

Bypassing ECU functions using XCP stimulation mechanism

by Oliver Kitt, Vector Informatik

THE XCP STIM MECHANISM IS A VERY EFFECTIVE METHOD FOR BYPASSING PARTS OF AN ECU ALGORITHM AND CAN SPEED UP SOFTWARE DEVELOPMENT CYCLES, DEPENDING ON THE DURATION OF THE ALTERNATIVE FLASH PROGRAMMING. THE REAL-TIME SUITABILITY OF THE SYSTEM COMPOSED OF XCP MASTER AND SLAVE DEPENDS ON THE OPERATING SYSTEMS USED.

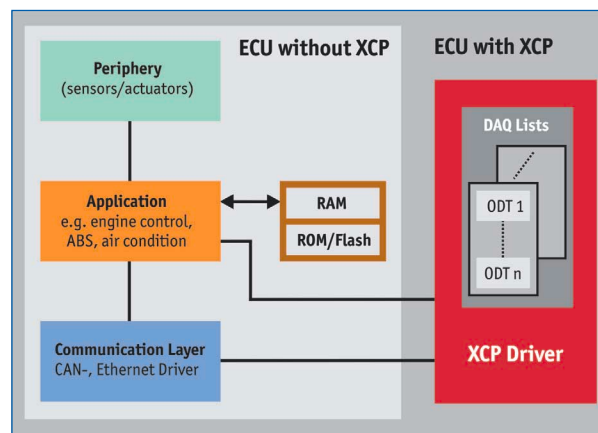


Figure 1. Interfaces of the XCP driver

■ Since most MC tools are WIN32-based, hard real-time requirements cannot be fulfilled. But soft real-time systems that only require a typical latency time and an upper limit of time-out incidents per unit time can use this mechanism. CANape Graph implemented this feature such that mean latency times of about 0.7 milliseconds can be achieved with greater than 99.99% reliability on modern PC hardware platforms with 10 ms cycle time. Compared to a hardware-based solution, bypassing based on XCP STIM is very inexpensive and only requires components that are already available in the ECU development and calibration environments.

Bypassing parts of the firmware code of an ECU (electronic control unit) is a useful mechanism in developing control algorithms in the area of automotive development. Since the algorithm can be developed and tested using modelling tools, the development cycle time can be reduced. One of the major problems was the time delay between input data retrieval and output data availability in the ECU when explicit UPLOAD and DOWNLOAD commands of the XCP protocol are used, because these commands require the response of the ECU.

XCP (a standard automotive calibration protocol defined by the ASAM organization) intro-

duces the so named XCP stimulation mechanism which does not require ECU response and therefore needs less than the half the time previously required, thereby achieving task cycle times well below 10 ms. The ECU acronym itself, i.e. electronic control unit, offers a clue to the typical architecture of such a device. For a device to be able to control a physical system, it needs sensors to get measurement data from its environment, a control algorithm that is usually executed periodically and actuators that influence the environment. The cycle time of the algorithm depends on the time dependent behaviour of the closed-loop control. Typical cycle times range from 10 ms to 1 s or more.

The algorithm itself may be split into several blocks that can be executed sequentially or in parallel. An example might be ECU software consisting of a low-pass input filter and the actual control algorithm. In such a scenario the control algorithm block would be a target for bypassing, especially if the relevant physical properties of the system are not completely known initially. If the algorithm must be changed, normally a complete development cycle has to be performed. This involves: building the software target; flashing to non-volatile memory; performing measurements in the physical environment as well as analyzing the

results and validating the algorithm change.

The basic features of XCP offer the functionalities of synchronous data acquisition/stimulation, read/write access and calibration data page initialization/switching. Furthermore XCP supports ECU flash programming. This is based on a memory-oriented view of the device. The following diagram gives an overview of how a calibration tool is interconnected with the ECU. If an ECU contains an XCP driver, the software architecture is normally organized in a way depicted in figure 1. This diagram illustrates that the XCP driver has only two important interfaces. One links the driver to the ECU communication layer. The second synchronizes the DAQ and STIM memory sampling mechanisms to the ECU application. The DAQ lists are internal XCP driver information and describe the ECU memory to be sampled and transmitted.

Important XCP protocol properties:

- ✓ It is a master-slave-based protocol. The measurement and calibration (MC) tool acts as master, the ECU as slave.
- ✓ Handshake communication (request-response) is used to initialize, configure and calibrate parameters for the ECU communication session.

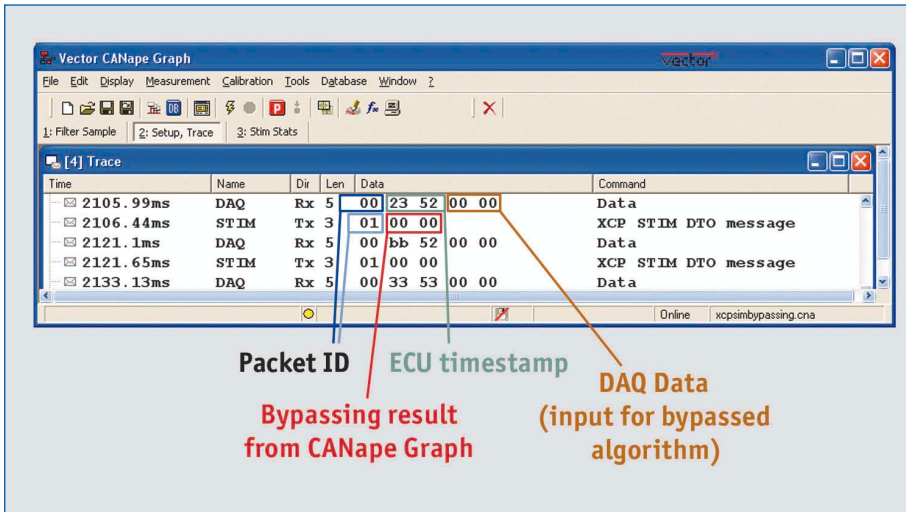


Figure 2. CANape Graph interprets the received data using the DAQ list information

```
void ecuCyclic()
{
  for (;;) {
    /* initial algorithm parts, e.g. low-pass filter for input data */

    if (ApplXcpFilterBypass() != BYPASSING_DATA_RECEIVED) {
      builtInAlgorithm();
    }

    /* remaining algorithm parts, actuator setting, etc. */
  }
}
```

Table 1. C-code fragment, illustrating the principle of task-synchronized bypassing

```
result_type ApplXcpFilterBypass( void ) {
  int status;
  bool timeoutValue;
  /* Data acquisition for bypass */
  /* Busy, wait for stimulation data if data acquisition is active */
  XcpStimEventStatus(AlgoBypassingStim, STIM_RESET_BUFFER);
  status = XcpEvent(AlgoBypassingDaq); /* Send data to XCP master */
  if (status & XCP_EVENT_DAQ) {
    t1 = XcpDaqGetTimestamp();
    timeoutState = false;
    while (!XcpStimEventStatus(AlgoBypassingStim, STIM_CHECK_BUFFER)) {

      /* Poll the XCP receive message queue */
      xcpMessageHandler();

      /* Code fragment continues on next page! */
      /* Check timeout */
      t2 = XcpDaqGetTimestamp();
      if ((t2-t1)>timeoutValue) {
        timeoutState = true;
        break;
      }
    }

    /* Use internal algorithm if no STIM data available */
    status = XcpEvent(AlgoBypassingStim); /* Write back the received bypassing
      data to the ECU memory */
    if ((status & (XCP_EVENT_STIM | XCP_EVENT_STIM_OVERRUN)) != XCP_EVENT_STIM) {
      return 0; // No STIM data available
    } else {
      return BYPASSING_DATA_RECEIVED;
    }
  }
}
```

Table 2. Sample implementation of the function which can be made part of the XCP framework

✓ Data acquisition mode (DAQ) provides transfer of application data. It is ensured that the data is consistent regarding the application cycle. The memory blocks are selected by XCP DAQ configuration commands before the measurement is started. The data flows from ECU to MC tool.

✓ Stimulation mode is similar to DAQ mode, but the data flows in the opposite direction. The data packets from the MC tool are buffered and saved back to ECU memory in a task-synchronized way.

Basic requirements of the XCP driver component:

- ✓ Access to the memory where the parameters and measurement signals of the algorithm are located.

- ✓ Access to the ECU's communication layer component for sending and receiving XCP protocol data.

To integrate the XCP driver component, the developer essentially needs to insert function calls at suitable points in the ECU code to XcpCommand() (the XCP command processor) and XcpEvent() (responsible for synchronous data acquisition and stimulation).

Without the XCP STIM mechanism, a minimal bypassing communication cycle for one input parameter and one output parameter consisted of 5 protocol messages. Only the first DAQ message was guaranteed to be synchronized to the ECU task cycle. The messages that carry the bypassing results are not synchronized to the task cycle. Using the XCP STIM mechanism, the number of messages needed for one bypassing cycle is reduced to 2. The ratio is even better when more parameters are involved since, in contrast to the SET_MTA/DNLOAD scenario, with DAQ and STIM messages more than one parameter can be inserted in a data frame. Furthermore, the sampling times for both DAQ data (input values) and STIM data (result of bypassed algorithm), are synchronized to the ECU cycle. The packet ID serves as a unique key for identifying the transmitted data. Both the XCP master and slave keep lists of memory blocks specified by address and size (DAQ lists), which are created during the DAQ configuration phase. The ECU's XCP driver samples and transmits the memory blocks according to the DAQ lists. The measurement and calibration tool interprets the received data using the DAQ list information.

The C-code fragment in table 1 illustrates the principle of task-synchronized bypassing. The original algorithm code only needs to be enhanced by the boldfaced lines. Table 2 shows a sample implementation of this function which can be made part of the XCP framework. In the configuration header file of the XCP driver, the constants "AlgoBypassingDaq" and "AlgoBypassingStim" are defined and configured for

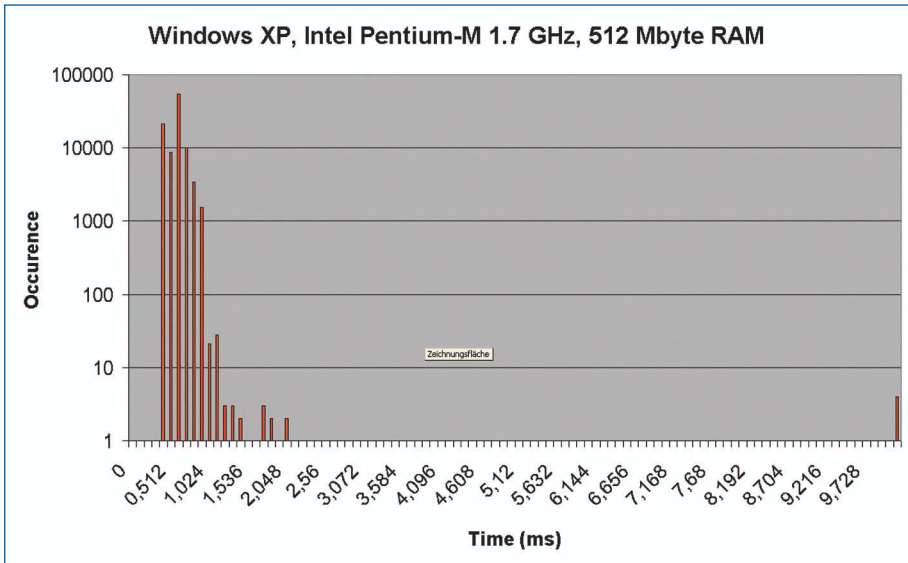


Figure 3. On this system CANape Graph was the only application with GUI elements. The virus scanner was disabled and the PC was not connected to a network. Only 4 bypassing cycles out of 100,000 were too late, i.e. the latency was greater than 8 ms. The worst case was a latency time of 10.9 ms.

Click-for-More

Interested in more information about bypassing ECU functions using XCP?

Visit our specific website with links to more details about:

- ▶ Technical details about XCP
- ▶ The development tool CANape Graph 5.6
- ▶ Measurement, Calibration and Diagnostics of ECUs
- ▶ Vector Informatik GmbH

Simply type-in Reader Service #: 621 at Embedded-Control-Europe.com/know-how

DAQ and STIM mode, respectively. The MC tool is responsible for calculating the external part of the algorithm. Therefore it must configure the input data required for the XCP event “AlgoBypassingDaq” and the bypassing results of “AlgoBypassingStim”. When the ECU sends the input data the calculations must be performed, and the bypass data must be sent back to the ECU. CANape Graph can use code from DLLs generated by MATLAB/Simulink thereby making the development and modification of controlling algorithms very fast and easy.

Vector Informatik offers two variants of the XCP driver component for ECUs. The basic variant is free of charge and provides online calibration and DAQ measurement mode. The additional features of the professional variant are XCP STIM mode, flashing and certain other features. The ECU software developer does not need to be concerned with the XCP protocol

implementation. All the developer has to do is include the XCP framework functions as shown above and perform some configuration steps, e.g. set the XCP DAQ processor buffer size.

Benefits of using bypassing based on XCP STIM:

- ✓ Shorter development cycle
- ✓ Faster feedback on algorithm changes
- ✓ More flexibility in the development process with intermediate step between pure software simulation (SIL) and hardware-in-the-loop
- ✓ No need for (time-consuming) flashing
- ✓ No need to publish the entire ECU source code
- ✓ No need for the cross-development platform in algorithm optimization
- ✓ No need for expensive real-time bypassing hardware.

The bypassing feature has been implemented in Vector’s MC tool CANape Graph and the professional variant of the Vector XCP driver component for ECUs. The current CANape Graph platform is a WIN32-based operating system on INTEL compatible hardware. This means that hard real-time cannot be supported due to operating system limitations.

The PC used for bypassing based on XCP STIM should have:

- ✓ No other applications or background processes running, e.g. virus detection programs
- ✓ Enough memory to avoid swapping
- ✓ A fast CPU
- ✓ No intensive user interaction while the bypassing application is running.

Measurements on several system configurations yielded the diagrams in figure 3. In all cases the peak of the latency time distributions was found to be at about 0.7 ms. The decadic logarithmic scale of the y-axis was chosen because a linear axis would properly distinguish between values approaching zero or exactly zero. ■