UNIGATE@CHIP: DK60 as a PROFIBUS slave

This application note describes how to use the PROFIBUS function kit from Deutschmann Automation GmbH & Co. KG on the DK60. This function kit consists of a PROFIBUS slave module (UNIGATE IC PROFIBUS) and the UNIGATE@CHIP adapter.

Please note, that the UNIGATE modules have the advantage of using a script language for customizing the serial port protocol. A tool for downloading customized scripts to the UNIGATE modules is available from Deutschmann Automation GmbH & Co. KG.

This application note is based on the contents of the accompanying ZIP-archive "AN_Unigate_Example_V1.0.zip". It only describes the communication between the IPC@CHIP and the PROFIBUS function kit based on the serial protocol explained in chapter “Concept”.

Background

The PROFIBUS function kit utilizes the connectors for UART0 (S40 & S42), UART1 (S41 & S43) and CAN1 (S38). For data communication between the PROFIBUS module and the IPC@CHIP® SC143 only UART0 and two PIOs (14 & 15) are used. The connection to UART1 is only for mechanical purposes and does not affect communication on this port in any way.

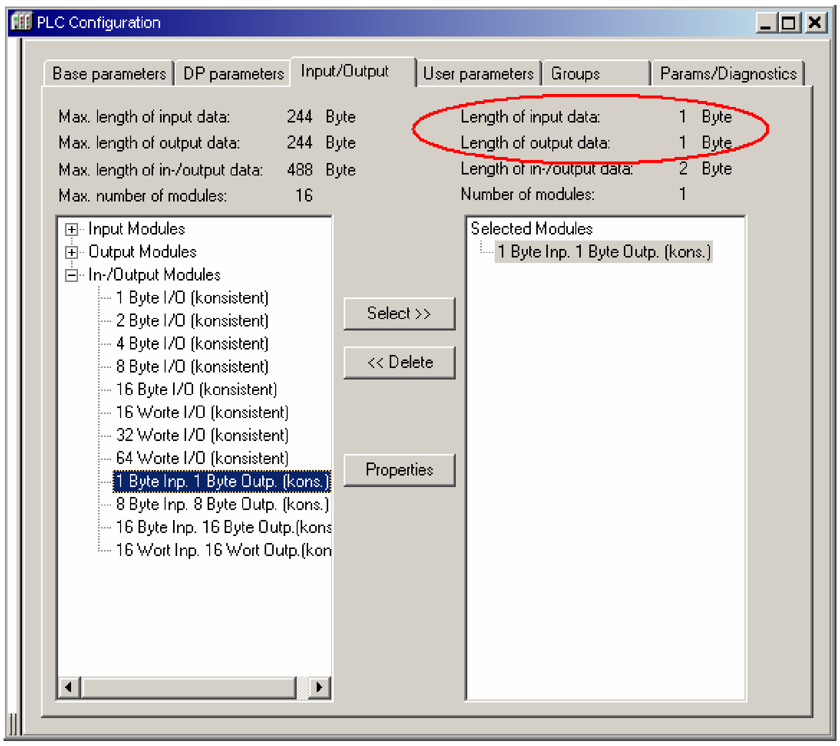
The application within the accompanying ZIP-archive shows an example of how to initialize the field bus module and read and write data from / to the bus. For getting this example application working it is necessary to have a PROFIBUS master system (PLC), which loops back the received data to the slave. The aim is to write the current state of the DIP switch on the DK60 to the field bus, get it read back from the master and show it on the LEDs on the DK60.

This is done in a strictly response-by-request basis which means every single request has to follow a response, before another request can be send. This should be taken into account when setting up a communication concept based on this application note, because a communication where telegrams could be send and received in an arbitrary order would need an own and more complex message handling system.

For use with the example in the ZIP-archive named above there is also a CoDeSys 2.3 project file included for use with the PROFIBUS master PLC, which does this loopback. If you want to use this for your own evaluation, please make sure to use the settings for the UNIGATE field bus module as shown below:

Beck IPC GmbH
Grüninger Weg 24
35415 Pohlheim-Garbenteich

E-Mail: info@beck-ipc.com
www.beck-ipc.com
A member of the Festo group



Max. length of input data: 244 Byte
Max. length of output data: 244 Byte
Max. length of in-/output data: 488 Byte
Max. number of modules: 16

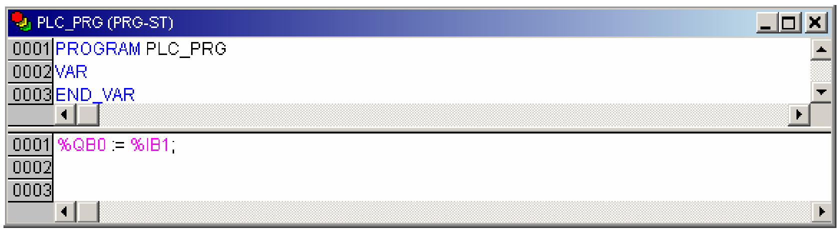
Length of input data: 1 Byte
Length of output data: 1 Byte
Length of in-/output data: 2 Byte
Number of modules: 1

Input Modules
Output Modules
In-/Output Modules
1 Byte I/O (konsistent)
2 Byte I/O (konsistent)
4 Byte I/O (konsistent)
8 Byte I/O (konsistent)
16 Byte I/O (konsistent)
16 Worte I/O (konsistent)
32 Worte I/O (konsistent)
64 Worte I/O (konsistent)
1 Byte Inp. 1 Byte Outp. (kons.)
8 Byte Inp. 8 Byte Outp. (kons.)
16 Byte Inp. 16 Byte Outp.(kons
16 Wort Inp. 16 Wort Outp.(kon

Selected Modules
1 Byte Inp. 1 Byte Outp. (kons.)

It is important to select the “1 Byte Inp. 1 Byte Outp. (kons.)” from the list on the left side to get the value of 1 Byte for “Length of input data” and “Length of output data” (marked red). These settings belong directly to the parameters “BusDataLenToIPC” and “BusDataLenFromIPC” discussed later in this application note. Please note, that also the station address of the UNIGATE field bus module has to match the setting in the CoDeSys PLC configuration. With this conditions met, the example application is ready for operation.

The PLC program itself is as simple as possible (structured text):



```
PROGRAM PLC_PRG
VAR
END_VAR
```

```
%QB0 := %IB1;
```

Concept

The current serial communication model between the UNIGATE field bus module and the IPC@CHIP® SC143 is also a master/slave system. The IPC@CHIP® SC143 is the master while the UNIGATE field bus module is the slave. In the defined protocol of the UNIGATE field bus module every instruction sequence has to start with a request from the master, followed by a response from the slave.

Based on the datagrams below, the protocol between the IPC@CHIP® and the UNIGATE field bus module will be explained.

```
+------+------+-------+-------+-…-+-------+---------+---------+
| LenH | LenL |  Type | Byte1 | … | ByteN | ~CheckH | ~CheckL |
+------+------+-------+-------+-…-+-------+---------+---------+
              \                                /
               \        data block            /
                --------------------------
```

Every telegram consists of a length information field (2 bytes), followed by a number of data bytes (N bytes data block) and is completed by a checksum (2 bytes). The length field states the number of bytes following including the 2 checksum bytes. The checksum is a byte-by-byte sum of all bytes excluding the 2 checksum bytes themselves and is finally inverted. Both words are transferred with the high byte first.

In dependence of the type field there are three possible telegram types. Every data block starts with a type identifier, followed by 3 control bytes and possibly some data bytes:

Type = 0 (reserved)

Type = 1 (request)

```
+---+--------+---------+---------+-----------+- … -+-----------+
| 1 |   ID   | Para-No | Command |  Req.Dat0 |  …  |  Req.DatN |
+---+--------+---------+---------+-----------+- … -+-----------+
```

Type = 2 (response)

```
+---+--------+---------+---------+-----------+- … -+-----------+
| 2 |   ID   | Para-No |  State  | Resp.Dat0 |  …  | Resp.DatN |
+---+--------+---------+---------+-----------+- … -+-----------+
```

The single fields are explained as follows:

ID:	An identification of the data records so that a response telegram can be assigned clearly to a request telegram. This identification number can be incremented simply e.g. with each new inquiry.
Para-No:	The indication, which parameter is to be read and/or written. A list of the supported parameters with indication of the data length and/or the construction of data is in the appendix. Valid values are: 1 = BusDataLenFromIPC 2 = BusDataLenToIPC 3 = BusStart 4 = BusData
Command:	The command of the request is defined here. Valid values are: 0 = read, 1 = write
State:	The result of the former request is stated here. Valid values are: 0 = ok, 1 = error (error code is in the byte "Resp.Dat0")

For a list of examples on how some datagrams look like, see the appendix.

In general, a few simple steps have to be followed for setting up the UNIGATE field bus slave module and to get into the data exchange mode:

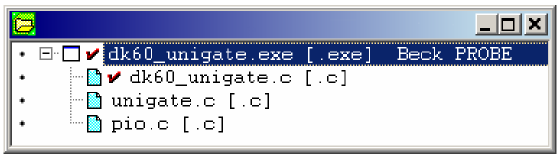
1.) Hardware reset and start in “normal mode”
2.) Parameter “BusDataLenFromIPC” has to be written
3.) Parameter “BusDataLenToIPC” has to be written
4.) Command “BusStart” has to be written

If all the parameter and command requests are successfully written without any error in the response (see field “state” above), the field bus module will be ready for transceiving data from / to bus.

Implementation

The software archive (ZIP-archive) contains the ready-to-run application "DK60_unigate.exe" and also the sources for this application example.

The sources consist of the application itself called "DK60_Unigate.c", the UNIGATE driver "unigate.c" and another module called "pio.c" which is used by the application to get access to the DIP switch and LEDs.
In addition there is a project file "DK60_Unigate.pdl" for the Paradigm C++ compiler Beck IPC edition, which can be used for quick evaluation and starting an own development. The structure of the project is shown below:



```
dk60_unigate.exe [.exe]  Beck PROBE
    dk60_unigate.c [.c]
    unigate.c [.c]
    pio.c [.c]
```

For getting access to the field bus module as described above, the driver "unigate.c" provides three major functions:

- Init_UNIGATE()
- WriteBus()
- ReadBus()

These functions are described briefly below. For more information, please refer to the source code of the driver "unigate.c". In addition, all main parameters used within the driver can be easily modified through corresponding defines in the header file of the driver ("unigate.h").

1. **unsigned char** Init_UNIGATE (**unsigned int** BusDataLenFromIPC,
   unsigned int BusDataLenToIPC)

   This function does the four-step initialization procedure as described above in addition to setup the UART0 port. The current settings for this communication port to the UNIGATE are:

• Baud rate:	9600
• Data bits:	8
• Stop bits:	1
• Parity:	None

   It returns either "0" if initialization procedure was ok or an error code "1" to "5" depending on the occurred error as defined in the header file "unigate.h". Please see the source code for more detailed information.

2. **unsigned char** WriteBus (**unsigned char** *SendData,
   unsigned int AnzahlByteToSend)

   This function writes a given number of bytes from a given buffer to the field bus module. It returns always "0". Please see the source code for more detailed information.

3. **unsigned char** ReadBus (**unsigned char** *RecvData,
   unsigned int *AnzahlByteToRecv)

   This function reads a given number of bytes from the UNIGATE into a given buffer. It returns either "0" if reading was ok or error code "3" if there was a timeout while reading. Please see the source code for more detailed information.

31.10.2007 B. Große

Appendix

Following is a list of possible example datagrams used for initialization and data exchange according to the protocol specification. For easier understanding the leading length and the final checksum fields were omitted.

BusDataLenFromIPC (write)

Request (from IPC):
```
+---+----+---+---+----------+----------+
| 1 | ID1 | 1 | 1 | DataLenH | DataLenL |
+---+----+---+---+----------+----------+
```
Response (from UNIGATE):
```
+---+----+---+---+
| 2 | ID1 | 1 | 0 |
+---+----+---+---+
```

BusDataLenFromIPC (read)

Request (from IPC):
```
+---+----+---+---+
| 1 | ID2 | 1 | 0 |
+---+----+---+---+
```
Response (from UNIGATE):
```
+---+----+---+---+----------+----------+
| 2 | ID2 | 1 | 0 | DataLenH | DataLenL |
+---+----+---+---+----------+----------+
```

BusDataLenToIPC (write)

Request (from IPC):
```
+---+----+---+---+----------+----------+
| 1 | ID3 | 2 | 1 | DataLenH | DataLenL |
+---+----+---+---+----------+----------+
```
Response (from UNIGATE):
```
+---+----+---+---+
| 2 | ID3 | 2 | 0 |
+---+----+---+---+
```

BusDataLenToIPC (read)

Request (from IPC):
```
+---+----+---+---+
| 1 | ID4 | 2 | 0 |
+---+----+---+---+
```
Response (from UNIGATE):
```
+---+----+---+---+----------+----------+
| 2 | ID4 | 2 | 0 | DataLenH | DataLenL |
+---+----+---+---+----------+----------+
```

BusStart (without error reported)

Request (from IPC):
```
+---+----+---+---+
| 1 | ID5 | 3 | 1 |
+---+----+---+---+
```
Response (from UNIGATE):
```
+---+----+---+---+
| 2 | ID5 | 3 | 0 |
+---+----+---+---+
```

BusStart (with error reported)

Request (from IPC):
```
+---+----+---+---+
| 1 | ID6 | 3 | 1 |
+---+----+---+---+
```
Response (from UNIGATE):
```
+---+----+---+---+-----------+
| 2 | ID6 | 3 | 1 | ErrorCode |
+---+----+---+---+-----------+
```

BusData (read)

Request (from IPC):
```
+---+----+---+---+
| 1 | ID7 | 4 | 0 |
+---+----+---+---+
```
Response (from UNIGATE):
```
+---+----+---+---+---------+-…-+---------+
| 2 | ID7 | 4 | 0 | BusDat0 | … | BusDatN |
+---+----+---+---+---------+-…-+---------+
```

BusData (write)

Request (from IPC):
```
+---+----+---+---+--------+-…-+--------+
| 1 | ID8 | 4 | 1 |BusDat0| … |BusDataN|
+---+----+---+---+--------+-…-+--------+
```
Response (from UNIGATE):
```
+---+----+---+---+
| 2 | ID8 | 4 | 0 |
+---+----+---+---+
```