
<td>char</td>
<td>APP_CAN_2_State.ESS.PGN001.Pos_temp_1</td>
</tr>
<tr>
<td>Pos_temp_2</td>
<td>char</td>
<td>APP_CAN_2.ESS.PGN001.Pos_temp_2</td>
</tr>
<tr>
<td></td>
<td>char</td>
<td>APP_CAN_2.Raw.ESS.PGN001.Pos_temp_2</td>
</tr>
<tr>
<td></td>
<td>char</td>
<td>APP_CAN_2_State.ESS.PGN001.Pos_temp_2</td>
</tr>
<tr>
<td>Pos_temp_3</td>
<td>char</td>
<td>APP_CAN_2.ESS.PGN001.Pos_temp_3</td>
</tr>
<tr>
<td></td>
<td>char</td>
<td>APP_CAN_2.Raw.ESS.PGN001.Pos_temp_3</td>
</tr>
<tr>
<td></td>
<td>char</td>
<td>APP_CAN_2_State.ESS.PGN001.Pos_temp_3</td>
</tr>
<tr>
<td>Pos_temp_4</td>
<td>char</td>
<td>APP_CAN_2.ESS.PGN001.Pos_temp_4</td>
</tr>
<tr>
<td></td>
<td>char</td>
<td>APP_CAN_2.Raw.ESS.PGN001.Pos_temp_4</td>
</tr>
<tr>
<td></td>
<td>char</td>
<td>APP_CAN_2_State.ESS.PGN001.Pos_temp_4</td>
</tr>
<tr>
<td>Pos_temp_5</td>
<td>char</td>
<td>APP_CAN_2.ESS.PGN001.Pos_temp_5</td>
</tr>
<tr>
<td></td>
<td>char</td>
<td>APP_CAN_2.Raw.ESS.PGN001.Pos_temp_5</td>
</tr>
<tr>
<td></td>
<td>char</td>
<td>APP_CAN_2_State.ESS.PGN001.Pos_temp_5</td>
</tr>
<tr>
<td>Temp_cool_1</td>
<td>short</td>
<td>APP_CAN_2.ESS.PGN001.Temp_cool_1</td>
</tr>
<tr>
<td></td>
<td>unsigned char</td>
<td>APP_CAN_2.Raw.ESS.PGN001.Temp_cool_1</td>
</tr>
<tr>
<td></td>
<td>char</td>
<td>APP_CAN_2_State.ESS.PGN001.Temp_cool_1</td>
</tr>
<tr>
<td>Temp_cool_2</td>
<td>short</td>
<td>APP_CAN_2.ESS.PGN001.Temp_cool_2</td>
</tr>
<tr>
<td></td>
<td>unsigned char</td>
<td>APP_CAN_2.Raw.ESS.PGN001.Temp_cool_2</td>
</tr>
<tr>
<td></td>
<td>char</td>
<td>APP_CAN_2_State.ESS.PGN001.Temp_cool_2</td>
</tr>
<tr>
<td>Temp_cool_3</td>
<td>short</td>
<td>APP_CAN_2.ESS.PGN001.Temp_cool_3</td>
</tr>
<tr>
<td></td>
<td>unsigned char</td>
<td>APP_CAN_2.Raw.ESS.PGN001.Temp_cool_3</td>
</tr>
<tr>
<td></td>
<td>char</td>
<td>APP_CAN_2_State.ESS.PGN001.Temp_cool_3</td>
</tr>
<tr>
<td>Temp_cool_4</td>
<td>short</td>
<td>APP_CAN_2.ESS.PGN001.Temp_cool_4</td>
</tr>
<tr>
<td></td>
<td>unsigned char</td>
<td>APP_CAN_2.Raw.ESS.PGN001.Temp_cool_4</td>
</tr>
<tr>
<td></td>
<td>char</td>
<td>APP_CAN_2_State.ESS.PGN001.Temp_cool_4</td>
</tr>
<tr>
<td>Temp_cool_5</td>
<td>short</td>
<td>APP_CAN_2.ESS.PGN001.Temp_cool_5</td>
</tr>
<tr>
<td></td>
<td>unsigned char</td>
<td>APP_CAN_2.Raw.ESS.PGN001.Temp_cool_5</td>
</tr>
<tr>
<td></td>
<td>char</td>
<td>APP_CAN_2_State.ESS.PGN001.Temp_cool_5</td>
</tr>
</tbody>
</table>
### CAN Msg 0x002 (PGN002):
<table>
<thead>
<tr>
<th>signal</th>
<th>data type</th>
<th>full variable name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGN002_DLC int</td>
<td>APP_CAN_2.ESS.PGN002_DLC</td>
<td></td>
</tr>
<tr>
<td>ESS_Req unsigned char</td>
<td>APP_CAN_2.ESS.PGN002.ESS_Req</td>
<td></td>
</tr>
<tr>
<td>ESS_Req char</td>
<td>APP_CAN_2_State.ESS.PGN002.ESS_Req</td>
<td></td>
</tr>
</tbody>
</table>

### CAN Msg 0x003 (PGN003):
<table>
<thead>
<tr>
<th>signal</th>
<th>data type</th>
<th>full variable name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGN003_DLC int</td>
<td>APP_CAN_2.ESS.PGN003_DLC</td>
<td></td>
</tr>
<tr>
<td>ESS_Act_Cur float</td>
<td>APP_CAN_2.ESS.PGN003.ESS_Act_Cur</td>
<td></td>
</tr>
<tr>
<td>ESS_Act_Cur unsigned short</td>
<td>APP_CAN_2_State.ESS.PGN003.ESS_Act_Cur</td>
<td></td>
</tr>
<tr>
<td>ESS_Act_Cur char</td>
<td>APP_CAN_2_State.ESS.PGN003.ESS_Act_Cur</td>
<td></td>
</tr>
<tr>
<td>ESS_Act_Vol float</td>
<td>APP_CAN_2.ESS.PGN003.ESS_Act_Vol</td>
<td></td>
</tr>
<tr>
<td>ESS_Act_Vol short</td>
<td>APP_CAN_2_State.ESS.PGN003.ESS_Act_Vol</td>
<td></td>
</tr>
<tr>
<td>ESS_Act_Vol char</td>
<td>APP_CAN_2_State.ESS.PGN003.ESS_Act_Vol</td>
<td></td>
</tr>
<tr>
<td>ESS_Char_Cur float</td>
<td>APP_CAN_2.ESS.PGN003.ESS_Char_Cur</td>
<td></td>
</tr>
<tr>
<td>ESS_Char_Cur short</td>
<td>APP_CAN_2_State.ESS.PGN003.ESS_Char_Cur</td>
<td></td>
</tr>
<tr>
<td>ESS_Char_Cur char</td>
<td>APP_CAN_2_State.ESS.PGN003.ESS_Char_Cur</td>
<td></td>
</tr>
<tr>
<td>ESS_Dis_Cur float</td>
<td>APP_CAN_2.ESS.PGN003.ESS_Dis_Cur</td>
<td></td>
</tr>
<tr>
<td>ESS_Dis_Cur short</td>
<td>APP_CAN_2_State.ESS.PGN003.ESS_Dis_Cur</td>
<td></td>
</tr>
<tr>
<td>ESS_Dis_Cur char</td>
<td>APP_CAN_2_State.ESS.PGN003.ESS_Dis_Cur</td>
<td></td>
</tr>
</tbody>
</table>

### CAN Msg 0x004 (PGN004):
<table>
<thead>
<tr>
<th>signal</th>
<th>data type</th>
<th>full variable name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGN004_DLC int</td>
<td>APP_CAN_2.ESS.PGN004_DLC</td>
<td></td>
</tr>
<tr>
<td>ESS_Act_Pow float</td>
<td>APP_CAN_2.ESS.PGN004.ESS_Act_Pow</td>
<td></td>
</tr>
<tr>
<td>ESS_Act_Pow short</td>
<td>APP_CAN_2_State.ESS.PGN004.ESS_Act_Pow</td>
<td></td>
</tr>
<tr>
<td>ESS_Act_Pow char</td>
<td>APP_CAN_2_State.ESS.PGN004.ESS_Act_Pow</td>
<td></td>
</tr>
<tr>
<td>ESS_Act_Tor float</td>
<td>APP_CAN_2.ESS.PGN004.ESS_Act_Tor</td>
<td></td>
</tr>
<tr>
<td>ESS_Act_Tor unsigned short</td>
<td>APP_CAN_2_State.ESS.PGN004.ESS_Act_Tor</td>
<td></td>
</tr>
<tr>
<td>ESS_Act_Tor char</td>
<td>APP_CAN_2_State.ESS.PGN004.ESS_Act_Tor</td>
<td></td>
</tr>
<tr>
<td>ESS_Max_Char_Pow float</td>
<td>APP_CAN_2.ESS.PGN004.ESS_Max_Char_Pow</td>
<td></td>
</tr>
<tr>
<td>ESS_Max_Char_Pow unsigned char</td>
<td>APP_CAN_2_State.ESS.PGN004.ESS_Max_Char_Pow</td>
<td></td>
</tr>
<tr>
<td>ESS_Max_Char_Pow char</td>
<td>APP_CAN_2_State.ESS.PGN004.ESS_Max_Char_Pow</td>
<td></td>
</tr>
<tr>
<td>ESS_Max_Dischar_Pow float</td>
<td>APP_CAN_2.ESS.PGN004.ESS_Max_Dischar_Pow</td>
<td></td>
</tr>
<tr>
<td>ESS_Max_Dischar_Pow unsigned char</td>
<td>APP_CAN_2_State.ESS.PGN004.ESS_Max_Dischar_Pow</td>
<td></td>
</tr>
<tr>
<td>ESS_Max_Dischar_Pow char</td>
<td>APP_CAN_2_State.ESS.PGN004.ESS_Max_Dischar_Pow</td>
<td></td>
</tr>
</tbody>
</table>

### CAN Msg 0x005 (PGN005):
<table>
<thead>
<tr>
<th>signal</th>
<th>data type</th>
<th>full variable name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGN005_DLC int</td>
<td>APP_CAN_2.ESS.PGN005_DLC</td>
<td></td>
</tr>
<tr>
<td>ESS_Act_Flow float</td>
<td>APP_CAN_2.ESS.PGN005.ESS_Act_Flow</td>
<td></td>
</tr>
<tr>
<td>ESS_Act_Flow short</td>
<td>APP_CAN_2_State.ESS.PGN005.ESS_Act_Flow</td>
<td></td>
</tr>
<tr>
<td>ESS_Act_Flow char</td>
<td>APP_CAN_2_State.ESS.PGN005.ESS_Act_Flow</td>
<td></td>
</tr>
<tr>
<td>ESS_Act_Pres unsigned int</td>
<td>APP_CAN_2.ESS.PGN005.ESS_Act_Pres</td>
<td></td>
</tr>
<tr>
<td>ESS_Act_Pres unsigned short</td>
<td>APP_CAN_2_State.ESS.PGN005.ESS_Act_Pres</td>
<td></td>
</tr>
<tr>
<td>ESS_Act_Pres char</td>
<td>APP_CAN_2_State.ESS.PGN005.ESS_Act_Pres</td>
<td></td>
</tr>
<tr>
<td>ESS_Act_Speed unsigned int</td>
<td>APP_CAN_2.ESS.PGN005.ESS_Act_Speed</td>
<td></td>
</tr>
<tr>
<td>ESS_Act_Speed unsigned short</td>
<td>APP_CAN_2_State.ESS.PGN005.ESS_Act_Speed</td>
<td></td>
</tr>
</tbody>
</table>
### CAN Msg 0x006 (PGN006):

<table>
<thead>
<tr>
<th>signal</th>
<th>data type</th>
<th>full variable name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGN006_DLC</td>
<td>int</td>
<td>APP_CAN_2.ESS.PGN006_DLC</td>
</tr>
<tr>
<td>ESS_Avg_System</td>
<td>short</td>
<td>APP_CAN_2.ESS.PGN006.ESS_Avg_System</td>
</tr>
<tr>
<td>ESS_High_System</td>
<td>short</td>
<td>APP_CAN_2.ESS.PGN006.ESS_High_System</td>
</tr>
<tr>
<td>ESS_Low_System</td>
<td>short</td>
<td>APP_CAN_2.ESS.PGN006.ESS_Low_System</td>
</tr>
<tr>
<td>ESS_Max_System</td>
<td>short</td>
<td>APP_CAN_2.ESS.PGN006.ESS_Max_System</td>
</tr>
<tr>
<td>ESS_Min_System</td>
<td>short</td>
<td>APP_CAN_2.ESS.PGN006.ESS_Min_System</td>
</tr>
<tr>
<td>ESS_Status</td>
<td>unsigned char</td>
<td>APP_CAN_2.ESS.PGN006.ESS_Status</td>
</tr>
</tbody>
</table>

### CAN Msg 0x007 (PGN007):

<table>
<thead>
<tr>
<th>signal</th>
<th>data type</th>
<th>full variable name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGN007_DLC</td>
<td>int</td>
<td>APP_CAN_2.ESS.PGN007_DLC</td>
</tr>
<tr>
<td>ESS_Con_Charge_Cur</td>
<td>float</td>
<td>APP_CAN_2.ESS.PGN007.ESS_Con_Charge_Cur</td>
</tr>
<tr>
<td>ESS_Clon_Discharge_Cur</td>
<td>float</td>
<td>APP_CAN_2.ESS.PGN007.ESS_Clon_Discharge_Cur</td>
</tr>
<tr>
<td>ESS_Elec_Capa</td>
<td>float</td>
<td>APP_CAN_2.ESS.PGN007.ESS_Elec_Capa</td>
</tr>
<tr>
<td>ESS_SOC</td>
<td>float</td>
<td>APP_CAN_2.ESS.PGN007.ESS_SOC</td>
</tr>
</tbody>
</table>

### CAN Msg 0x008 (PGN008):

<table>
<thead>
<tr>
<th>signal</th>
<th>data type</th>
<th>full variable name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGN008_DLC</td>
<td>int</td>
<td>APP_CAN_2.ESS.PGN008_DLC</td>
</tr>
<tr>
<td>ESS_Max_Vol</td>
<td>float</td>
<td>APP_CAN_2.ESS.PGN008.ESS_Max_Vol</td>
</tr>
<tr>
<td>ESS_Min_Vol</td>
<td>float</td>
<td>APP_CAN_2.ESS.PGN008.ESS_Min_Vol</td>
</tr>
<tr>
<td>ESS_Peak_Charge_Cur</td>
<td>float</td>
<td>APP_CAN_2.ESS.PGN008.ESS_Peak_Charge_Cur</td>
</tr>
</tbody>
</table>
## HCV Hybrid Commercial Vehicle

### CAN Msg 0x009 (PGN009):

<table>
<thead>
<tr>
<th>signal</th>
<th>data type</th>
<th>full variable name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGN09_DLC</td>
<td>int</td>
<td>APP_CAN_2.ESS.PGN09_DLC</td>
</tr>
<tr>
<td>ESS_Max_Flow</td>
<td>float</td>
<td>APP_CAN_2.ESS.PGN09.ESS_Max_Flow</td>
</tr>
<tr>
<td>short</td>
<td>APP_CAN_2.Raw.ESS.PGN09.ESS_Max_Flow</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>APP_CAN_2.State.ESS.PGN09.ESS_Max_Flow</td>
<td></td>
</tr>
<tr>
<td>ESS_Max_Pres</td>
<td>unsigned int</td>
<td>APP_CAN_2.ESS.PGN09.ESS_Max_Pres</td>
</tr>
<tr>
<td>unsigned short</td>
<td>APP_CAN_2.Raw.ESS.PGN09.ESS_Max_Pres</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>APP_CAN_2.State.ESS.PGN09.ESS_Max_Pres</td>
<td></td>
</tr>
<tr>
<td>ESS_Min_Flow</td>
<td>float</td>
<td>APP_CAN_2.ESS.PGN09.ESS_Min_Flow</td>
</tr>
<tr>
<td>short</td>
<td>APP_CAN_2.Raw.ESS.PGN09.ESS_Min_Flow</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>APP_CAN_2.State.ESS.PGN09.ESS_Min_Flow</td>
<td></td>
</tr>
<tr>
<td>ESS_Min_Pres</td>
<td>unsigned int</td>
<td>APP_CAN_2.ESS.PGN09.ESS_Min_Pres</td>
</tr>
<tr>
<td>unsigned short</td>
<td>APP_CAN_2.Raw.ESS.PGN09.ESS_Min_Pres</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>APP_CAN_2.State.ESS.PGN09.ESS_Min_Pres</td>
<td></td>
</tr>
</tbody>
</table>

### CAN Msg 0x010 (PGN010):

<table>
<thead>
<tr>
<th>signal</th>
<th>data type</th>
<th>full variable name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGN10_DLC</td>
<td>int</td>
<td>APP_CAN_2.ESS.PGN10_DLC</td>
</tr>
<tr>
<td>ESS_Speed</td>
<td>unsigned int</td>
<td>APP_CAN_2.ESS.PGN10.ESS_Spd</td>
</tr>
<tr>
<td>unsigned short</td>
<td>APP_CAN_2.Raw.ESS.PGN10.ESS_Spd</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>APP_CAN_2.State.ESS.PGN10.ESS_Spd</td>
<td></td>
</tr>
<tr>
<td>ESS_Mot</td>
<td>float</td>
<td>APP_CAN_2.ESS.PGN10.ESS_Mot</td>
</tr>
<tr>
<td>short</td>
<td>APP_CAN_2.Raw.ESS.PGN10.ESS_Mot</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>APP_CAN_2.State.ESS.PGN10.ESS_Mot</td>
<td></td>
</tr>
<tr>
<td>ESS_Min_Spd</td>
<td>unsigned int</td>
<td>APP_CAN_2.ESS.PGN10.ESS_Min_Spd</td>
</tr>
<tr>
<td>unsigned short</td>
<td>APP_CAN_2.Raw.ESS.PGN10.ESS_Min_Spd</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>APP_CAN_2.State.ESS.PGN10.ESS_Min_Spd</td>
<td></td>
</tr>
<tr>
<td>ESS_Fow_Ref</td>
<td>short</td>
<td>APP_CAN_2.ESS.PGN10.ESS_Fow_Ref</td>
</tr>
<tr>
<td>short</td>
<td>APP_CAN_2.Raw.ESS.PGN10.ESS_Fow_Ref</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>APP_CAN_2.State.ESS.PGN10.ESS_Fow_Ref</td>
<td></td>
</tr>
</tbody>
</table>

### CAN Msg 0x011 (PGN011):

<table>
<thead>
<tr>
<th>signal</th>
<th>data type</th>
<th>full variable name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGN11_DLC</td>
<td>int</td>
<td>APP_CAN_2.ESS.PGN11_DLC</td>
</tr>
<tr>
<td>ESS_Con_Ch_Pow</td>
<td>float</td>
<td>APP_CAN_2.ESS.PGN11.ESS_Con_Ch_Pow</td>
</tr>
<tr>
<td>unsigned char</td>
<td>APP_CAN_2.Raw.ESS.PGN11.ESS_Con_Ch_Pow</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>APP_CAN_2.State.ESS.PGN11.ESS_Con_Ch_Pow</td>
<td></td>
</tr>
<tr>
<td>ESS_Con_disch_Pow</td>
<td>float</td>
<td>APP_CAN_2.ESS.PGN11.ESS_Con_disch_Pow</td>
</tr>
<tr>
<td>unsigned char</td>
<td>APP_CAN_2.Raw.ESS.PGN11.ESS_Con_disch_Pow</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>APP_CAN_2.State.ESS.PGN11.ESS_Con_disch_Pow</td>
<td></td>
</tr>
<tr>
<td>ESS_Mot_Pow</td>
<td>float</td>
<td>APP_CAN_2.ESS.PGN11.ESS_Mot_Pow</td>
</tr>
<tr>
<td>unsigned char</td>
<td>APP_CAN_2.Raw.ESS.PGN11.ESS_Mot_Pow</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>APP_CAN_2.State.ESS.PGN11.ESS_Mot_Pow</td>
<td></td>
</tr>
<tr>
<td>ESS_Stt_Pow</td>
<td>float</td>
<td>APP_CAN_2.ESS.PGN11.ESS_Stt_Pow</td>
</tr>
<tr>
<td>unsigned char</td>
<td>APP_CAN_2.Raw.ESS.PGN11.ESS_Stt_Pow</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>APP_CAN_2.State.ESS.PGN11.ESS_Stt_Pow</td>
<td></td>
</tr>
<tr>
<td>ESS_Max_ChRTL</td>
<td>float</td>
<td>APP_CAN_2.ESS.PGN11.ESS_Max_ChRTL</td>
</tr>
<tr>
<td>unsigned char</td>
<td>APP_CAN_2.Raw.ESS.PGN11.ESS_Max_ChRTL</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>APP_CAN_2.State.ESS.PGN11.ESS_Max_ChRTL</td>
<td></td>
</tr>
<tr>
<td>ESS_Mot_Disch_Pow</td>
<td>float</td>
<td>APP_CAN_2.ESS.PGN11.ESS_Mot_Disch_Pow</td>
</tr>
<tr>
<td>unsigned char</td>
<td>APP_CAN_2.Raw.ESS.PGN11.ESS_Mot_Disch_Pow</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>APP_CAN_2.State.ESS.PGN11.ESS_Mot_Disch_Pow</td>
<td></td>
</tr>
<tr>
<td>ESS_Max_disch_Pow</td>
<td>float</td>
<td>APP_CAN_2.ESS.PGN11.ESS_Max_disch_Pow</td>
</tr>
<tr>
<td>unsigned char</td>
<td>APP_CAN_2.Raw.ESS.PGN11.ESS_Max_disch_Pow</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>APP_CAN_2.State.ESS.PGN11.ESS_Max_disch_Pow</td>
<td></td>
</tr>
</tbody>
</table>
# Timing variables

<table>
<thead>
<tr>
<th>ECU ESS:</th>
<th>message</th>
<th>data type</th>
<th>full variable name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGN001</td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN001.SendPeriod</td>
</tr>
<tr>
<td></td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN001.SendDistrib</td>
</tr>
<tr>
<td>PGN002</td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN002.SendPeriod</td>
</tr>
<tr>
<td></td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN002.SendDistrib</td>
</tr>
<tr>
<td>PGN003</td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN003.SendPeriod</td>
</tr>
<tr>
<td></td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN003.SendDistrib</td>
</tr>
<tr>
<td>PGN004</td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN004.SendPeriod</td>
</tr>
<tr>
<td></td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN004.SendDistrib</td>
</tr>
<tr>
<td>PGN005</td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN005.SendPeriod</td>
</tr>
<tr>
<td></td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN005.SendDistrib</td>
</tr>
<tr>
<td>PGN006</td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN006.SendPeriod</td>
</tr>
<tr>
<td></td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN006.SendDistrib</td>
</tr>
<tr>
<td>PGN007</td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN007.SendPeriod</td>
</tr>
<tr>
<td></td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN007.SendDistrib</td>
</tr>
<tr>
<td>PGN008</td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN008.SendPeriod</td>
</tr>
<tr>
<td></td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN008.SendDistrib</td>
</tr>
<tr>
<td>PGN009</td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN009.SendPeriod</td>
</tr>
<tr>
<td></td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN009.SendDistrib</td>
</tr>
<tr>
<td>PGN010</td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN010.SendPeriod</td>
</tr>
<tr>
<td></td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN010.SendDistrib</td>
</tr>
<tr>
<td>PGN011</td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN011.SendPeriod</td>
</tr>
<tr>
<td></td>
<td></td>
<td>int</td>
<td>APP_CAN_2_Timings.ESS.PGN011.SendDistrib</td>
</tr>
</tbody>
</table>
APP_CAN_2.c

/** Simulated ECUs (all Rx- and cyclic Tx- CAN messages are handled):
 ** ================================================================================
 **  - ESS
 **  **
 **  Handled Tx messages for:
 **  /navbar**************************************************************************
 **  - ECU ESS:
 **  + PGNO01 (0x001)
 **  + PGNO02 (0x002)
 **  + PGNO03 (0x003)
 **  + PGNO04 (0x004)
 **  + PGNO05 (0x005)
 **  + PGNO06 (0x006)
 **  + PGNO07 (0x007)
 **  + PGNO08 (0x008)
 **  + PGNO09 (0x009)
 **  + PGNO10 (0x010)
 **  + PGNO11 (0x011)
 ** ================================================================================
 */

#ifdef(DSPACE)
  #include <ipgsr.h>
  #include <dsio.h>
#endif

#ifdef(XENO)
  #define _T50C99_SOURCE 1
  #include <math.h>
  #include <string.h>
  #include <mic.h>
#endif

#include <MathUtils.h>
#include <DataDict.h>
#include <Log.h>

#include "APP_CAN_2_User.h"
#include "APP_CAN_2.h"

/* Signal states */
int APP_CAN_2_SigStates[APP_CAN_2_SState_nStates];

/* I/O vector for CAN communication */
tAPP_CAN_2Vec APP_CAN_2;
tAPP_CAN_2Timings APP_CAN_2_Timings;
/**
 * APP_CAN_2_Init()
 */

int
APP_CAN_2_Init(void)
{
    /* Check recv function for CAN messages */
    if (APP_CAN_2_CAN_Recv == NULL) {
        LogErrF(EC_Init, "APP_CAN_2: don't know how to receive CAN messages");
        return -1;
    }

    /* Check send function for CAN messages */
    if (APP_CAN_2_CAN_Send == NULL) {
        LogErrF(EC_Init, "APP_CAN_2: don't know how to send CAN messages");
        return -1;
    }

    /* Call APP_CAN_2_User_Init() of User */
    return APP_CAN_2_User_Init();
}

int
APP_CAN_2_Recv(
    struct CAN_Msg *Msg,
    const unsigned CycleNo)
{
    if (APP_CAN_2_RxHook != NULL
        && APP_CAN_2_RxHook(Msg, CycleNo, APP_CAN_2_PRE_DECODE) < 0) {
        return 1;
    }
    switch (Msg->MsgId | (Msg->FrameFmt<<31)) {
        default:
            return -1;
        }
    if (APP_CAN_2_RxHook != NULL)
        APP_CAN_2_RxHook(Msg, CycleNo, APP_CAN_2_POST_DECODE);

    return 0;
}
void APP_CAN_2_RecvLoop(
    int Slot,
    int Channel,
    const unsigned CycleNo)
{
    CAN_Msg Msg;

    while (APP_CAN_2_CAN_Recv(Slot, Channel, &Msg) == 0)
        APP_CAN_2_Recv(&Msg, CycleNo);
}

int APP_CAN_2_Send(
    int Slot,
    int Channel,
    const unsigned CycleNo)
{
    /* ECU ESS */
    /* CAN Msg 0x001 (PGN001) */
    if (APP_CAN_2_Timings.ESS.PGN001.SendPeriod > 0
        && (CycleNo%APP_CAN_2_Timings.ESS.PGN001.SendPeriod
            == APP_CAN_2_Timings.ESS.PGN001.SendDistrib)) {
        APP_CAN_2_SendMsg(Slot, Channel, CycleNo, APP_CAN_2_PGN001_ID);
    }

    /* CAN Msg 0x002 (PGN002) */
    if (APP_CAN_2_Timings.ESS.PGN002.SendPeriod > 0
        && (CycleNo%APP_CAN_2_Timings.ESS.PGN002.SendPeriod
            == APP_CAN_2_Timings.ESS.PGN002.SendDistrib)) {
        APP_CAN_2_SendMsg(Slot, Channel, CycleNo, APP_CAN_2_PGN002_ID);
    }

    /* CAN Msg 0x003 (PGN003) */
    if (APP_CAN_2_Timings.ESS.PGN003.SendPeriod > 0
        && (CycleNo%APP_CAN_2_Timings.ESS.PGN003.SendPeriod
            == APP_CAN_2_Timings.ESS.PGN003.SendDistrib)) {
        APP_CAN_2_SendMsg(Slot, Channel, CycleNo, APP_CAN_2_PGN003_ID);
    }

    /* CAN Msg 0x004 (PGN004) */
    if (APP_CAN_2_Timings.ESS.PGN004.SendPeriod > 0
        && (CycleNo%APP_CAN_2_Timings.ESS.PGN004.SendPeriod
            == APP_CAN_2_Timings.ESS.PGN004.SendDistrib)) {
        APP_CAN_2_SendMsg(Slot, Channel, CycleNo, APP_CAN_2_PGN004_ID);
    }

    /* CAN Msg 0x005 (PGN005) */
    if (APP_CAN_2_Timings.ESS.PGN005.SendPeriod > 0
        && (CycleNo%APP_CAN_2_Timings.ESS.PGN005.SendPeriod
            == APP_CAN_2_Timings.ESS.PGN005.SendDistrib)) {
        APP_CAN_2_SendMsg(Slot, Channel, CycleNo, APP_CAN_2_PGN005_ID);
    }
}
/* CAN Msg 0x006 (PGN006) */
if (APP_CAN_2_Timings.ESS.PGN006.SendPeriod > 0
    && (CycleNo%APP_CAN_2_Timings.ESS.PGN006.SendPeriod
        == APP_CAN_2_Timings.ESS.PGN006.SendDistrib))
    APP_CAN_2_SendMsg(Slot, Channel, CycleNo, APP_CAN_2_PGN006_ID);
}

/* CAN Msg 0x007 (PGN007) */
if (APP_CAN_2_Timings.ESS.PGN007.SendPeriod > 0
    && (CycleNo%APP_CAN_2_Timings.ESS.PGN007.SendPeriod
        == APP_CAN_2_Timings.ESS.PGN007.SendDistrib))
    APP_CAN_2_SendMsg(Slot, Channel, CycleNo, APP_CAN_2_PGN007_ID);
}

/* CAN Msg 0x008 (PGN008) */
if (APP_CAN_2_Timings.ESS.PGN008.SendPeriod > 0
    && (CycleNo%APP_CAN_2_Timings.ESS.PGN008.SendPeriod
        == APP_CAN_2_Timings.ESS.PGN008.SendDistrib))
    APP_CAN_2_SendMsg(Slot, Channel, CycleNo, APP_CAN_2_PGN008_ID);
}

/* CAN Msg 0x009 (PGN009) */
if (APP_CAN_2_Timings.ESS.PGN009.SendPeriod > 0
    && (CycleNo%APP_CAN_2_Timings.ESS.PGN009.SendPeriod
        == APP_CAN_2_Timings.ESS.PGN009.SendDistrib))
    APP_CAN_2_SendMsg(Slot, Channel, CycleNo, APP_CAN_2_PGN009_ID);
}

/* CAN Msg 0x010 (PGN010) */
if (APP_CAN_2_Timings.ESS.PGN010.SendPeriod > 0
    && (CycleNo%APP_CAN_2_Timings.ESS.PGN010.SendPeriod
        == APP_CAN_2_Timings.ESS.PGN010.SendDistrib))
    APP_CAN_2_SendMsg(Slot, Channel, CycleNo, APP_CAN_2_PGN010_ID);
}

/* CAN Msg 0x011 (PGN011) */
if (APP_CAN_2_Timings.ESS.PGN011.SendPeriod > 0
    && (CycleNo%APP_CAN_2_Timings.ESS.PGN011.SendPeriod
        == APP_CAN_2_Timings.ESS.PGN011.SendDistrib))
    APP_CAN_2_SendMsg(Slot, Channel, CycleNo, APP_CAN_2_PGN011_ID);
}

return 0;
}
# APP_CAN_2.h

```c
#ifndef _APP_CAN_2_H_
#define _APP_CAN_2_H_

#include <cplusplus>

extern "C" {
    #ifndef CAN_Msg;

    #if defined( DS1005)
        /* data type long long not supported on DS1005 */
        typedef LONG_LONG In64
        typedef UNSIGNED_LONG_LONG UInt64
        typedef DD4cDefLLong(s, u, n, p)
        typedef DD4cDefULLong(s, u, n, p)
    #else
        #define LONG_LONG long long
        #define UNSIGNED_LONG_LONG unsigned long long
    #endif

    /* Signal states */
    enum {
        APP_CAN_2_SState_None = 0,
        APP_CAN_2_SState_Valid,
        APP_CAN_2_SState_Invalid,
        APP_CAN_2_SState_Unavailable,
        APP_CAN_2_SState_Error,
        APP_CAN_2_SState_nStates
    };

    extern int APP_CAN_2_SSigStates(APP_CAN_2_SState_nStates);

    /* Input Vector for CAN communication */
    typedef struct {
        int DeclareQuanta;
        int DeclareQuantaRcv;
        int DeclareQuantaState;
    };

    /* ECU ESS */
    struct tAPP_CAN_2_ESS {
        /* CAN Msg 3x001 (PGN001) */
    };

    struct tAPP_CAN_2_ESS_PGN001 {
        char Pos_temp_1;
        char Pos_temp_1_Raw;
        char Pos_temp_1_State;
        char Pos_temp_2;
        char Pos_temp_2_Raw;
        char Pos_temp_2_State;
        char Pos_temp_3;
    }
```

HCV Hybrid Commercial Vehicle – D6200.4, Rev_0  page 36 of 39
```c
char Pos_temp_3_State;
char Pos_temp_4;
char Pos_temp_4_State;
char Pos_temp_5;
char Pos_temp_5_RAW;
char Pos_temp_5_State;
short Temp_cool_1;
unsigned char Temp_cool_1_RAW;
char Temp_cool_1_State;
short Temp_cool_2;
unsigned char Temp_cool_2_RAW;
char Temp_cool_2_State;
short Temp_cool_3;
unsigned char Temp_cool_3_RAW;
char Temp_cool_3_State;
short Temp_cool_4;
unsigned char Temp_cool_4_RAW;
char Temp_cool_4_State;
short Temp_cool_5;
unsigned char Temp_cool_5_RAW;
char Temp_cool_5_State;

} PGN001;

/* CAN Msg 0x002 (PGN002) */
int PGN002_DLC;
struct tAPP_CAN_2_ESS_PGN002 {
  unsigned char ESS_Req;
  unsigned char ESS_Req_RAW;
  char ESS_Req_State;
} PGN002;

/* CAN Msg 0x003 (PGN003) */
int PGN003_DLC;
struct tAPP_CAN_2_ESS_PGN003 {
  float ESS_Act_Cur;
  unsigned short ESS_Act_Cur_RAW;
  char ESS_Act_Cur_State;
  float ESS_Act_Vol;
  short ESS_Act_Vol_RAW;
  char ESS_Act_Vol_State;
  float ESS_Char_Cur;
  short ESS_Char_Cur_RAW;
  char ESS_Char_Cur_State;
  float ESS_Dis_Cur;
  short ESS_Dis_Cur_RAW;
  char ESS_Dis_Cur_State;
} PGN003;
```
/* CAN Msg 0x004 (PGN004) */

int PGN004_DLC;

struct tAPP_CAN_2_ESS_PGN004 {
    float ESS_Act_Pow;
    short ESS_Act_Pow_Raw;
    char ESS_Act_Pow_State;
    float ESS_Act_Tor;
    unsigned short ESS_Act_Tor_Raw;
    char ESS_Act_Tor_State;
    float ESS_Max_Char_Pow;
    unsigned char ESS_Max_Char_Pow_Raw;
    char ESS_Max_Char_Pow_State;
    float ESS_Max_Dischar_Pow;
    unsigned char ESS_Max_Dischar_Pow_Raw;
    char ESS_Max_Dischar_Pow_State;
} PGN004;

/* CAN Msg 0x005 (PGN005) */

int PGN005_DLC;

struct tAPP_CAN_2_ESS_PGN005 {
    float ESS_Act_Flow;
    short ESS_Act_Flow_Raw;
    char ESS_Act_Flow_State;
    unsigned int ESS_Act_Pres;
    unsigned short ESS_Act_Pres_Raw;
    char ESS_Act_Pres_State;
    unsigned int ESS_Act_Speed;
    unsigned short ESS_Act_Speed_Raw;
    char ESS_Act_Speed_State;
} PGN005;

/* CAN Msg 0x006 (PGN006) */

int PGN006_DLC;

struct tAPP_CAN_2_ESS_PGN006 {
    short ESS_Avg_Systemp;
    char ESS_Avg_Systemp_Raw;
    char ESS_Avg_Systemp_State;
    short ESS_High_Systemp;
    char ESS_High_Systemp_Raw;
    char ESS_High_Systemp_State;
    short ESS_Low_Systemp;
    char ESS_Low_Systemp_Raw;
    char ESS_Low_Systemp_State;
    short ESS_Max_Systemp;
    char ESS_Max_Systemp_Raw;
    char ESS_Max_Systemp_State;
    short ESS_Min_Systemp;
    char ESS_Min_Systemp_Raw;
    char ESS_Min_Systemp_State;
    unsigned char ESS_Status;
    char ESS_Status_Raw;
    char ESS_Status_State;
} PGN006;
6. Conclusion

In the task 6230, *Implementation of extended communication protocol*, the standard communication protocol for hybrid powertrains developed in WP6320 is successfully integrated into the simulation platform AVL InMotion. This is achieved by the following procedure:

- Creating a database file based on the D6300.2.
- Parameterizing the MIO module. In this task, we used M51 module for developing the CAN interface with the simulation platform.
- Using the CANiogen, a database import tool used to import the control units into the simulation platform.
- Finally with the required modifications, integrated into the AVL InMotion/HIL.