VPC 3+

Software Manual

Revision 6.00

profichip®
The Clever Alternative

Table of Contents

Table of Contents

Notes:

Introduction 2

Profichip's **VPC3+** is a communication chip with processor interface for intelligent slave applications. VPC3+ handles the complete PROFIBUS-DP/DPV1 slave protocol independently and relieves the application processor of all time critical communication tasks. When VPC3+ carries out a DP communication it automatically sets up all DP-SAPs. All necessary timers and monitoring functions are integrated in the chip. Therefore almost the entire performance of the external controller is available for the application.

The UART converts the asynchronous serial PROFIBUS data stream into internal parallel data or vice-versa. Data is synchronized to system clock and processed by the microsequencer. The VPC3+ is capable of automatically identifying and controlling transmission rates up to 12 Mbit/s. The baudrate-generator derives the transmission clock from the system clock. The IDLE- and SYNI- (synchronization interval) timer observes the correct timing of the DP-telegrams according to the PROFIBUS-DP standards and especially controls the idle time before the next request telegram may occur. In case of timing violations the microsequencer will get a notification. The watchdog-timer observes the entire communication. If the watchdog is not re-triggered within the parameterized time (e.g. if the master application fails), the outputs are switched off automatically.

The 2/4 KByte on-chip communication RAM serves as an interface between the VPC3+ and the software/ application. Various telegram information is made available to the user in separate data buffers. Three input buffers and three outputs are provided for data communication. One buffer is always available for communication. Therefore, no resource problems can occur. For optimal diagnosis support, VPC3+ has two diagnosis buffers, that is, one diagnosis buffer is always assigned to VPC3+.

The microsequencer controls the entire process of PROFIBUS-DP/DPV1 protocol handling. Incoming data handed over by the UART is analyzed according to PROFIBUS-DP. If a service is recognized to be valid, user data is stored in the communication RAM and the interrupt controller generates an indication interrupt. Telegrams having frame errors (e.g. parity- or checksum errors) will be rejected. If the service of the telegram is recognized but its request does not make sense, a corresponding response telegram will be generated automatically. As a result user data will then be rejected to avoid unnecessary resource allocation within the microcontroller. The behavior of the microsequencer can be parameterized via mode- and parameter registers.

The Bus Interface Unit is a configurable synchronous/ asynchronous 8-bit interface for various microcontrollers / processors. The user can directly access the internal RAM or the Parameter Registers via the 11-bit address bus.

2 Introduction

2.1 Software package

The VPC3+ program package relieves the user of hardware register manipulations and memory calculations. It also provides a convenient „C“-interface to the DP and handles the completely statemachine for DPV1.

2.2 Software package PA007050

The software package consist of three application demos and is free available (http://www.profichip.com/products/overviewasics/dp-slave-vpc3-c/dp-v0-firmware/?L=5):

- EASY4711:
  - Simple slave with 2 byte of input data and 2 byte of output data
  - Diagnostic: No
- EASYADAC:
  - Modular slave (up to 244 modules)
  - Diagnostic: No
- DPV0AFFE:
  - Modular slave with 6 modules
  - Diagnostic: Yes
    - Modulstatus
    - Identifier related
    - Device related

2.3 Software package PA007062

- DPV1AFFE
  - Modular slave with 6 modules
  - Diagnostic: Yes
    - Modulstatus
    - Identifier related
  - DP-V1 functions:
    - Data set read
    - Data set write
    - I&M functions
    - Alarm
      - Processalarm
      - Diagnosticalarm

2.4 Structure of PA007062 / PA007050 software package

Directory/ Sub-Directory	File Name	Explanation
\DOC_DIR\	VPC3+CLF3_UMxyz.pdf VPC3+S_UMxyz.pdf VPC3+_SoftwareDescription_Vxyz.pdf PROFIBUS_Description.pdf Diagnosis.pdf GSD_Spec_2122_V51.pdf ProfileGuidelines-I&M_3502.pdf	Manual VPC3+CLF3 Manual VPC3+S Documentation of VPC3+ software Short PROFIBUS description Description of PROFIBUS diagnosis GSD-file description Description of DP-V1 I&M-functionality
\Customer\ \DPV0_DRV\		Directory of DP-V0 functions:
\DPV1_DRV\		Directory of DP-V1 functions
\GSD		GSD-file
	Main.c Platform.h DpCfg.h DpAppl.c DpAppl.h DpPrm.c DpCfg.c DpDiag.c DpV1.c DpIm.c	Main function call Microcontroller settings, data types Configuration file for VPC3+ Application demo Structures of application demo Handling of PROFIBUS Parameter-telegram Handling of PROFIBUS Configuration-telegram Handling of PROFIBUS diagnostics Handling of DP-V1 services Handling of I&M functionality
\EvalBoard\ \DPV0_DRV\		PROFICHIP-Evaluation-board Directory of DP-V0 functions:
\DPV1_DRV\		Directory of DP-V1 functions
\GSD		GSD-file
\Ext\	 startup.asm regsnd1.h extsnd1.h DpDebug.h DpDebug.c Lcd.h Lcd.c Serio.h Serio.c Twi.h Twi.c	Directory of ATMEL 8051 microcontroller Start routine Defines T8xC51SND1 components Extension to regsnd1.h Header file of debug functions Debug functions Defines for LCD-display Functions for LCD-display Defines for serial functions Serial functions Defines IIC Functions for IIC
	Main.c Platform.h DpCfg.h DpAppl.c DpAppl.h DpPrm.c DpCfg.c DpDiag.c DpV1.c DpIm.c	Main function call Microcontroller settings, data types Configuration file for VPC3+ Application demo Structures of application demo Handling of PROFIBUS Parameter-telegram Handling of PROFIBUS Configuration-telegram Handling of PROFIBUS diagnostics Handling of DP-V1 services Handling of I&M functionality
\Examples\		

\DPV0AFFE\		DP-V0 example with 6 modules and diagnostics
\EASY4711\		DP-V0 example with two byte of input and two byte of output data, no diagnostic
\EASYADAC\		DP-V0 example with 244 byte of input and 244 byte of output data, no diagnostic
	platform_cust.h	Microcontroller settings, data types for customer project (parallel mode)
	platform_eva.h	Microcontroller settings, data types for profichip evaluation board (parallel mode)
	platform_cust_ser.h	Microcontroller settings, data types for customer project (serial mode)
	platform_eva_ser.h	Microcontroller settings, data types for profichip evaluation board (serial mode)
	DpCfg_isr.h	Configuration file for VPC3+ (interrupt driven)
	DpCfg_poll.h	Configuration file for VPC3+ (polling mode)
	DpAppl.c	Application demo
	DpAppl.h	Structures of application demo
	DpPrm.c	Handling of PROFIBUS Parameter-telegram
	DpCfg.c	Handling of PROFIBUS Configuration-telegram
	DpDiag.c	Handling of PROFIBUS diagnostics
\DPV1_AFFE\		DP-V1 example with 6 modules, alarms and I&M functionality
	platform_cust.h	Microcontroller settings, data types for customer project (parallel mode)
	platform_eva.h	Microcontroller settings, data types for profichip evaluation board (parallel mode)
	platform_cust_ser.h	Microcontroller settings, data types for customer project (serial mode)
	platform_eva_ser.h	Microcontroller settings, data types for profichip evaluation board (serial mode)
	DpCfg_isr.h	Configuration file for VPC3+ (interrupt driven)
	DpCfg_poll.h	Configuration file for VPC3+ (polling mode)
	DpAppl.c	Application demo
	DpAppl.h	Structures of application demo
	DpPrm.c	Handling of PROFIBUS Parameter-telegram
	DpCfg.c	Handling of PROFIBUS Configuration-telegram
	DpDiag.c	Handling of PROFIBUS diagnostics
	DpV1.c	Handling of DP-V1 services
	DpIm.c	Handling of I&M functionality

Figure 2-1: Content of the directory

Subdirectory DPV0_DRV:

Directory/ Sub-Directory	File Name	Explanation
\DPV0_DRV\		Directory of DP-V0 functions:
	dp_if.h	Defines,structures and macros of VPC3+
	dp_if.c	Basic functions for VPC3+
	dp_isr.c	Interrupt, poll routine for VPC3+
	dpl_list.h	Macros for double pointered list
	dp_inc.h	Header include hierarchy

Figure 2-2: Content of the directory

Subdirectory DPV1_DRV:

Directory/ Sub-Directory	File Name	Explanation
\DPV1_DRV\		Directory of DP-V1 functions
	dp_fdl.c	Basic fdl-driver
	dp_msac1.c	Driver for acyclic class1 messages
	dp_msac2.c	Driver for acyclic class2 messages

Figure 2-3: Content of the directory

⚠ **Only in software package PA007062!**

2.5 PROFIBUS DP

PROFIBUS DP was developed for fast, cyclical input and output traffic, with the application emphasis being on the field level. The data traffic in the master-slave method is standardized in the EN 50 170; simple as well as intelligent field devices can be interconnected.

2.6 PROFIBUS DPV1

In many cases, cyclical data exchange according to EN 50 170 is no longer sufficient today for more complex devices. For that reason, it became necessary to define acyclical services as PROFIBUS extensions. These extensions have been defined in the technical guideline of the Profibus Trade Organization (PNO). Field devices can use these services optionally.

Some intelligent field devices need the following:

- Gapless reparameterizaton of the application process
- Free access to any parameters in a field device
- Transmission of data of variable length

For the sake of simplicity, these services may be transferred to the field devices acyclically, and run parallel to the cyclical data traffic. Standard field devices and devices that need these optional extensions can be operated jointly on the same bus with the functionality that is supported respectively.

The following services are specified as optional services between Class 1 masters and a slave as MSAC_C1 (Master-Slave acyclic communication Class 1):

- Read the data set of a slave (DS_Read)
- Write the data set of a slave (DS_Write)
- Alarm acknowledgement (Alarm_Ack)

The following services are specified as optional services between Class 2 masters and a slave as MSAC_C2 (Master-Slave acyclic communication Class 2):

- Initiate
- Read the data set of a slave (DS_Read)
- Write the data set of a slave (DS_Write)
- Transport (Data_Transport)
- Abort

2.7 PROFIBUS DPV2

PROFIBUS DPV2 adds a number of new features to the existing protocol stack to provide for slave-to-slave communications, time synchronization and an isochronal bus cycle. PROFIBUS now has the capability to provide for both acyclic communications via DPV1 and also slave-to-slave communications via DPV2, creating new application areas particularly in motion control (PROFIdrive) and safety (PROFIsafe).

The new functions of DPV2 include the establishment of an isochronous bus cycle (occurring in equal intervals of time) which allows closed-loop control between master and slave devices. With clock deviations of less than 1 microsecond, high-precision positioning can be realized. Slave-to-slave communication decreases the cycle time between master and slave and reduces the response time by 60 – 90 %.

Time synchronization provides a time stamp function so that events can be followed or tracked precisely, easing the registration/detection of timed events and facilitating the diagnosis of malfunctions and the correct chronological planning of actions. With the new upload and download functionality, any size data packet can be loaded into a field device with one command. Program updates or exchange of devices can be carried out without the troublesome and complicated loading processes, which are different for every manufacturer. The transfer into non-volatile storage or the start/stop command for the field device are also supported.

2.8 How a PROFIBUS DP Slave Works

For clarification, the state machine of a DP slave is briefly described below. The state machine regulates the defined, standard-conforming response of a DP slave in the possible situations. A detailed description is provided in the corresponding documents.

The sequence, in principle, of this state machine is helpful to understanding the firmware sequence. The details are provided in the standard EN 50 170, and the Technical Guidelines. The MSAC_C2 connection is not interfaced with the cyclical state machine. For that reason, the Class 2 connection is established and cancelled via Initiate and Abort; it is monitored by an idle mechanism.

Power_On

A Set_Slave_Address message is only accepted in the mode Power_On.

Wait_Prm

After power-up, the slave expects a parameter assignment message. All other types of messages are rejected or are not edited. Data exchange is not yet possible. In the parameter message, at least the information specified by the standard -such as the PNO Ident number, sync/freeze capability, etc.- is stored. In addition, user-specific parameter data is possible. Only the application specifies the meaning of this data. In the configuration of the master interface, certain bits are set, for example, in order to indicate a desired measuring range. The firmware makes this user-specific data available to the application program; the application program evaluates the data; it can accept it or reject it (for example, the desired measuring range can’t be set, and therefore meaningful operation is not possible).

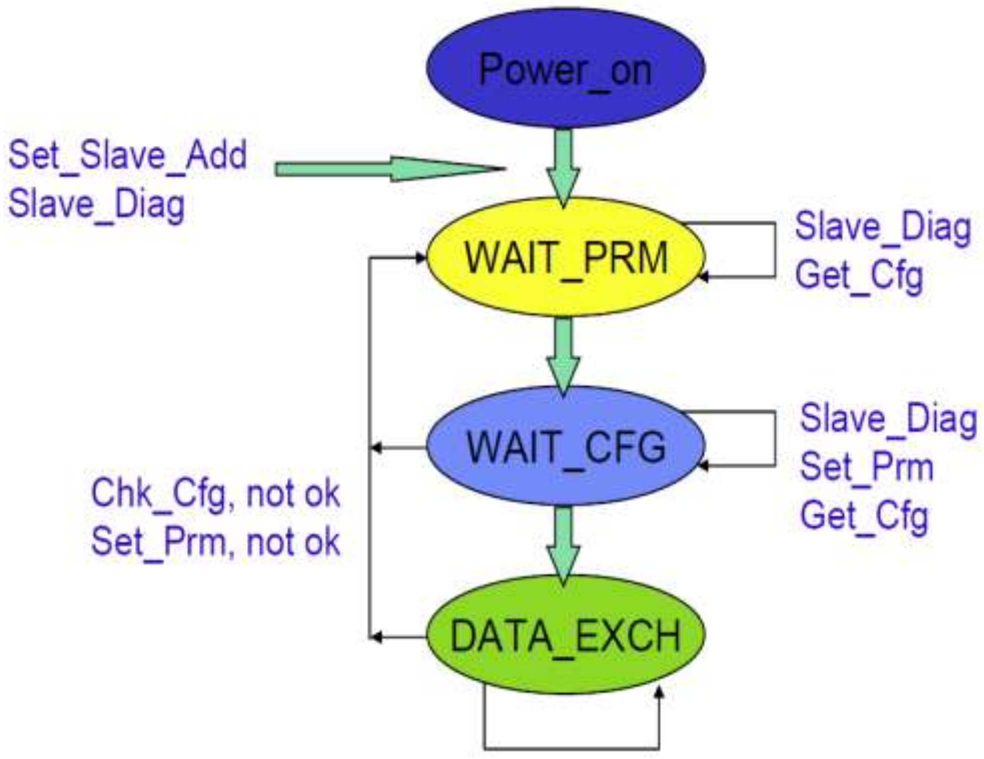


Figure 2-4 : State Machine

Wait_Cfg

The configuration message specifies the number of input and output bytes. The master informs the slave of how many bytes I/O are being transmitted. The application is informed of the requested configuration for checking. This check results either in a right, a wrong, or an adaptable configuration. If the slave wants to adapt to the desired configuration, a new user data length has to be calculated from the configuration bytes (for example, 4 bytes inputs predefined; only 3 bytes utilized). The application has to decide whether this adaptability is useful. In addition, is possible for each master to poll the configuration of any slave.

Data_Exchange

If the firmware as well as the application have accepted the parameter assignment and the configuration as correct, the slave transitions to the mode Data_Exchange; that is, it exchanges user data with the master.

Diagnosis

Via the diagnosis, the slave informs the master of its current mode. It consists at least of the information, specified in the standard, in the first six octets, such as the status of the state machine. The user can supplement this information (user diagnosis) with process-specific information (for example, wire break). On the slave's initiative, the diagnosis can be transmitted as error message and as status message. In addition to three defined bits, the user also influences the application-specific diagnostic data. However, any Master (not only the assigned master) can poll the current diagnostic information.

Read_Inputs, Read_Outputs

Every master can poll the current states of the inputs and outputs of any slave (in the Data_Exchange mode). The ASIC and the firmware process this function autonomously.

Watchdog

Along with the parameter message, the slave also receives a watchdog value. If this watchdog is not retriggered through the bus traffic, the state machine transitions to the "safe" state Wait_Prm.

MSAC_C1 (Master Slave Acyclic Communication of Class 1)

The MSAC_C1 services are used for communicating with a Class 1 master (typically, PLC). These services are available after the master has parameterized and configured the slave; that is, if the slave is in the DataEx mode.

The following services are available:

- DS_READ read data set
- DS_WRITE write data set
- ALARM_ACK acknowledge alarm

Since these services are permanently coupled to the configuring master C1 and since they run via permanently defined SAPs (50/51), the INITIATE/ABORT/IDLE mechanism is not required. If there is a fault in acyclically data transfer, cyclical communication is influenced also, and vice versa.

MSAC_C2 (Master Slave Acyclical Communication of Class 2)

The MSAC_C2 services are used for communicating with a Class 2 master (typically PC/PG as parameter assignment tool). These services are available immediately after initialization. Since these services are used dynamically, the master has to initiate the establishment of the connection with INITIATE so that the slave can adapt itself to it, and reject the services if necessary (insufficient memory, or no free SAP, …). While the connection is established, both sides monitor the connection with IDLE messages. If the connection is no longer needed, the master or the slave can de-establish the connection by transmitting an ABORT PDU. The IDLE messages are processed within the firmware.

The following services are available:

- INITIATE establishment of connection
- READ read data set
- WRITE write data set
- DATA_TRANSPORT general transport service
- ABORT Cancellation of connection

2.9 Helpful documents

Title	Available Language	Link
Book:“**The New Rapid Way to PROFIBUS DP**” Describing PROFIBUS from DP-V0 to DP-V2	German, English	http://www.profibus.com/press-media/pi-books
PDF: “**PROFIBUS System Description**”	German, English, Chinese, Japanese, French, Polish, Russian	http://www.profibus.com/nc/downloads/downloads/profibus-technology-and-application-system-description/display/
Book: “**PROFIBUS Manual**” A collection of explaining PROFIBUS networks	German, English	http://www.profibus.felser.ch/

2.10 Helpful links

PROFIBUS international	http://www.profibus.com
PROFIBUS Prof. Max Felser	http://www.profibus.felser.ch/

Notes:

Initialization 3

3.1 Configuration of platform.h

In order to support the different storage models with some processors, the memory accesses are to be provided with attributes. These attributes depends on the compiler settings.

3.1.1 Settings of platform.h

#define VPC3_SERIAL_MODE	0: VPC3+ works with parallel data bus 1: VPC3+S works with SPI, IIC or PORT_PIN-mode	
#define MOTOROLA_MODE	0: the VPC3+ configuration pin MOT/XINT is set to parallel interface INTEL format 1:the VPC3+ configuration pin MOT/XINT is set to parallel interface MOTOROLA format	
#define TRUE	1	
#define FALSE	!(TRUE)	
#define BOOL	unsigned char	1 bit basic type
#define uint8_t	unsigned char	8 bit basic type
#define uint16_t	unsigned int	16 bit basic type
#define uint32_t	unsigned long	32 bit basic type
#define PTR_ATTR	xdata	Memory model attribute of VPC3+. (xdata, near, far, huge ...)
#define VPC3_PTR	PTR_ATTR *	VPC3 Pointer attribut
#define VPC3_ADR	uint16_t	Attribute of the asic address. (uint16_t, uint32_t)
#define VPC3_UNSIGNED8_PTR	uint8_t PTR_ATTR *	Pointer of byte to VPC3+
#define NULL_PTR	(void VPC3_PTR)0	Zero-pointer
#define MEM_ATTR	xdata	Memory model attribute of local memory.(xdata, near, far, huge ...)
#define MEM_PTR	MEM_PTR_ATTR *	Pointer attribut of local memory
#define MEM_UNSIGNED8_PTR	uint8_t MEM_PTR_ATTR *	Pointer of byte to local memory
#define ROM_CONST__	code	Attribute of constant variables.
#define _PACKED_		Feed a keyword for packing structures.
#define LITTLE_ENDIAN	0	0: deactivated, 1:activated
#define BIG_ENDIAN	1	0: deactivated, 1:activated
#define VPC3_ASIC_ADDRESS	((unsigned char *)0x28000)	VPC3+ address

Figure 3-1 : platform.h settings

3.1.2 Example 8051, KEIL compiler

```
#define TRUE                        1
#define FALSE                       !(TRUE)

#define BOOL                        unsigned char
#define uint8_t                     unsigned char
#define uint16_t                    unsigned int
#define uint32_t                    unsigned long

#define PTR_ATTR                    xdata
#define VPC3_PTR                    PTR_ATTR *
#define VPC3_ADR                    uint16_t
#define VPC3_UNSIGNED8_PTR          uint8_t PTR_ATTR *
#define NULL_PTR                    (void VPC3_PTR)0

#define MEM_ATTR                    xdata
#define MEM_PTR                     MEM_PTR_ATTR *
#define MEM_UNSIGNED8_PTR           uint8_t MEM_PTR_ATTR *

#define ROM_CONST__                 code

#define _PACKED_

#define LITTLE_ENDIAN               0
#define BIG_ENDIAN                  1

#define VPC3_ASIC_ADDRESS           ((unsigned char *)0x28000)
```

Figure 3-2 : Example 8051

3.1.3 Example 80165, TASKING compiler

```
#define TRUE                        1
#define FALSE                       !(TRUE)

#define BOOL                        unsigned char
#define uint8_t                     unsigned char
#define uint16_t                    unsigned int
#define uint32_t                    unsigned long

#define PTR_ATTR                    far
#define VPC3_PTR                    PTR_ATTR *
#define VPC3_ADR                    uint32_t
#define VPC3_UNSIGNED8_PTR          uint8_t PTR_ATTR *
#define NULL_PTR                    (void VPC3_PTR)0

#define MEM_ATTR                    near
#define MEM_PTR                     MEM_PTR_ATTR *
#define MEM_UNSIGNED8_PTR           uint8_t MEM_PTR_ATTR *

#define ROM_CONST__                 const

#define _PACKED_

#define LITTLE_ENDIAN               1
#define BIG_ENDIAN                  0

#define VPC3_ASIC_ADDRESS           0x18000
```

Figure 3-3 : Example 80165

3.1.4 Example AtMega128

```
#define TRUE                       1
#define FALSE                      !(TRUE)

#define BOOL                       unsigned char
#define uint8_t                    unsigned char
#define uint16_t                   unsigned int
#define uint32_t                   unsigned long

#define PTR_ATTR
#define VPC3_PTR                   PTR_ATTR *
#define VPC3_ADR                   uint16_t
#define VPC3_UNSIGNED8_PTR         uint8_t PTR_ATTR *
#define NULL_PTR                   (void VPC3_PTR)0

#define MEM_ATTR
#define MEM_PTR                    MEM_PTR_ATTR *
#define MEM_UNSIGNED8_PTR          uint8_t MEM_PTR_ATTR *

#define ROM_CONST__                const

#define _PACKED_

#define LITTLE_ENDIAN              0
#define BIG_ENDIAN                 1

#define VPC3_ASIC_ADDRESS          ((unsigned char *)0x8000)
```

Figure 3-4 : Example AtMega128

3.1.5 Example ARM9, GNU compiler

```
#define TRUE                       1
#define FALSE                      !(TRUE)

#define BOOL                       unsigned char
#define uint8_t                    unsigned char
#define uint16_t                   unsigned short
#define uint32_t                   unsigned long

#define PTR_ATTR
#define VPC3_PTR                   PTR_ATTR *
#define VPC3_ADR                   uint32_t
#define VPC3_UNSIGNED8_PTR         uint8_t PTR_ATTR *
#define NULL_PTR                   (void VPC3_PTR)0

#define MEM_ATTR
#define MEM_PTR                    MEM_PTR_ATTR *
#define MEM_UNSIGNED8_PTR          uint8_t MEM_PTR_ATTR *

#define ROM_CONST__                const

#define _PACKED_                   __attribute__ ( ( packed ) )

#define LITTLE_ENDIAN              1
#define BIG_ENDIAN                 0

#define VPC3_ASIC_ADDRESS          0x40000000
```

Figure 3-5 : Example ARM9

3.2 Configuration of DpCfg.h

The different PROFIBUS services and their parameter defines the user in the file “DpCfg.h”.

3.2.1 Profibus Services

The user connects the different services via #define in “DpCfg.h”, so that the program code is adapted to the required services respectively.

Service	
#define DP_MSAC_C1	1: Activation of the functionality for the expansion services of the Class 1 master. 0: not activated
#define DP_MSAC_C2	1: Activation of the functionality for the expansion services of the Class 2 master. 0: not activated
#define DP_ALARM	1: Activation of the functionality for the expansion services of the alarm mode. 0: not activated
#define DPV1_IM_SUPP	1: Activation of the functionality for the expansion services of the I&M functionality. 0: not activated
#define DP_SUBSCRIBER	1: Activation of the functionality for the expansion services of the DXB subscriber mode. 0: not activated
#define DP_TIMESTAMP	1: Activation of the functionality for the expansion services of the timestamp mode. 0: not activated
#define DP_ISOCHRONOUS_MODE	1: Activation of the functionality for the expansion services of the isochronous mode. 0: not activated

Figure 3-6 : PROFIBUS Services

3.2.2 General Slave Parameter

General Slave Parameter		
#define DP_ADDR	uint8_t	PROFIBUS DP-Slave Address (1..125)
#define IDENT_NR	uint16_t	PROFIBUS Ident Number
#define USER_WD	uint16_t	User Watchdog

Figure 3-7 : General Slave Parameter

The **ident number** is used for clearly identifying the slave and is included with each diagnostic message from the slave to the master. Request your own number (www.profibus.com).

The **user watchdog** provides that, if the connected microcontroller fails, the VPC3+ leaves the Data Exchange mode after a defined number of data-exchange messages. As long as the microcontroller doesn't crash, it has to retrigger this watchdog.

3.2.3 Buffer Initialization

The user must enter the length of the exchange buffers for the different messages in the VPC3+ structure. These lengths determine the data buffers setup in the ASIC, and therefore are dependent in total sum on the ASIC memory.

Buffer		
#define DIN_BUFSIZE	uint8_t	Length of the DIn Buffer (0..244 Bytes)
#define DOUT_BUFSIZE	uint8_t	Length of the DOut Buffer (0..244 Bytes)
#define PRM_BUFSIZE	uint8_t	Length of the Parameter Buffer (7..244 Bytes)
#define DIAG_BUFSIZE	uint8_t	Length of the Diagnosis Buffer (6..244 Bytes)
#define CFG_BUFSIZE	uint8_t	Length of the Configuration Buffer (1..244 Bytes)
#define SSA_BUFSIZE	uint8_t	Length of the Input Data in the Set_Slave_Address-Buffer 0 and 4..244 Bytes

Figure 3-8 : Buffer Initialization

Specifying length 0 for the Set-Slave-Address buffer disables this utility.

3.2.4 Settings for I&M functionality

I&M		
#define MANUFACTURER_ID	uint16_t	Manufacturer ID, order from www.profibus.com
#define IM1_SUPP		0: service is deactivated 1: service is activated
#define IM2_SUPP		0: service is deactivated 1: service is activated
#define IM3_SUPP		0: service is deactivated 1: service is activated
#define IM4_SUPP		0: service is deactivated 1: service is activated

Figure 3-9 : Settings for I&M

3.2.5 Settings for MSAC_C1

Settings for MSAC_C1 Service		
#define C1_LEN	uint8_t	Length of MSAC_C1 Data (4..244 Bytes)

Figure 3-10 : Settings for MSAC_C1

3.2.6 Settings for MSAC_C1 Alarm

Settings for MSAC_C1 Alarm	
#define DP_ALARM_OVER_SAP50	1: The master handles the Alarm Acknowledge over SAP 50 0: The master handles the Alarm Acknowledge over SAP 51

Figure 3-11 : Settings for MSAC_C1_Alarm

3.2.7 Settings for MSAC_C2 Service

Settings for MSAC_C2 Service		
#define DP_MSAC_C2_Time		Enables time control for C2 services
#define C2_NUM_SAPS	uint8_t	Number of SAPs that the firmware makes available for MSAC_C2 Connections
#define C2_LEN	uint8_t	MSAC_C2 PDU length of the C2-SAP (20...244)
#define	uint8_t	= 0x01 (MSAC_C2_READ and

C2_FEATURES_SUPPORTED_1		MSAC_C2_WRITE supported)
#define C2_FEATURES_SUPPORTED_2	uint8_t	= 0x00
#define C2_PROFILE_FEATURES_1	uint8_t	Profile or vendor specific
#define C2_PROFILE_FEATURES_2	uint8_t	Profile or vendor specific
#define C2_PROFILE_NUMBER	uint16_t	Profile or vendor specific

Figure 3-12 : Settings for MSAC_C2 Service

3.2.8 Settings for Isochron Mode

Settings for Isochron Mode		
#define SYNCH_PULSEWIDTH	uint8_t	Width of synch pulse in 1/12µs

Figure 3-13 : Settings for Isochron Mode

3.2.9 Settings for DXB Subscriber Mode

Settings for DXB Subscriber Mode		
#define MAX_LINK_SUPPORTED	uint8_t	Number of Links
#define MAX_DATA_PER_LINK	uint8_t	maximal Number of Data per Link

Figure 3-14 : Settings for DXB Subscriber Mode

⚠ **PROFICHIPS asics support only one DxB connection!**

3.2.10 Set Hardware Mode

Next, the user has to configure the hardware function and telegram processing in the Mode Register 0 and 2 of the VPC3+:

Changes in Mode Register 0 and 2 are only allowed during start-up, when the VPC3+ is 'offline'.

Settings for Hardware Mode		
#define INIT_VPC3_MODE_REG_L	uint8_t	Mode Register 0 (LowByte)
#define INIT_VPC3_MODE_REG_H	uint8_t	Mode Register 0 (HighByte)
#define INIT_VPC3_MODE_REG_2	uint8_t	Mode Register 2
#define INIT_VPC3_MODE_REG_3	uint8_t	Mode Register 3
#define INIT_VPC3_MODE_IND_L	uint8_t	Interrupt Indication (LowByte)
#define INIT_VPC3_MODE_IND_H	uint8_t	Interrupt Indication (HighByte)

Figure 3-15 : Settings for Hardware Mode

ModeRegister0

Address	Bit Position								Designation
	7	6	5	4	3	2	1	0	
06H (Intel)	Freeze_ Supported	Sync_ Supported	Early_Rdy	Int_Pol	MinTSDR	WD_Base	Dis_Stop_ Control	Dis_Start_ Control	Mode Reg 0 7 .. 0 See below for coding

	Mode Register 0, Low-Byte, Address 06H (Intel):
Bit 7	**Freeze_Supported**: Freeze_Mode support 0 = Freeze_Mode is not supported. 1 = Freeze_Mode is supported
Bit 6	**Sync_Supported:** Sync_Mode support 0 = Sync_Mode is not supported. 1 = Sync_Mode is supported.
Bit 5	**Early_Rdy**: Early Ready 0 = Normal Ready: Ready is generated when data is valid (read) or when data has been accepted (write). 1 = Ready is generated one clock pulse earlier.
Bit 4	**INT_Pol:** Interrupt Polarity 0 = The interrupt output is low-active. 1 = The interrupt output is high-active.
Bit 3	**MinTSDR:** Default setting for the MinTSDR after reset for DP operation or combi operation. 0 = Pure DP operation (default configuration!)
Bit 2	**WD_Base:** Watchdog Time Base 0 = Watchdog time base is 10 ms (default state) 1 = Watchdog time base is 1 ms
Bit 1	**Dis_Stop_Control:** Disable Stopbit Control 0 = Stop bit monitoring is enabled. 1 = Stop bit monitoring is switched off A Set-Param telegram overwrites this memory cell in the DP mode. (Refer to the user specific data.)
Bit 0	**Dis_Start_Control:** Disable Startbit Control 0 = Monitoring the following start bit is enabled. 1 = Monitoring the following start bit is switched off A Set-Param telegram overwrites this memory cell in the DP mode. (Refer to the user specific data.)

Figure 3-16 : Coding of Mode Register 0, Low-Byte

Address	Bit Position								Designation
	15	14	13	12	11	10	9	8	
07H (Intel)	Reserved	PrmCmd_ Supported	Spec_Clear_ Mode	Spec_Prm_ Buf_Mode	Set_Ext_Prm _Supported	User_Time_ Base	EOI_Time_ Base	DP_Mode	Mode Reg 0 15 .. 8 See below for coding

	Mode Register 0, High-Byte, Address 07H (Intel):
Bit 15	**Reserved**
Bit 14	**PrmCmd_Supported**: PrmCmd support for redundancy 0 = PrmCmd is not supported. 1 = PrmCmd is supported
Bit 13	**Spec_Clear_Mode:** Special Clear Mode (Fail Safe Mode) 0 = No special clear mode. 1 = Special clear mode. VPC3+ will accept data telegrams with data unit = 0
Bit 12	**Spec_Prm_Buf_Mode:** Special Parameter Buffer Mode 0 = No special parameter buffer. 1 = Special parameter buffer mode. Parameterization data will be stored directly in the special parameter buffer.
Bit 11	**Set_Ext_Prm_Supported:** Set_Ext_Prm telegram support 0 = SAP 53 is deactivated 1 = SAP 53 is activated
Bit 10	***)User_Time_Base:** Timebase of the cyclical User_Time_Clock-Interrupt 0 = The User_Time_Clock-Interrupt occurs every 1 ms. 1 = The User_Time_Clock-Interrupt occurs every 10 ms. (mandatory DPV1)
Bit 9	**EOI_Time_Base:** End-of-Interrupt Timebase 0 = The interrupt inactive time is at least 1 µsec long. 1 = The interrupt inactive time is at least 1 ms long
Bit 8	**DP_Mode:** DP_Mode enable 0 = DP_Mode is disabled. 1 = DP_Mode is enabled. VPC3+ sets up all DP_SAPs (default configuration!)

Figure 3-17 : Coding of Mode Register 0, High-Byte

*) The User_Time_Clock is a timer that is used for the timeouts of the MSAC_C2 connection. It generates a timer tick of 1ms or 10 ms that causes an interrupt if enabled. **The timer has to be set to 10ms if DP_MSAC_C2 is defined!** However, the user can attach himself to the timer interrupt routine for his own purposes. If the macro DP_MSAC_C2 is not defined, the timer is freely available.

ModeRegister2

Address	Bit Position								Designation
	7	6	5	4	3	2	1	0	
	0	0	0	0	0	0	0	1	Reset Value
0CH	4kB_Mode	No_Check_ Prm_Reserved	SYNC_Pol	SYNC_Ena	DX_Int_Port	DX_Int_Mode	No_Check_ GC_Reserved	New_GC_ Int_Mode	Mode Reg 2 7 .. 0

	Mode Register 2, Address 0CH:
Bit 7	**4kB_Mode:** Size of internal RAM 0 = 2kB RAM (default). 1 = 4kB RAM
bit 6	**No_Check_Prm_Reserved:** Disables checking of the reserved Prm bits 0 = Reserved bits of Prm-telegram are checked (default). 1 = Reserved bits of Prm-telegram are not checked.
bit 5	**SYNC_Pol:** Polarity of SYNC pulse (for Isochron Mode only) 0 = negative polarity of SYNC pulse (default) 1 = positive polarity of SYNC pulse
bit 4	**SYNC_Ena:** Enable generation of SYNC pulse (for Isochron Mode only) 0 = SYNC pulse generation is disabled (default). 1 = SYNC pulse generation is enabled.
bit 3	**DX_Int_Port:** Port mode for Dataexchange Interrupt 0 = DX Interrupt not assigned to port DATA_EXCH (default). 1 = DX Interrupt (synchronized to GC-SYNC) assigned to port DATA_EXCH.
bit 2	**DX_Int_Mode:** Mode of Dataexchange Interrupt 0 = DX Interrupt only generated, if DOUT length not 0 (default). 1 = DX Interrupt generated after every DX-telegram
bit 1	**No_Check_GC_Reserved:** Disables checking of the reserved GC bits 0 = Reserved bits of GC-telegram are checked (default). 1 = Reserved bits of GC-telegram are not checked.
bit 0	**GC_Int_Mode:** Controls generation of GC Interrupt 0 = GC Interrupt is only generated, if changed GC telegram is received 1 = GC Interrupt is generated after every GC telegram (default)

Figure 3-18 : Coding of Mode Register 2

ModeRegister3

Address	Bit Position								Designation
	7	6	5	4	3	2	1	0	
	0	0	0	0	0	0	0	0	Reset Value
12H	Reserved	Reserved	Reserved	Reserved	PLL_Supported	En_Chk_SSAP	DX_Int_Mode_2	GC_Int_Mode_Ext	Mode Reg 3 7 .. 0

	Mode Register 3, Address 12H:
Bit 7	**Reserved**
bit 6	**Reserved**
bit 5	**Reserved**
bit 4	**Reserved**
bit 3	**PLL_Supported:** Enables IsoM-PLL 0 = PLL is disabled. 1 = PLL is enable; For use of PLL, SYNC_Ena must be set.
bit 2	**En_Chk_SSAP:** Evaluation of Source Address Extension 0 = VPC3+ accept any value of S_SAP 1 = VPC3+ only process the received telegram if the S_SAP match to the default values represented by the IEC 61158
bit 1	**DX_Int_Mode_2:** Mode of DX_Out interrupt 0 = DX_Out interrupt is generated after each Data_Exch telegram 1 = DX_Out interrupt is only generated, if received data is not equal to current data in Dx_Out buffer of user
bit 0	**GC_Int_Mode_Ext:** extend GC_Int_Mode, works only if GC_Int_Mode=0 0 = GC Interrupt is only generated, if changed GC telegram is received 1 = GC Interrupt is only generated, if GC telegram with changed Control_Command is received.

Figure 3-19 : Coding of Mode Register 3

Activating the Indication Function

The user activates or deactivates interrupts by setting or clearing the corresponding bit in the Interrupt Mask Register. If a bit is set, the corresponding interrupt is disabled (interrupt masked).

⚠ **Masking of an already active interrupt is not possible, that is, an active interrupt remains active after masking, but further activation of this interrupt is rejected.**

Address	Bit Position								Designation
	7	6	5	4	3	2	1	0	
04H (Intel)	DXB_Out	New_Ext_Prm	DXB_Link_Error	User_Timer_ Clock	WD_DP_ Mode_Timeout	Baud_Rate_ Detect	Go/Leave Data_EX	MAC_Reset/ Clock_Sync	Int-Mask-Reg 7 .. 0 See below for coding

	Interrupt-Mask-Register, Low-Byte, Address 04H (Intel):
Bit 7	**DXB_Out:** VPC 3+ has received a 'DXB telegram' and made the new output data available in the 'N' buffer.
Bit 6	**New_Ext_Prm_Data:** The VPC 3+ has received a 'Set_Ext_Param telegram' and made the data available in the Prm buffer.
Bit 5	**DXB_Link_Error:** The Watchdog cycle is elapsed and at least one Publisher-Subscriber connection breaks down.
Bit 4	**User_Timer_Clock:** The time base for the User_Timer_Clocks has run out (1 /10ms).
Bit 3	**WD_DP_Control_Timeout:** The watchdog timer has run out in the 'DP_Control' WD state
Bit 2	**Baudrate_Detect:** The VPC3+ has left the 'Baud_Search state' and found a baud rate.
Bit 1	**Go/Leave_DATA_EX:** The DP_SM has entered or exited the 'DATA_EX' state

Bit 0	**MAC_Reset (used if CS_Supported=0):**
	After it processes the current request, the VPC3+ has arrived at the offline state (by setting the 'Go_Offline bit')
	
	Clock_Sync (used if CS_Supported=1):
	The VPC3+ has received a Clock_Value telegram or an error occurs. Further differentiation is made in the Clock_Sync_buffer.

Figure 3-20 : Interrupt Mask Register, Low-Byte

Address	Bit Position								Designation
	15	14	13	12	11	10	9	8	
05H (Intel)	FDL_ind	Poll_End_ind	DX_Out	Diag_Buffer_ Changed	New_Prm_ Data	New_Cfg_ Data	New_SSA_ Data	New_GC Command	Int-Mask-Reg 15 .. 8 See below for coding

	Interrupt Mask Register 0, High-Byte, Address 05H (Intel):
Bit 15	**FDL_Ind:** The VPC 3+ has received an acyclic service request and made the data available in an indication buffer.
Bit 14	**Poll_End_Ind:** The VPC 3+ has sent the response to an acyclic service.
Bit 13	**DX_Out:** 0 = No special clear mode. 1 = Special clear mode. VPC3+ will accept data telegrams with data unit = 0
Bit 12	**Diag_Buffer_Changed:** Due to the request made by 'New_Diag_Cmd,' VPC3+ exchanged the diagnostics buffer and again made the old buffer available to the user.
Bit 11	**New_Prm_Data:** The VPC3+ has received a 'Set_Param telegram' and made the data available in the Prm buffer.
Bit 10	**New_Cfg_Data:** The VPC3+ has received a 'Check_Cfg telegram' and made the data available in the Cfg buffer.
Bit 9	**New_SSA_Date:** The VPC3+ has received a 'Set_Slave_Address telegram' and made the data available in the SSA buffer.
Bit 8	**New_GC_Command:** The VPC3+ has received a 'Global_Control telegram' and this byte is stored in the 'R_GC_Command' RAM cell.

Figure 3-21 : Interrupt Mask Register, High-Byte

For test purpose, the user can trigger any interrupt by writing to the Interrupt Request Register.

3.3 Configuration of dp_inc.h

In the dp_inc.h header file all external functions for the handling of VPC3+ are defined. The user can adapt here own functions for copying data to the VPC3+ or for copying data from the VPC3+.

```
#define CopyToVpc3_( _pToVpc3Memory, _pLocalMemory, _wLength )\
            memcpy( _pToVpc3Memory, _pLocalMemory, _wLength )

#define CopyFromVpc3_( _pLocalMemory, _pToVpc3Memory, _wLength )\
            memcpy( _pLocalMemory, _pToVpc3Memory, _wLength )

#define Vpc3MemSet_( _pToVpc3Memory, _bValue, _wLength )\
            memset( _pToVpc3Memory, _bValue, _wLength )

#define Vpc3MemCmp_( _pToVpc3Memory1, _pToVpc3Memory2, _wLength )\
            memcmp( _pToVpc3Memory1, _pToVpc3Memory2, _wLength )
```

3.4 Configuration of main.c

The user must add own code to following functions:

```
void DpAppl_SetResetVPC3Channel1          ( void );
void DpAppl_ClrResetVPC3Channel1          ( void );
void DpAppl_SetResetVPC3Channel2          ( void );
void DpAppl_ClrResetVPC3Channel2          ( void );
void DpAppl_EnableInterruptVPC3Channel1   ( void );
void DpAppl_DisableInterruptVPC3Channel1  ( void );
void DpAppl_EnableInterruptVPC3Channel2   ( void );
void DpAppl_DisableInterruptVPC3Channel2  ( void );
void DpAppl_EnableInterruptVPC3Sync       ( void );
void DpAppl_DisableInterruptVPC3Sync      ( void );
void DpAppl_EnableAllInterrupts           ( void );
void DpAppl_DisableAllInterrupts          ( void );
void Vpc3Wait_1ms                         ( void );
```

If the VPC3+S is setup to serial mode, following functions must be added:

```
#if VPC3_SERIAL_MODE

   void    Vpc3Write   ( VPC3_ADR wAddress, uint8_t bData );
   uint8_t Vpc3Read    ( VPC3_ADR wAddress );
   void    Vpc3MemSet  ( VPC3_ADR wAddress, uint8_t bValue,
                         uint16_t wLength );
   uint8_t Vpc3MemCmp  ( VPC3_UNSIGNED8_PTR pToVpc3Memory1,
                         VPC3_UNSIGNED8_PTR pToVpc3Memory2,
                         uint16_t wLength );
   void    CopyToVpc3  ( VPC3_UNSIGNED8_PTR pToVpc3Memory,
                         MEM_UNSIGNED8_PTR pLocalMemory,
                         uint16_t wLength );
   void    CopyFromVpc3( MEM_UNSIGNED8_PTR pLocalMemory,
                         VPC3_UNSIGNED8_PTR pToVpc3Memory,
                         uint16_t wLength );

#endif//#if VPC3_SERIAL_MODE
```

3.5 Memorytest of VPC3+

Before initialization of VPC3+ the memory of VPC3+ should be checked.

DP_ERROR_CODE VPC3_MemoryTest(void)		
Function	This function checks the memory of VPC3+. The starting address is 16hex and the end address depends on DP_VPC3_4KB_MODE (DpCfg.h).	
Parameter		
Return Value	DP_OK DP_VPC3_ERROR	Memory test OK Memory test failed

Figure 3-22 : Function VPC3_MemoryTest()

Memory test failed:
In this case read the status register of VPC3+. This register has a default value.

Address	**Bit Position**								**Designation**
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	
05H	VPC 3+ Release				Baudrate				Status-Reg
(Intel)	3	2	1	0	3	2	1	0	

	Status Register, High-Byte, Address 05H (Intel):
bit 15-12	**VPC 3+-Release 3..0 :** Release number for VPC 3+ 0000 = VPC3+ 1011 = VPC3+B 1100 = VPC3+C 1101 = MPI12x 1110 = VPC3+S Rest = Not possible
bit 11-8	**Baudrate 3..0 :** The baudrate found by VPC 3+ 0000 = 12,00 Mbit/s 0001 = 6,00 Mbit/s 0010 = 3,00 Mbit/s 0011 = 1,50 Mbit/s 0100 = 500,00 Kbit/s 0101 = 187,50 Kbit/s 0110 = 93,75 Kbit/s 0111 = 45,45 Kbit/s 1000 = 19,20 Kbit/s 1001 = 9,60 Kbit/s 1111 = after reset and during baudrate search Rest = not possible

Figure 3-23: Status Register, High-Byte

3.6 Initializing of VPC3+

The function **VPC3_Initialization()** handles the completely initializing of the VPC3+.

- Initializing RAM to zero
- Calculating buffer structures
- Initializing the ASIC with DP and FDL if necessary
- If necessary: setting up the MSAC_C2 SAPs according to transfer parameters. The MSAC_C1 SAPs mentioned above are set up, but are not yet opened.
- Initializing the resource manager (RM) and setting up the RM SAP. The RM SAP will only be opened after the ASIC is started with DPSE_START. The MSAC_C2 services are available immediately after DPSE_START.
- Enter the first free SAP as response data for RM SAP.

DP_ERROR_CODE VPC3_Initialization(uint8_t bSlaveAddress, uint16_t wIdentNumber, CFG_STRUCT sCfgData)		
Function	Initialization of VPC3+	
Parameter	bSlaveAddress	Address of the slave
	wIdentNumber	PROFIBUS Ident Number
	sCfgData	Default configuration of the slave
Return Value	DP_OK	Initialization OK
	***DP_NOT_OFFLINE_ERROR**	***Error VPC3 is not in OFFLINE state**
	DP_ADDRESS_ERROR	Error, DP Slave address
	DP_CALCULATE_IO_ERROR	Error with configuration bytes
	DP_DOUT_LEN_ERROR	Error with Dout length
	DP_DIN_LEN_ERROR	Error with Din length
	DP_DIAG_LEN_ERROR	Error with diagnostics length
	DP_PRM_LEN_ERROR	Error with parameter assignment data length
	DP_SSA_LEN_ERROR	Error with address data length
	DP_CFG_LEN_ERROR	Error with configuration data length
	DP_LESS_MEM_ERROR	Error Overall, too much memory used
	DP_LESS_MEM_FDL_ERROR	Error Overall, too much memory used

Figure 3-24 : Function VPC3_Initialization()

***If the VPC3+ not in the “OFFLINE” state, reset the VPC3+ once more!**

Before call up the **VPC3_Initialization()** function the user has to define the default configuration over the structure CFG_STRUCT. The default configuration will be placed into Read-Configuration buffer.

For example:

```
typedef struct
{
   uint8_t bLength;
   uint8_t abData[CFG_BUFSIZE];
} CFG_STRUCT;                              // defined in dp_if.h

CFG_STRUCT      sRealCfg;


sRealCfg.bLength                  = 0x02;    // length of configuration data
sRealCfg.abData[0]                = 0x25;    // master to slave (6Byte)
sRealCfg.abData[1]                = 0x17;    // slave to master (8Byte)

bError = VPC3_Initialization( 0x05, 0xADAC, sRealCfg );
```

3.7 Starting VPC3

If the ASIC could be correctly initialized with **VPC3_Initialization()**, it still has to be started. Between initialization and start, the user can still initialize buffers in the ASIC.
The VPC3+ goes online with the command:

VPC3_Start()		
Function	Starts the VPC3+	
Parameter	None	
Return Value	None	

Figure 3-25 : Function VPC3_Start()

After the command VPC3_Start() the VPC3+ generates one DxOut-event to clear the output data..

3.8 Startup Telegram Sequence

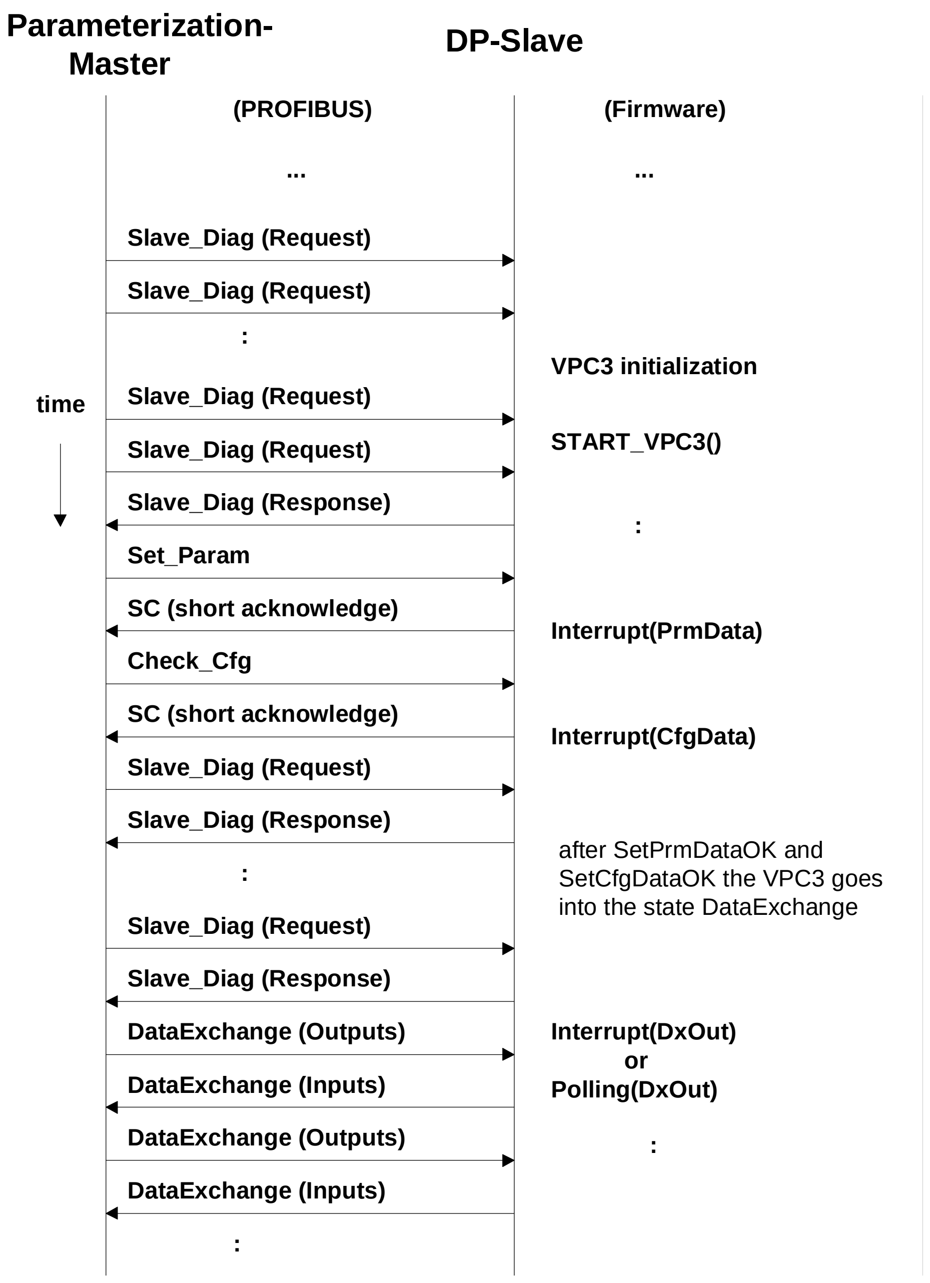


Figure 3-26 : Startup Telegram Sequence

3.8.1 Bus monitoring (Startup sequence)

Frame	Addr	Service	Msg type	Req/Res	SAPS	Datalen	Data
SD2	2->7	SRD_HIGH	Get Diagnostics	Req	62->60	0	
SD2	2<-7	DL	Get Diagnostics	Res	62<-60	6	02 05 00 FF AF FE
SD2	2->8	SRD_HIGH	Get Diagnostics	Req	62->60	0	
SD2	2<-8	DL	Get Diagnostics	Res	62<-60	6	02 05 00 FF AF FE
SD2	2->10	SRD_HIGH	Get Diagnostics	Req	62->60	0	
SD2	2<-10	DL	Get Diagnostics	Res	62<-60	6	02 05 00 FF AF FE
SD2	2->7	SRD_HIGH	Set Parameters	Req	62->61	19	B8 02 03 25 AF FE 00 E0 60 00 09 05 00 00 01 FF FF 00 00
ACK			Short acknowledge	Res			
SD2	2->8	SRD_HIGH	Set Parameters	Req	62->61	19	B8 02 03 25 AF FE 00 E0 60 00 09 05 00 00 01 FF FF 00 00
ACK			Short acknowledge	Res			
SD2	2->10	SRD_HIGH	Set Parameters	Req	62->61	39	B8 02 03 0B AF FE 00 C0 60 08 11 07 00 00 01 07 08 01 06 00 01 08 08 02 07 00 04 0C 81 00 00 05 00 00 01 FF FF 00 00
ACK			Short acknowledge	Res			
SD2	2->7	SRD_HIGH	Check Config	Req	62->62	20	42 00 00 01 42 00 00 02 82 00 00 03 C1 03 03 04 C1 01 01 05
ACK			Short acknowledge	Res			
SD2	2->8	SRD_HIGH	Check Config	Req	62->62	20	42 00 00 01 42 00 00 02 82 00 00 03 C1 03 03 04 C1 01 01 05
ACK			Short acknowledge	Res			
SD2	2->10	SRD_HIGH	Check Config	Req	62->62	36	42 00 00 01 42 00 00 02 82 00 00 03 C1 03 03 04 C1 01 01 05 42 00 FD 00 42 03 FD 03 03 00 00 FF 03 00 00 FF
ACK			Short acknowledge	Res			
SD2	2->7	SRD_HIGH	Get Diagnostics	Req	62->60	0	
SD2	2<-7	DL	Get Diagnostics	Res	62<-60	6	02 0C 00 02 AF FE
SD2	2->8	SRD_HIGH	Get Diagnostics	Req	62->60	0	
SD2	2<-8	DL	Get Diagnostics	Res	62<-60	6	02 0C 00 02 AF FE
SD2	2->10	SRD_HIGH	Get Diagnostics	Req	62->60	0	
SD2	2<-10	DL	Get Diagnostics	Res	62<-60	6	02 0C 00 02 AF FE
SD2	2->7	SRD_HIGH	Get Diagnostics	Req	62->60	0	
SD2	2<-7	DL	Get Diagnostics	Res	62<-60	6	02 0C 00 02 AF FE
SD2	2->8	SRD_HIGH	Get Diagnostics	Req	62->60	0	
SD2	2<-8	DL	Get Diagnostics	Res	62<-60	6	02 0E 00 02 AF FE
SD2	2->10	SRD_HIGH	Get Diagnostics	Req	62->60	0	

SD2	2<-10	DL	Get Diagnostics	Res	62<-60	6	02 0C 00 02 AF FE
SD2	2->7	SRD_HIGH	Get Diagnostics	Req	62->60	0	
SD2	2<-7	DL	Get Diagnostics	Res	62<-60	6	02 0E 00 02 AF FE
SD2	2->8	SRD_HIGH	Get Diagnostics	Req	62->60	0	
SD2	2<-8	DL	Get Diagnostics	Res	62<-60	6	00 0E 00 02 AF FE
SD2	2->10	SRD_HIGH	Get Diagnostics	Req	62->60	0	
SD2	2<-10	DL	Get Diagnostics	Res	62<-60	6	02 0E 00 02 AF FE
SD2	2->7	SRD_HIGH	Get Diagnostics	Req	62->60	0	
SD2	2<-7	DL	Get Diagnostics	Res	62<-60	6	00 0E 00 02 AF FE
SD2	2->8	SRD_HIGH	Get Diagnostics	Req	62->60	0	
SD2	2<-8	DL	Get Diagnostics	Res	62<-60	6	00 0C 00 02 AF FE
SD2	2->10	SRD_HIGH	Get Diagnostics	Req	62->60	0	
SD2	2<-10	DL	Get Diagnostics	Res	62<-60	6	00 0E 00 02 AF FE
SD1	2->8			Req			
SD2	127<-8	DL	Data Exchange	Res		8	08 00 00 00 00 00 00 00
SD2	2->7	SRD_HIGH	Get Diagnostics	Req	62->60	0	
SD2	2<-7	DL	Get Diagnostics	Res	62<-60	6	00 0C 00 02 AF FE
SD2	2->10	SRD_HIGH	Get Diagnostics	Req	62->60	0	
SD2	2<-10	DL	Get Diagnostics	Res	62<-60	6	00 0E 00 02 AF FE
SD1	2->7			Req			
SD2	127<-7	DL	Data Exchange	Res		8	07 E0 00 00 00 00 00 00
SD1	2->8			Req			
SD2	127<-8	DL	Data Exchange	Res		8	08 00 00 00 00 00 00 00
SD2	2->10	SRD_HIGH	Get Diagnostics	Req	62->60	0	
SD2	2<-10	DL	Get Diagnostics	Res	62<-60	6	00 0C 00 02 AF FE
SD1	2->7			Req			
SD2	127<-7	DL	Data Exchange	Res		8	07 E0 00 00 00 00 00 00
SD1	2->8			Req			
SD2	127<-8	DL	Data Exchange	Res		8	08 00 00 00 00 00 00 00
SD1	2->10	SRD_HIGH	Data Exchange	Req			
SD2	2<-10	DL	Data Exchange	Res		13	0A 80 00 00 00 00 00 00 00 00 00 00 00
SD1	2->7			Req			
SD2	127<-7	DL	Data Exchange	Res		8	07 E0 00 00 00 00 00 00
SD1	2->8			Req			
SD2	127<-8	DL	Data Exchange	Res		8	08 00 00 00 00 00 00 00
SD1	2->10	SRD_HIGH	Data Exchange	Req			
SD2	2<-10	DL	Data Exchange	Res		13	0A 80 00 00 00 00 00 00 00 00 00 00 00
SD1	2->7			Req			
SD2	127<-7	DL	Data Exchange	Res		8	07 E0 00 00 00 00 00 00
SD1	2->8			Req			
SD2	127<-8	DL	Data Exchange	Res		8	08 00 00 00 00 00 00 00
SD1	2->10	SRD_HIGH	Data Exchange	Req			

SD2	2<-10	DL	Data Exchange	Res		13	0A 80 00 00 00 00 00 00 00 00 00 00 00
SD1	2->7			Req			
SD2	127<-7	DL	Data Exchange	Res		8	07 E0 00 00 00 00 00 00
SD1	2->8			Req			
SD2	127<-8	DL	Data Exchange	Res		8	08 00 00 00 00 00 00 00
SD1	2->10	SRD_HIGH	Data Exchange	Req			
SD2	2<-10	DL	Data Exchange	Res		13	0A 80 00 00 00 00 00 00 00 00 00 00 00
SD1	2->7			Req			
SD2	127<-7	DL	Data Exchange	Res		8	07 E0 00 00 00 00 00 00
SD1	2->8			Req			
SD2	127<-8	DL	Data Exchange	Res		8	08 00 00 00 00 00 00 00
SD1	2->10	SRD_HIGH	Data Exchange	Req			
SD2	2<-10	DL	Data Exchange	Res		13	0A 80 00 00 00 00 00 00 00 00 00 00 00
SD1	2->7			Req			
SD2	127<-7	DL	Data Exchange	Res		8	07 E0 00 00 00 00 00 00
SD1	2->8			Req			
SD2	127<-8	DL	Data Exchange	Res		8	08 00 00 00 00 00 00 00
SD1	2->10	SRD_HIGH	Data Exchange	Req			
SD2	2<-10	DL	Data Exchange	Res		13	0A 80 00 00 00 00 00 00 00 00 00 00 00
SD1	2->7			Req			
SD2	127<-7	DL	Data Exchange	Res		8	07 E0 00 00 00 00 00 00
SD1	2->8			Req			
SD2	127<-8	DL	Data Exchange	Res		8	08 00 00 00 00 00 00 00

Figure 3-27 : Bus monitoring

Notes:

General VPC3-DP Functions 4

4.1 Interrupt Indication Function

The VPC3+ generates indications based on internals events. The indications can observed by means of polling or interrupt.

The user can mask each interrupt by setting the corresponding bit in the Interrupt Mask Register (DpCfg.h,). If interrupt are masked, the application must poll the Interrupt Request Register for active indications.

For interrupt handling and poll-mode refer to file “dp_isr.c”.

4.1.1 Reading the Indication

The user receives the event which has caused the interrupt by reading the Interrupt Register:

Address	**Bit Position**								**Designation**
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
02H (Intel)	DXB_Out	New_Ext_Prm	DXB_Link_Error	User_Timer_ Clock	WD_DP_ Mode_Timeout	Baud_Rate_ Detect	Go/Leave Data_EX	MAC_Reset \ Clock_Sync	Interrupt Register 7 .. 0

Address	**Bit Position**								**Designation**
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	
03H (Intel)	FDL_ind	Poll_End_ind	DX_Out	Diag_Buffer_ Changed	New_Prm_ Data	New_Cfg_ Data	New_SSA_ Data	New_GC Command	Interrupt-Register 15 .. 8

Indication	Description
VPC3_GET_IND_MAC_RESET	After processing the current request, the VPC3+ has entered the offline state (by setting the ‘Go_Offline’ bit).
VPC3_GET_IND_CLOCK_SYNC	The VPC3+ has received a Clock-Sync-telegram.
VPC3_ GET_IND _GO_LEAVE_DATA_EX	The DP_SM has entered the ‘DATA_EX’ state or has exited it.
VPC3_ GET_IND _BAUDRATE_DETECT	The VPC3+ has left the ‘Baud_Search state’ and has found a baud rate.
VPC3_ GET_IND _DP_WD_TIMEOUT	In the ‘DP_Control’ WD state , the watchdog timer has expired.
VPC3_ GET_IND _USER_TIMER_CLOCK	The time base of the User_Timer_Clock has expired (1/10ms).
VPC3_GET_IND_DXB_LINK_ERROR	The VPC3+ has updated the DXB Link structure. The data is available in the DXB_Link_Table buffer.
VPC3_GET_IND_NEW_EXT_PRM_DATA	The VPC3+ has received ‘Set_Ext_Param Message’ and has made the data available in the Prm buffer.
VPC3_GET_IND_DXB_OUT	The VPC3+ has received new data from the DXB Publisher. The data is available in the DXB_OUT buffer.
VPC3_GET_IND_NEW_GC_COMMAND	The VPC3+ has received a ‘Global_Control Message’ with a changed ‘GC_Command Byte’ and has stored this byte in the ‘R_GC_Command’ RAM cell.
VPC3_ GET_IND _NEW_SSA_DATA	The VPC3+ has received ‘Set_Slave_Address Message’ and has made the data available in the SSA buffer.
VPC3_ GET_IND _NEW_CFG_DATA	The VPC3+ has received Check_Cfg Message’ and has made the data available in the Cfg buffer.
VPC3_ GET_IND _NEW_PRM_DATA	The VPC3+ has received ‘Set_Param Message’ and has made the data available in the Prm buffer.
VPC3_GET_IND_DIAG_BUF_CHANGED	Requested by ‘New_Diag_Cmd’ , the VPC3+ has Exchanged the diagnostics buffer and has made the old buffer available again to the user.
VPC3_ GET_IND _DX_OUT	The VPC3+ has received a ‘Write_Read_Data Message’ and has made the new output data available in the N buffer. For ‘Power_On’ and for ‘Leave_Master’, the VPC3+ clears the N buffer contents and also generates this interrupt.
VPC3_GET_IND_POLL_END_IND	The master has fetched the FDL response.
VPC3_GET_IND_FDL_IND	The VPC3+ has received a FDL indication.

Figure 4-1 : Interrupt indication

4.1.2 Acknowledging the Indication

The user acknowledges the indication received through the interrupt routine by writing to the Interrupt Acknowledge Register:

```
VPC3_CON_IND_MAC_RESET()
VPC3_CON_IND_CLOCK_SYNC()
VPC3_CON_IND_GO_LEAVE_DATA_EX()
VPC3_CON_IND_BAUDRATE_DETECT()
VPC3_CON_IND_DP_WD_TIMEOUT()
VPC3_CON_IND_USER_TIMER_CLOCK()
VPC3_CON_IND_DXB_LINK_ERROR()
VPC3_CON_IND_NEW_EXT_PRM_DATA()
VPC3_CON_IND_DXB_OUT()
VPC3_CON_IND_NEW_GC_COMMAND()
VPC3_CON_IND_NEW_SSA_DATA()
VPC3_CON_IND_DIAG_BUF_CHANGED()
VPC3_CON_IND_DX_OUT()
VPC3_CON_IND_POLL_END_IND()
VPC3_CON_IND_FDL_IND()
```

Interrupt 10 (New_Cfg_Data) and interrupt 11 (New_Prm_Data) can not be acknowledged with the Interrupt Acknowledge Register. They are acknowledged by reading from

```
VPC3_SET_PRM_DATA_OK()
VPC3_SET_PRM_DATA_NOK()

VPC3_SET_CFG_DATA_OK()
VPC3_SET_CFG_DATA_NOK()
```

4.1.3 Ending the Indication

The EOI-bit (End Of Interrupt) in mode register 1, bit 1, ends the indication sequence / interrupt function:

VPC3_SET_EOI()		
Function	Ends indication of interrupt function	
Parameter	None	
Return Value	None	

Figure 4-2 : Function VPC3_SET_EOI()

4.1.4 Polling the Indication

The user can poll indications via the Interrupt Request Register:

```
VPC3_POLL_IND_MAC_RESET()
VPC3_POLL_IND_CLOCK_SYNC()
VPC3_POLL_IND_GO_LEAVE_DATA_EX()
VPC3_POLL_IND_BAUDRATE_DETECT()
VPC3_POLL_IND_DP_WD_TIMEOUT()
VPC3_POLL_IND_USER_TIMER_CLOCK()
VPC3_POLL_IND_DXB_LINK_ERROR()
VPC3_POLL_IND_NEW_EXT_PRM_DATA()
VPC3_POLL_IND_DXB_OUT()
VPC3_POLL_IND_NEW_GC_COMMAND()
VPC3_POLL_IND_NEW_SSA_DATA()
VPC3_POLL_IND_DIAG_BUF_CHANGED()
VPC3_POLL_IND_DX_OUT()
VPC3_POLL_IND_POLL_END_IND()
VPC3_POLL_IND_FDL_IND()
```

Poll indications can be acknowledged via the Interrupt Acknowledge Register:

```
VPC3_CON_IND_MAC_RESET()
VPC3_CON_IND_CLOCK_SYNC()
VPC3_CON_IND_GO_LEAVE_DATA_EX()
VPC3_CON_IND_BAUDRATE_DETECT()
VPC3_CON_IND_DP_WD_TIMEOUT()
VPC3_CON_IND_USER_TIMER_CLOCK()
VPC3_CON_IND_DXB_LINK_ERROR()
VPC3_CON_IND_NEW_EXT_PRM_DATA()
VPC3_CON_IND_DXB_OUT()
VPC3_CON_IND_NEW_GC_COMMAND()
VPC3_CON_IND_NEW_SSA_DATA()
VPC3_CON_IND_DIAG_BUF_CHANGED()
VPC3_CON_IND_DX_OUT()
VPC3_CON_IND_POLL_END_IND()
VPC3_CON_IND_FDL_IND()
```

4.2 Parameter Data

4.2.1 Checking the Parameter Data

Checking of parameter data is application dependent. Therefore the user is responsible for checking the received user specific parameter data. With the interrupt VPC3_GET_IND_NEW_PRM_DATA the function VPC3_Isr is called and then, if necessary, the user specific parameter data checking sequence within the interrupt routine.

Callback function:

DP_ERROR_CODE DpPrm_ChkNewPrmData(MEM_UNSIGNED8_PTR pbPrmData, uint8_t bPrmLength)		
Function	Checking parameter data	
Parameter	pbPrmData	Pointer to parameter data
	bPrmLength	Length of parameter data
Return Value	DP_OK DP_NOK	Parameter data OK ***Error Parameter data not OK**

Figure 4-3 : Function DpPrm_ChkNewPrmData()

Functions:

uint8_t VPC3_GET_PRM_LEN()		
Function	Get the length of the received parameter data	
Parameter	None	
Return Value	Length of prm data	

Figure 4-4 : Function VPC3_GET_PRM_LEN

VPC3_UNSIGNED8_PTR VPC3_GET_PRM_BUF_PTR ()		
Function	Fetch buffer pointer of the parameter buffer.	
Parameter	None	
Return Value	pointer to the parameter data buffer	

Figure 4-5 : Function VPC3_GET_PRM_BUF_PTR

uint8_t VPC3_SET_PRM_DATA_OK()		
Function	Positive acknowledge of the checked parameter data.	
Parameter	None	
Return Value	VPC3_PRM_FINISHED	No further parameter assignment message is present => end of sequence.
	VPC3_PRM_CONFLICT	Another parameter assignment message is present! => repeat check of requested parameter assignment.
	VPC3_PRM_NOT_ALLOWED	Access in present bus mode is not permitted. For example, it is possible that the watchdog has expired during verification.

Figure 4-6 : Function VPC3_SET_PRM_DATA_OK

uint8_t VPC3_SET_PRM_DATA_NOK()		
Function	Negative acknowledge of the checked parameter data.	
Parameter	None	
Return Value	VPC3_PRM_FINISHED	No further parameter assignment message is present => end of sequence.
	VPC3_PRM_CONFLICT	Another parameter assignment message is present! => repeat check of requested parameter assignment.
	VPC3_PRM_NOT_ALLOWED	Access in present bus mode is not permitted. For example, it is possible that the watchdog has expired during verification. Verifying the parameter data (and possibly series-connected functions in the application) are to be cancelled.

Figure 4-7 : Function VPC3_SET_PRM_DATA_NOK()

⚠ Acknowledging the New_Prm_Data interrupt by using one of these commands means, that the corresponding interrupt request bit is cleared. The New_Prm_Data interrupt can not be acknowledged via the Interrupt Acknowledge Register

⚠ **Caution:**
When both, configuration settings and parameter settings, are received, it is mandatory to verify and acknowledge parameter data first. Then the configuration settings may be processed.

4.2.2 Parameter Data Structure

VPC3+ evaluates the first seven data bytes (without user prm data), or the first eight data bytes (with user prm data). The first seven bytes are specified according to the standard. The next three bytes are used for the extended profibus services DPV1 and DPV2. The additional bytes are available to the application.

Byte	Bit Position								Designation
	7	6	5	4	3	2	1	0	
0	Lock_ Req	Unlock_ Req	Sync_ Req	Freeze_ Req	WD_On	Reserved	Reserved	Reserved	Station Status
1									WD_Fact_1
2									WD_Fact_2
3									MinTSDR
4									Ident_Number_High
5									Ident_Number_Low
6									Group_Ident
7 : 9									DPV1_STATUS1..3
10 : 243									User_Prm_Data

Figure 4-8 : Format of the Set_Param Telegram

⚠ **Don’t use DPV1_STATUS1..3 as User_Prm_Data.**

	DPV1_STATUS1:
Bit 7	**DPV1_Enable:** 0 = The slave is operated in the DP mode. (default state) 1 = The slave is operated in the DPV1 mode.
Bit 6	***Fail_Safe:** 0 = The slave is not operated in the Fail Safe mode (default state). 1 = The slave is operated in the Fail Safe mode.
Bit 5	**Publisher_Enable:** 0 = The slave is not operated in the DXB Publisher mode (default state). 1 = The slave is operated in the DXB Publisher mode.
Bit 4-3	**Reserved:** To be parameterized with '0'
Bit 2	**WD_Base:** Watchdog Time Base 0 = Watchdog time base is 10 ms (default state) 1 = Watchdog time base is 1 ms
Bit 1	**Dis_Stop_Control:** Disable Stop-Bit Control 0 = Stop-bit monitoring in the receiver is enabled (default state) 1 = Stop-bit monitoring in the receiver is disabled
Bit 0	**Dis_Start_Control:** Disable Start-Bit Control 0 = Start-bit monitoring in the receiver is enabled (default state) 1 = Start-bit monitoring in the receiver is disabled

Figure 4-9 : DPV1_STATUS1

*)If the DP-Slave requires the Fail Safe mode and the master does not set this bit, the slave has to reject the parameter assignment.

	DPV1_STATUS2:
Bit 7	**Enable_Pull_Plug_Alarm:** 0 = Enable_Pull_Plug_Alarm disabled 1 = Enable_Pull_Plug_Alarm enabled.
Bit 6	**Enable_Process_Alarm:** 0 = Enable_Process_Alarm disabled 1 = Enable_Process_Alarm enabled.
Bit 5	**Enable_Diagnostic_Alarm:** 0 = Enable_Diagnostic_Alarm disabled 1 = Enable_Diagnostic_Alarm enabled.
Bit 4	**Enable_Manufacturer_Specific_Alarm:** 0 = Enable_Manufacturer_Specific_Alarm disabled 1 = Enable_Manufacturer_Specific_Alarm enabled.
Bit 3	**Enable_Status_Alarm:** 0 = Enable_Status_Alarm disabled 1 = Enable_Status_Alarm enabled.
Bit 2	**Enable_Update_Alarm:** 0 = Enable_Update_Alarm disabled 1 = Enable_Update_Alarm enabled.
Bit 1	**Reserved:** To be parameterized with ‘0’
Bit 0	**Chk_Cfg_Mode:** 0 = Chk_Cfg according to EN50170 (default state) 1 = User-specific evaluation of Chk_Cfg

Figure 4-10 : DPV1_STATUS2

	DPV1_STATUS3:
bit 7-5	**Reserved:** To be parameterized with ‘0’
bit 4	**IsoM_Req:** Isochron Mode Request 0 = Isochron Mode disabled 1 = Isochron Mode enabled
bit 3	**Prm_Structure:** 0 = Prm telegram according to EN50170 1 = Prm telegram in structured form (DPV2 extension)
bit 0-2	**Alarm_Mode:** limits the number of active alarms 0 = 1 alarm of each type 1 = 2 alarms in total 2 = 4 alarms in total 3 = 8 alarms in total 4 = 12 alarms in total 5 = 16 alarms in total 6 = 24 alarms in total 7 = 32 alarms in total

Figure 4-11 : DPV1_STATUS3

If **Prm_Structure set to 1**, the prm-data are in the structured form:

Byte	Bit Position								Designation
	7	6	5	4	3	2	1	0	
0 : 6									See above
7									DPV1_STATUS1
8									DPV1_STATUS2
9					1				DPV1_STATUS3
10									Structured_Length
11									Structure_Type 0x02: PrmCmd 0x03: DXB LinkTable 0x04: ISOCHRON 0x07: DXB Subscriber 0x08: Time AR 0x81: USER_PRM
12									Slotnumber
13									Reserved
14 : :									Data

Figure 4-12 : Structured Format of the Set_Param Telegram

4.3 Configuration Data

4.3.1 Checking Configuration Data

Checking of configuration data is application dependent. Therefore the user is responsible for checking the received configuration data. With the interrupt VPC3_INT_NEW_CFG_DATA function VPC3_Isr is called and then, if necessary, the user specific configuration data checking sequence within the interrupt routine.

Callback function:

E_DP_CFG_ERROR DpCfg_ChkNewCfgData(MEM_UNSIGNED8_PTR pbCfgData, uint8_t bCfgLength)		
Function	Checking configuration data	
Parameter	pbCfgData	Pointer to configuration data
	bCfgLength	Length of configuration data
Return Value	DP_CFG_OK DP_CFG_FAULT DP_CFG_UPDATE	Configuration data OK ***Error Configuration data not OK** Configuration data is OK, but different from ReadCfg-buffer.

Figure 4-13 : Function DpCfg_ChkNewCfgData()

Functions:

uint8_t VPC3_GET_READ_CFG_LEN() **uint8_t VPC3_GET_CFG_LEN()**		
Function	Get the length of the configuration data.	
Parameter	None	
Return Value	Length of cfg data	

Figure 4-14 : Function VPC3_GET_CFG_LEN

VPC3_UNSIGNED8_PTR VPC3_GET_READ_CFG_BUF_PTR () **VPC3_UNSIGNED8_PTR VPC3_GET_CFG_BUF_PTR ()**		
Function	Fetch buffer pointer of the configuration buffer.	
Parameter	None	
Return Value	pointer to the configuration data buffer	

Figure 4-15 : Function VPC3_GET_CFG_BUF_PTR

Within the verification function, the user compares the received Cfg_Data with the Real_Cfg_Data (Real_Cfg_Data was set during initialization).

uint8_t VPC3_SET_CFG_DATA_OK()		
Function	Positive acknowledge of the checked configuration data.	
Parameter	None	
Return Value	VPC3_CFG_FINISHED	No further configuration message is present => end of sequence.
	VPC3_CFG_CONFLICT	An additional configuration message is present! => Repeat verification of the requested configuration.
	VPC3_CFG_NOT_ALLOWED	Access is not permitted in the present bus mode. For example, it is possible the watchdog has run out during verification. The verification of the configuration data (and possibly subsequent functions in the application) are to be cancelled.

Figure 4-16 : Function VPC3_SET_CFG_DATA_OK

uint8_t VPC3_SET_CFG_DATA_NOK()		
Function	Negative acknowledge of the checked configuration data.	
Parameter	None	
Return Value	VPC3_CFG_FINISHED	No further configuration message is present => end of sequence.
	VPC3_CFG_CONFLICT	An additional configuration message is present! => Repeat verification of the requested configuration.
	VPC3_CFG_NOT_ALLOWED	Access is not permitted in the present bus mode. For example, it is possible the watchdog has run out during verification. The verification of the configuration data (and possibly subsequent functions in the application) are to be cancelled.

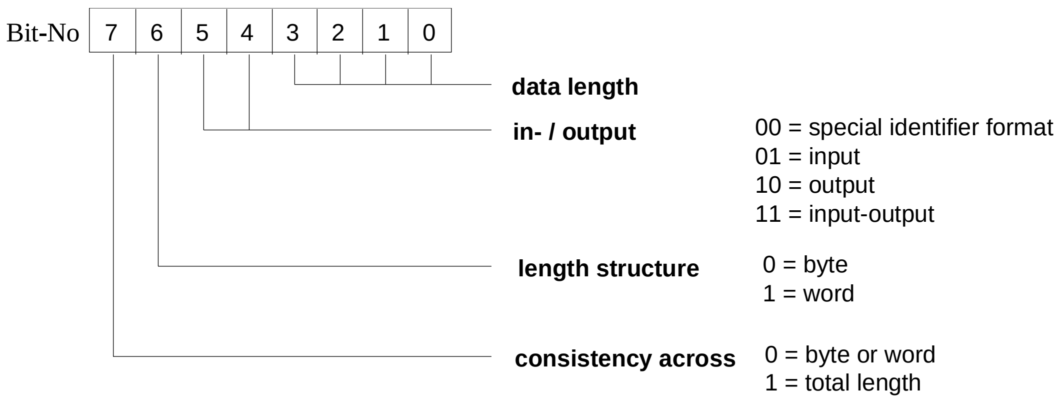
Figure 4-17 : Function VPC3_SET_CFG_DATA_NOK

Acknowledging the New_Cfg_Data interrupt by using one of these commands means, that the corresponding interrupt request bit is cleared. The New_Cfg_Data interrupt can not be acknowledged via the Interrupt Acknowledge Register

Caution:
When both, configuration settings and parameter settings, are received, it is mandatory to verify and acknowledge parameter data first. Then the configuration settings may be processed.

4.3.2 Configuration Data Formats

General format:



For example, the identifiers correspond to
14 hex = 5 bytes input
27 hex = 8 bytes output

Figure 4-18 : General Configuration Data Format

In order to cover complexer configurations, greater flexibility is attained in the case of PROFIBUS DP through a special expansion of the actual identification system. In addition, this special ID format makes it possible to determine the number of the input- and output bytes of this ID. Furthermore, user-specific data can be added.

Special format:

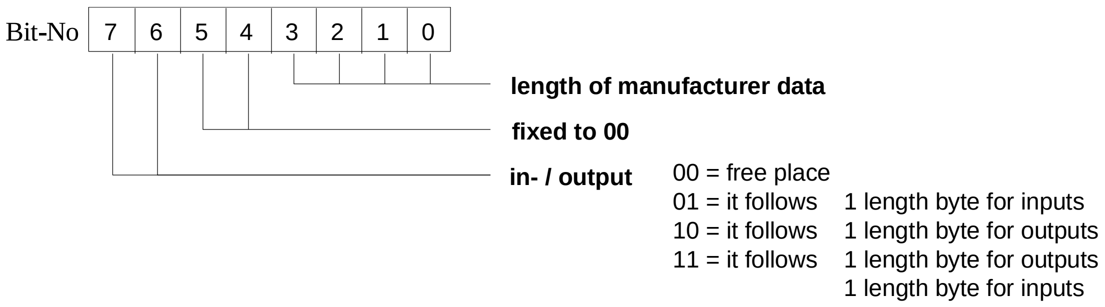

Bit-No
7
6
5
4
3
2
1
0
length of manufacturer data
fixed to 00
in- / output
00 = free place
01 = it follows 1 length byte for inputs
10 = it follows 1 length byte for outputs
11 = it follows 1 length byte for outputs
1 length byte for inputs

Figure 4-19 : Special Configuration Data Format

The length indication for manufacturer-specific data is to be interpreted as follows:

0	No manufacturer-specific data follows; it is not to be present in the Real_Cfg_Data.
1 to 14	Manufacturer-specific data of the specified length follows; it has to agree with the data contained in Real_Cfg_Data
15	No manufacturer-specific data follows; there is no check.

The structure of the length bytes looks like this:

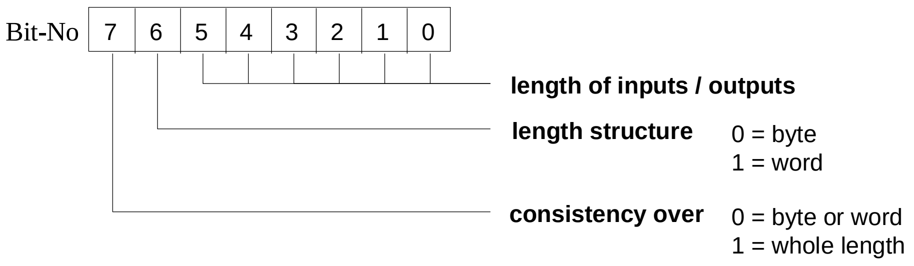


Figure 4-20 : Special Configuration Data Format

For example: C0hex, 87hex,84hex (8 bytes output, 5 bytes input)

4.4 Transfer of Output Data

VPC3_INT_DX_OUT in the interrupt function VPC3_Isr() indicates the receipt of output data from the DP-Master. The function VPC3_GetDoutBufPtr () returns the buffer pointer, and also the state of the Dout-buffer. The lengths of the outputs are not transferred with every update. The length agrees with the length transferred with VPC3_SetIoDataLength(), otherwise VPC3+ would branch to the WAIT_PRM state.

VPC3_UNSIGNED8_PTR VPC3_GetDoutBufPtr (MEM_UNSIGNED8_PTR pbState)		
Function	Fetch buffer pointer and state of the output buffer.	
Parameter	Pointer to variable into which the state of the output buffer is to be written	
Return Value	pointer to the output data buffer NIL, if no diagnostics buffer in the ‘U’ state	
	state of the output buffer	NEW_DOUT_BUF DOUT_BUF_CLEARED

Figure 4-21 : Function VPC3_Get_DoutBufPtr()

The input-/output data length can be reconfigured with the functions described in the Initialization section (VPC3_SetIoDataLength(), VPC3_CalculateInpOutpLength(),...).

4.5 Transfer of Input Data

As described, the application has to fetch a buffer for the input data with the VPC3_GetDinBufPtr() function before the first entry of its input data. With the command

uint8_t VPC3_INPUT_UPDATE ()		
Function	Change the input buffer.	
Parameter	None	
Return Value	New U-buffer	1 = Din_Buf_Ptr1 2 = Din_Buf_Ptr2 3 = Din_Buf_Ptr3

Figure 4-22 : Function VPC3_INPUT_UPDATE

the user can repeatedly transfer the current input data from the user to the VPC3+. The length of the inputs is not transferred with every update. The length must agree with the length transferred with function VPC3_SetIoDataLength().

VPC3_UNSIGNED8_PTR VPC3_GetDinBufPtr ()		
Function	Fetch buffer pointer of the input buffer.	
Parameter	None	
Return Value	pointer to the input data buffer NIL, if no diagnostics buffer in the 'U' state	

Figure 4-23 : Function VPC3_GetDinBufPtr

The input-/output data length can be reconfigured with the functions described in the Initialization section (VPC3_SetIoDataLength(), VPC3_CalculateInpOutpLength(),...).

4.6 Diagnostic

4.6.1 Transferring Diagnostic Data

With the function VPC3_SetDiagnosis(), the user can transfer diagnostic data to the VPC3+. Before calling this function, the user has to get a pointer to the free diagnosis buffer with the function VPC3_GetDiagBufPtr.

DP_ERROR_CODE VPC3_SetDiagnosis(MEM_UNSIGNED8_PTR pbToUserDiagData, uint8_t bUserDiagLength, uint8_t bDiagControl, uint8_t bCheckDiagFlag)		
Function	Transfer user diagnostic to VPC3+	
Parameter	pbToUserDiagData	Pointer to user diagnostic block
	bUserDiagLength	Length of user diagnostic block
	bDiagControl	0: reset Diag.ExtDiag / Diag.StatDiag-bit 1: set Diag.ExtDiag-bit 2: set Diag.StatDiag-bit
	bCheckDiagFlag	FALSE: don't check DIAG_FLAG of VPC3+ TRUE: check DIAG_FLAG of VPC3+ (see VPC3+ Status register)
Return value	DP_OK	Execution OK, message copied to VPC3+
	OLD_DIAG_NOT_SEND_ERROR	Error, wait because last diagnostic message isn't send
	BUFFER_LENGTH_ERROR	Error, diagnostic message is too long
	NO_BUFFER_ERROR	Error, wait no VPC3+ diagnostic buffer available
	CONTROL_BYTE_ERROR	Error of bDiagControl
	BUFFER_ERROR	Error, wrong diagnostic header
	NOT_POSSIBLE_ERROR	Error, unknown error

Figure 4-24 : Function VPC3_SetDiagnosis()

VPC3_UNSIGNED8_PTR VPC3_GetDiagBufPtr ()		
Function	Fetch buffer pointer of the diagnostic buffer.	
Parameter	None	
Return Value	Pointer to the diagnostics buffer NIL, if no diagnostics buffer in the 'U' state	

Figure 4-25 : Function VPC3_GetDiagBufPtr

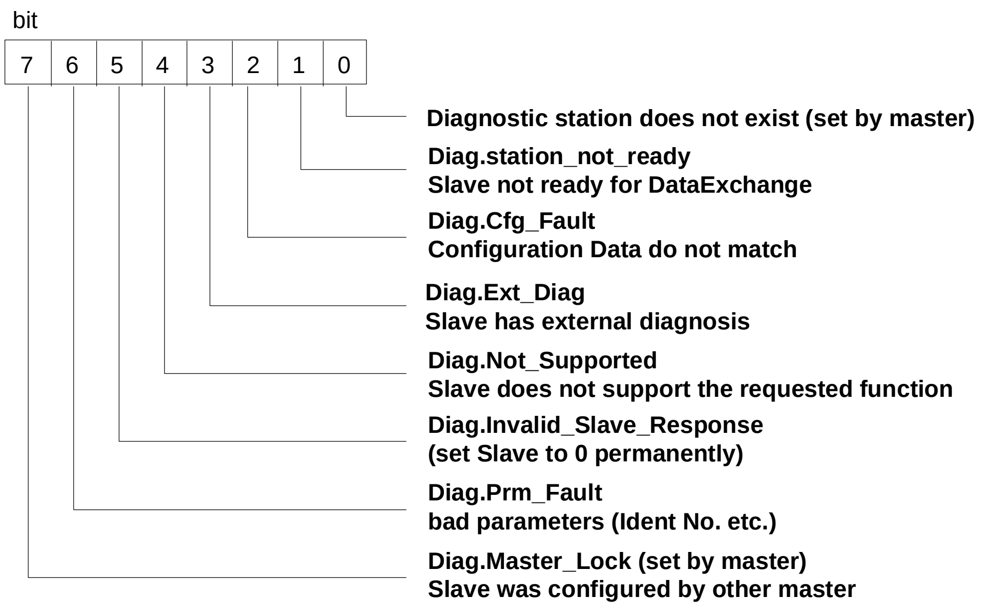
4.6.2 Structure of diagnostic block

Structure of the data block to be transferred for expanded diagnostics:

Byte	Diagnosis Data	Comment
0	Station Status_1	Byte 0 to 5 permanent diagnostic header
1	Station Status_2	
2	Station Status_3	
3	Diag.Master_Add	
4	Ident_Number_High	
5	Ident_Number_Low	
6 : 243	Ext_Diag_Data	Start of user diagnostic in the DP Standard format

Figure 4-26 : Structure of diagnosticv block

Station Status_1



bit
7
6
5
4
3
2
1
0
Diagnostic station does not exist (set by master)
Diag.station_not_ready
Slave not ready for DataExchange
Diag.Cfg_Fault
Configuration Data do not match
Diag.Ext_Diag
Slave has external diagnosis
Diag.Not_Supported
Slave does not support the requested function
Diag.Invalid_Slave_Response
(set Slave to 0 permanently)
Diag.Prm_Fault
bad parameters (Ident No. etc.)
Diag.Master_Lock (set by master)
Slave was configured by other master

Figure 4-27 : Structure of Station_Status_1

Station Status_2

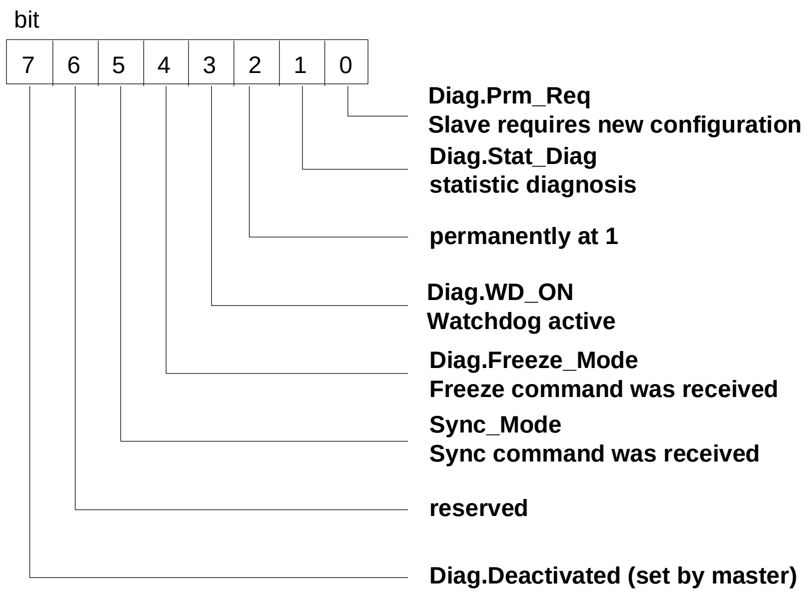


Figure 4-28 : Structure of Station_Status_2

Station Status_3

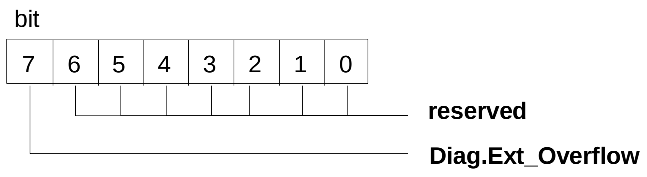


Figure 4-29 : Structure of Station_Status_3

4.6.3 User specific diagnostic

The user-specific diagnostic can be filed in three different formats:

Device related diagnostic

The diagnostic information can be coded as required:

	Bit7	Bit6	Bit5-0
Header byte	**0**	**0**	Block length in bytes,including header
Diagnostics field ...	Coding of diagnostic is device specific, can be specified as required		

Figure 4-30 : Device related diagnostic

Identifier related diagnostic

For each used identifier byte at the configuration one bit is reserved. It is padded to byte limits. The bits which are not configured shall be set to zero. A set bit means that in this I/O area diagnostic is pending.

	Bit7	Bit6	Bit5-0					
Header byte	**0**	**1**	Block length in bytes,including header					
Bit structure ...	1	0	1	1	0	0	0	0

Figure 4-31 : Device related diagnostic

Channel related diagnostic

In this block the diagnosed channels and the diagnostic reason are entered in turn. The length per entry is 3 octets.

	Bit7	Bit6	Bit5	Bit4-0
Header byte	**1**	**0**	Identification number	
Channel Number	Coding Input/Output		Channel number (0..63)	
Type of diagnosis	Coding Channel type			Coding Error type

Figure 4-32 : Channel related diagnostic

Coding Input/Output	
00	Reserved
01	Input
10	Output
11	Input / Output

Figure 4-33 : Coding Input/Output

Coding Channel type	
000	Reserved
001	Bit
010	2 bit
011	4 bit
100	Byte
101	Word
110	2 words
111	Reserved

Figure 4-34 : Coding Channel type

Coding Error type	
0	reserved
1	short circuit
2	undervoltage
3	overvoltage
4	overload
5	overtemperature
6	line break
7	upper limit value exceeded
8	lower limit value exceeded
9	error
10	reserved
...	...
15	reserved
16	manufacturer specific
...	...
31	manufacturer specific

Figure 4-35 : Coding Error type

Example: Structure of a diagnostic according to the pattern above:

MSB							**LSB**	
7	6	5	4	3	2	1	0	
0	**0**	0	0	0	1	0	0	**Device** related diagnostic
Device specific								Meaning of the bits
diagnostics field of								is specified
length 3								manufacturer specific
0	**1**	0	0	0	1	0	1	**Identifier** related diagnostic
							1	Identification number 0 has diagnostic
			1					Identification number 12 has diagnostic
						1		Identification number 17 has diagnostic
1	**0**	0	0	0	0	0	0	**Channel** related diagnostic, number 0
0	0	0	0	0	0	1	0	Channel 2
0	0	0	1	0	1	0	0	Overload, channel organized bit by bit
1	**0**	0	0	1	1	0	0	**Channel** related diagnostic, number 12
0	0	0	0	0	1	1	0	Channel 6
1	0	1	0	0	1	1	1	Upper limit, word by word

Figure 4-36 : Example

4.7 Changing the Slave Address

A request for changing the slave address is indicated through NEW_SSA_DATA. With the macro VPC3_GET_SSA_BUF_PTR(), a pointer to the buffer with the new slave address can be read. With the macro VPC3_GET_SSA_LEN(), the user is informed of the length of the SSA buffer received.

Callback function:

void DpAppl_IsrNewSetSlaveAddress(MEM_STRUC_SSA_BLOCK_PTR psSsa)		
Function	VPC3+ has received new Set slave address telegram.	
Parameter	psSsa	Pointer to set slave address structure
Return value		

Figure 4-37 : Function DpAppl_IsrNewSetSlaveAddress ()

Functions:

uint8_t VPC3_GET_SSA_LEN()		
Function	Get the length of the received ssa data	
Parameter	None	
Return Value	Length of ssa data	

Figure 4-38 : Function VPC3_GET_SSA_LEN

VPC3_UNSIGNED8_PTR VPC3_GET_SSA_BUF_PTR ()		
Function	Fetch buffer pointer of the ssa buffer.	
Parameter	None	
Return Value	pointer to the ssa data buffer	

Figure 4-39 : Function VPC3_GET_SSA_BUF_PTR

Structure of the Set_Slave_Address telegram:

Byte	Bit Position								Designation
	7	6	5	4	3	2	1	0	
0									New_Slave_Address
1									Ident_Number_High
2									Ident_Number_Low
3									No_Add_Chg
4 : 243									Rem_Save_Data additional application specific data

Figure 4-40 : Structure of the Set_Slave_Address telegram

4.8 Global Control Commands

The interrupt New_GC_Command indicates the arrival of a Global_Control message. The command VPC3_GET_IND_NEW_GC_COMMAND supplies the Control_Command byte. This makes it possible for the user to react to these commands. The VPC3+ internally processes these commands regarding buffer management. That is, in the case of 'Clear', the output data is deleted and the cleared buffer is made available to the user.

Callback function:

void DpAppl_IsrNewGlobalControlCommand(uint8_t bGcCommand)		
Function	VPC3+ has received new global control command.	
Parameter	bGcCommand	Global control command
Return value		

Figure 4-41 : Function DpAppl_IsrNewGlobalControlCommand ()

Functions:

VPC3_UNSIGNED8_PTR VPC3_GET_GC_COMMAND ()		
Function	Fetch global control byte.	
Parameter	None	
Return Value	Global control byte	

Figure 4-42 : Function VPC3_GET_GC_COMMAND

Address	Bit Position								Designation
	7	6	5	4	3	2	1	0	
3CH	Reserved	Reserved	Sync	Unsync	Freeze	Unfreeze	Clear_Data	Reserved	R_GC_ Command See coding below

	R_GC_Command, Address 3CH:
Bit 7-6	**Reserved**
Bit 5	**Sync:** The output data transferred with a WRITE_READ_DATA telegram is changed from ‘D’ to ‘N.’ The following transferred output data is kept in ‘D’ until the next ‘Sync’ command is issued.
Bit 4	**Unsync:** The ‘Unsync’ command cancels the ‘Sync’ command.
Bit 3	**Freeze:** The input data is fetched from ‘N’ to ‘D’ and ‘frozen’. New input data is not fetched again until the master sends the next ‘Freeze’ command.
Bit 2	**Unfreeze:** The ‘Unfreeze’ command cancels the ‘Freeze’ command.
Bit 1	**Clear_Data:** With this command, the output data is deleted in ‘D’ and is changed to ‘N’.
Bit 0	**Reserved**

Figure 4-43 : Description GC_COMMAND

4.9 Watchdog Timeout in DP-Control

The interrupt VPC3_INT_DP_WD_TIMEOUT indicates that the slave lost bus communication to the master. The following command returns the status of the watchdog state machine.

uint8_t VPC3_GET_WD_STATE()		
Function	Get the Wactdog State.	
Parameter	None	
Return Value	Watchdog State	

Figure 4-44 : Function VPC3_GET_WD_STATE()

Watchdog State	**Description**
BAUD_SEARCH	Baudrate search
BAUD_CONTROL	Monitoring the baudrate
DP_MODE	DP_Mode; that is, bus watchdog activated

Figure 4-45 : Description Wachdog State

4.9.1 Leaving the Data Exchange State

The VPC3_INT_GO_LEAVE_DATA_EX message indicates that the VPC3+ made a state change in the internal state machine.

With the following command the application is informed whether the VPC3+ has entered the data exchange state or left it. The cause for this transition can be a faulty parameter assignment message in the data transfer phase, for example.

Callback function:

void DpAppl_IsrGoLeaveDataExchange(uint8_t bDpState)		
Function	VPC3+ has received new profibus state.	
Parameter	bDpState	State of profibus connection
Return value		

Figure 4-46 : Function DpAppl_IsrGoLeaveDataExchange()

Functions:

uint8_t VPC3_GET_DP_STATE()		
Function	Get the DP State.	
Parameter	None	
Return Value	DP State	

Figure 4-47 : Function VPC3_GET_DP_STATE()

States of the DP-State Machine:

DP- State	**Description**
WAIT_PRM	Wait for parameter assignment
WAIT_CFG	Wait for configuration
DATA_EX	Data exchange
DP_ERROR	Error

Figure 4-48 : DP States

4.10 VPC3_Reset (Go_Offline)

With the command **VPC3_GO_OFFLINE()** the VPC3+ enters the offline state, after the actual request is processed. The command **VPC3_GET_OFF_PASS()** determines whether the transition to offline was made. If the return value is ‘zero’, the VPC3+ is ‘Offline’. If the return value is 1, the VPC3+ is ‘Passiv Idle’.

4.11 Leave Master

The command **VPC3_SET_USER_LEAVE_MASTER()** causes the VPC3+ to change into the state ‘Wait_Prm’.

4.12 FatalError (DP+MSAC_C1+MSAC_C2)

The firmware calls this function if a grave error occurs that does not permit continuing useful processing. If the firmware calls this function, this indicates a software error in the user program. This function is not to return to the firmware!

FatalError			**Grave Error**
Transfer	File Line Errcb_ptr	DP_ERROR_FILE uint16_t VPC3_ERRCB_PTR	Filename Source code line Specific Error
Return			**Function must not return!**

Figure 4-49 : Function Fatal_ERROR

DP_ERROR_FILE		
DP_USER	0x10	
DP_IF	0x20	
DP_ISR	0x30	
DP_FDL	0x40	
DP_C1	0x50	
DP_C2	0x60	

Figure 4-50 : Description DP_ERROR_FILE

Notes:

DPV1 Extensions 5

5.1 Functional Description of the DPV1 Services

When the firmware is initialized, the DPV1 services are initialized also. If the DPV1 indications are to be processed in the polling mode, the application program has to cyclically call the macros VPC3_POLL_IND_FDL_IND() and VPC3_POLL_IND_POLL_END_IND() in the main loop. If the DPV1 indications are to be processed in the interrupt mode, the application program has to call the macros VPC3_GET_IND_FDL_IND() and VPC3_GET_IND_POLL_END_IND() in the interrupt routine.

5.1.1 Initiate (MSAC_C2)

- ♦ In the answer to an Initiate REQ PDU (on SAP 49), the firmware sends a free SAP (0..48) in the immediate response. This SAP (**S**ervice **A**ccess **P**oint) has been made available previously as response.
- ♦ The RM (**R**esource **M**anager) searches for a new free SAP, and makes it available as next response for SAP 49.
- ♦ The firmware calls the function msac_c2_initiate_req. The SAP that is to be used is transferred as parameter. In the function msac_c2_initiate_req, the application program can check the API and SCL, for example.
- ♦ If msac_c2_initiate_req was acknowledged positive, the SAP is marked as assigned.
- ♦ The SAP used is opened; via this SAP, the Initiate RES PDU is transmitted.

5.1.2 Abort (MSAC_C2)

The cancellation can be activated either by the local user via a function or via the response data, or by the master via a message.

- ♦ The FW closes the communication SAP
- ♦ The SAP is marked as free
- ♦ The function msac_c2_abort_ind is called. This only happens if the user has not requested a cancellation.

5.1.3 Read (MSAC_C1 and MSAC_C2)

- ♦ The firmware package calls the function dpv1_read_req as soon as a Read.req was received.
- ♦ If the data has been made available, or if an error was signalled, the reply is sent to the master.

5.1.4 Write (MASC_C1 and MSAC_C2)

- The firmware package calls the function dpv1_write_req as soon as a Write.req was received
- If the data has been processed, or if an error was signaled, the reply is sent to the master.

5.1.5 Data Transport (MSAC_C2)

- The firmware package calls the function msac_c2_data_transport_req as soon as a Data_Transport.req was received.
- If the response data was made available, or if an error was signaled, the reply is sent to the master.

5.1.6 Diagnosis, Alarms, and Status Messages in the case of DPV1

In DPV1, an alarm- and status model is defined. The alarms and status messages are transmitted via a device-related diagnosis. For that reason, The DPV1 slave is to use the device-related diagnoses only in this sense. The alarm is acknowledged by the master and the user enter the alarm diagnostic to the alarm state machine. The status message isn't acknowledge by the master. The user set the status message directly in the diagnostic buffer. The DPV1 slave can continue using the id-related and channel-related diagnoses, as described in the DP standard. The application program may write to the diagnostic data as is the case with the DP slave. In addition, the user can enter status messages in the diagnostic buffer. In DPV1, the static diagnosis has a special meaning: with static diagnosis, the slave signals that it is logically not ready to make useful data available. This is the case, for example, if a sensor was correctly parameterized and configured, but has not yet been set to its measuring range via the MSAC_C1 channel. If the slave can supply useful data, it removes the static diagnosis.

5.1.7 Error Handling

If the application detects an error while processing a user function, it writes the Error Code 1 and 2 according to the structure below to the response buffer that was transferred to it previously, and returns the value DPV1_NOK. The firmware fills in the function number and the decode field.

DPV1_NEG_RES_PDU		**Error Response Block**
Function_num	uint8_t	Is entered by the firmware
Err_decode	uint8_t	Always DPV1_ERR_DEC_DPE
Err_code1	uint8_t	DPV1 Error Code
Err_code2	uint8_t	User-specific

Figure 5-1 : Error Response Block

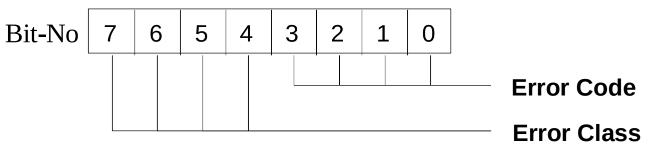


Figure 5-2 : Error Code / Error Class

Error_Class	Meaning	Error_Code
0 to 9	Reserved *)	
10	Application	0 = read error 1 = write error 2 = module failure 3 to 7 = reserved *) 8 = version conflict 9 = feature not supported 10 to 15 = user specific
11	Access	0 = invalid index 1 = write length error 2 = invalid slot 3 = type conflict 4 = invalid area 5 = state conflict 6 = access denied 7 = invalid range 8 = invalid parameter 9 = invalid type 10 to 15 = user specific
12	Resource	0 = read constrain conflict 1 = write constrain conflict 2 = resource busy 3 = resource unavailable 4 to 7 = reserved *) 8 to 15 = user specific
13 to 15	User specific	

Figure 5-3 : Error Code / Error Class

*) Reserved Error_Codes are intended to be passed unchanged to the user.

Defines for Error Code / Error Class in the firmware:

Error Class		
Reserved	0 – 9	Reserved
DPV1_ERRCL_APPLICATION	10	Error on application level
DPV1_ERRCL_ACCESS	11	Access error
DPV1_ERRCL_RESSOURCE	12	Resource error
DPV1_ERRCL_USER	13 (-15)	Free for application

Figure 5-4 : Error Class

Error_Code for Error_Class DPV1_ERRCL_APPLICATION		
DPV1_ERRCL_APP_READ	0	Read error
DPV1_ERRCL_APP_WRITE	1	Write error
DPV1_ERRCL_APP_MODULE	2	Module error
Reserved	3-7	reserved
DPV1_ERRCL_APP_VERSION	8	Version conflict
DPV1_ERRCL_APP_NOTSUPP	9	Not supported
DPV1_ERRCL_APP_USER	10 (-15)	Free for application

Figure 5-5 : Error Code for Application Error Class

Error_Code for Error_Class DPV1_ERRCL_ACCESS		
DPV1_ERRCL_ACC_INV_INDEX	0	Impermissible index
DPV1_ERRCL_ACC_WRITE_LEN	1	Write length wrong
DPV1_ERRCL_ACC_INV_SLOT	2	Impermissible slot
DPV1_ERRCL_ACC_TYPE	3	Type conflict
DPV1_ERRCL_ACC_INV_AEREA	4	Impermissible area
DPV1_ERRCL_ACC_STATE	5	State conflict
DPV1_ERRCL_ACC_ACCESS	6	Access not permitted
DPV1_ERRCL_ACC_INV_RANGE	7	Impermissible range
DPV1_ERRCL_ACC_INV_PARAM	8	Impermissible parameter
DPV1_ERRCL_ACC_INV_TYPE	9	Impermissible type
DPV1_ERRCL_ACC_USER	10 (-15)	Free for application

Figure 5-6 : Error Code for Access Error Class

Error_Code for Error_Class DPV1_ERRCL_RESOURCE		
DPV1_ERRCL_RES_READ_CONSTRAIN	0	Read constrain conflict
DPV1_ERRCL_RES_WRITE_CONSTRAIN	1	Write constrain conflict
DPV1_ERRCL_RES_BUSY	2	Resource busy
DPV1_ERRCL_RES_UNAVAIL	3	Resource unavailable
Reserved	4 – 7	reserved
DPV1_ERRCL_RES_USER	8 (- 15)	Free for application

Figure 5-7 : Error Code for Resource Error Class

5.2 Initialization

5.2.1 Settings for DPV1 in the DpCfg.h

The user connects the different services via #define in “cfg.h”, so that the program code is adapted to the required services respectively.

Service	
#define DP_MSAC_C1	1: Activation of the functionality for the expansion services of the Class 1 master. 0: not activated
#define DP_MSAC_C2	1: Activation of the functionality for the expansion services of the Class 2 master. 0: not activated
#define DP_ALARM	1: Activation of the functionality for the expansion services of the alarm mode. 0: not activated
#define DPV1_IM_SUPP	1: Activation of the functionality for the expansion services of the I&M functionality. 0: not activated

Figure 5-8 : PROFIBUS Services

Settings for MSAC_C2 Service		
#define DP_MSAC_C2_Time		Enables timecontrol for C2 services
#define C2_NUM_SAPS	uint8_t	Number of SAPs that the firmware makes available for MSAC_C2 Connections
#define C2_LEN	uint8_t	MSAC_C2 PDU length of the C2-SAP (20...244)
#define C2_FEATURES_SUPPORTED_1	uint8_t	= 0x01 (MSAC_C2_READ and MSAC_C2_WRITE supported)
#define C2_FEATURES_SUPPORTED_2	uint8_t	= 0x00
#define C2_PROFILE_FEATURES_1	uint8_t	Profile or vendor specific
#define C2_PROFILE_FEATURES_2	uint8_t	Profile or vendor specific
#define C2_PROFILE_NUMBER	uint16_t	Profile or vendor specific

Figure 5-9 : Settings for MSAC_C2 Service

Settings for MSAC_C1 Service		
#define C1_LEN	uint8_t	Length of MSAC_C1 Data (4..244 Bytes)

Figure 5-10 : Settings for MSAC_C1

Settings for MSAC_C1 Alarm	
#define DP_ALARM_OVER_SAP50	1: The master handles the Alarm Acknowledge over SAP 50 0: The master handles the Alarm Acknowledge over SAP 51

Figure 5-11 : Settings for MSAC_C1_Alarm

Mandatory settings in the VPC3+:

	Mode Register 0, High-Byte, Address 07H (Intel):
Bit 10	**User_Time_Base:** Timebase of the cyclical User_Time_Clock-Interrupt 0 = The User_Time_Clock-Interrupt occurs every 1 ms. **1 = The User_Time_Clock-Interrupt occurs every 10 ms. (mandatory DPV1)**

Figure 5-12 : Mode Register

Enable following interrupts:

	Interrupt-Mask-Register, Low-Byte, Address 04H (Intel):
Bit 4	**User_Timer_Clock:** The time base for the User_Timer_Clocks has run out (1 /10ms).
Bit 2	**Baudrate_Detect:** The VPC3+ has left the ‘Baud_Search state’ and found a baud rate.

Figure 5-13 : Interrupt Mask Register

	Interrupt Mask Register 0, High-Byte, Address 05H (Intel):
Bit 15	**FDL_Ind:** The VPC 3+ has received an acyclic service request and made the data available in an indication buffer.
Bit 14	**Poll_End_Ind:** The VPC 3+ have send the response to an acyclic service.

Figure 5-14 : Interrupt Mask Register

During the initialization the SAP-list will be generated (dp_fdl.c). Each entry in the SAP list consist of 7 bytes. The pointer at address 17H contains the segment base address of the first element of the SAP list. The last element in the list is always indicated with FFH. If the SAP list shall not be used, the first entry must be FFH, so the pointer at address 17H must point to a segment base address location which contains FFH.

The MSAC_C2 service is enabled after VPC3_START() and the MSAC_C1 is enabled with DPV1_Enable in the Set_Param telegram.

Function	Master SAP	Slave SAP	Service
MSAC_C1	51	50 or 51	Alarm_Ack
MSAC_C1	51	51	READ/WRITE
MSAC_C2	50	49	Initiate.req
MSAC_C2	50	48 .. 0	Abort, Read/Write, Data_Transfer

Figure 5-15 : SAPs for acyclic services

Structure of SAP-List entry:

Byte	Bit Position								Designation
	7	6	5	4	3	2	1	0	
0									SAP_Number
1									Request_SA
2									Request_SSAP
3									Service_Supported
4									Ind_Buf_Ptr[0]
5									Ind_Buf_Ptr[1]
6									Resp_Buf_Ptr

	SAP-List entry:
Byte 0	**Response_Sent:** Response-Buffer sent 0 = no Response sent 1 = Response sent **SAP_Number:** 0 – 63 In DP-Mode the SAPs 53, 55-62 are used for cyclic communication.
Byte 1	**Request_SA:** The source address of a request is compared with this value. At differences, the VPC 3+ response with No-Service-Activated (RS). The default value for this entry is 7FH.
Byte 2	**Request_SSAP:** The source SAP of a request is compared with this value. At differences, the VPC 3+ response with No-Service-Activated (RS). The default value for this entry is 7FH.
Byte 3	**Service_Supported:** Indicates the permitted FDL service. 00 = all FDL services allowed
Byte 4	**Ind_Buf_Ptr[0]:** pointer to indication buffer 0
Byte 5	**Ind_Buf_Ptr[1]:** pointer to indication buffer 1
Byte 6	**Resp_Buf_Ptr:** pointer to response buffer

Figure 5-16 : SAP list entry

Example of SAP-list:

SAP							Service
31	7F	7F	0B	5C	5C	5B	Initiate_Req (Resource Manager)
30	07	7F	0B	5C	5C	5C	MSAC_C2 channel 1
2F	07	7F	0B	63	63	63	MSAC_C2 channel 2
33	7F	7F	0B	6A	6A	6A	MSAC_C1 channel
FF	00	00	00				

Figure 5-17 : Example of SAP list (after START_VPC3())

In addition an indication and response buffers are needed. Each buffer consists of a 4 byte header for the buffer management and a data block of configurable length.

Byte	Bit Position								Designation
	7	6	5	4	3	2	1	0	
0	USER	IND	RESP	INUSE					Control
1									Max_Length
2									Length
3									Function Code

	SAP-List entry:
Byte 0	**Control:** bits for buffer management USER buffer assigned to user IND indication data included in buffer RESP response data included in buffer INUSE buffer assigned to VPC 3+
Byte 1	**Max_Length:** length of buffer
Byte 2	**Length:** length of data included in buffer
Byte 3	**Function Code:** function code of the telegram

Figure 5-18 : Buffer Header

5.3 DP-V1 Callback Functions

Callback functions are functions that the DPV1 state machine has to make available for the user application. Via the return value, the user controls whether he has completed the function successful, or whether he has completed the function with error, or he wanted to cancel the connection. The callback functions are handled in the file DpV1.c.

Return Values of the Callback Functions	
DPV1_OK	The function was completed successfully
DPV1_NOK	An error occurred. The user entered more detailed information about the error in the error block for this channel (refer to chapter Error Handling).
DPV1_DELAY	The application program is processing a request asynchronously.
DPV1_ABORT	The user wants to cancel the affected C2 connection. Previously, the user has preprocessed the abort PDU in the ASIC memory area.

Figure 5-19 : Return Value of Callback Function

Which return values are permitted respectively is provided with the individual functions.

5.3.1 Dpv1_Msac2InitiateReq (MSAC_C2)

The firmware calls this functon if a master wants to establish a MSAC_C2 connection.

DPV1_RET_VAL Dpv1_Msac2InitiateReq(uint8_t bSapNr, INITIATE_REQ_PDU_PTR, MSG_HEADER_PTR psMsgHeader,VPC3_UNSIGNED8_PTR pToDpv1Data)		
Function	Establish a C2 connection	
Parameter	bSapNr	Address of the slave
	psInitiateReq	Local copy of Initiate.req telegram
	psMsgHeader	Pointer to message header
	pToDpv1Data	Pointer to DPV1 data
Return Value	DPV1_OK DPV1_NOK DPV1_DLAY	See DPV1_RET_VAL

Figure 5-20 : Function Dpv1_Msac2_InitiateReq

When this function is called, the parameter PDU points to the structure MSAC_C2_INITIATE_REQ_PDU. When leaving the function, the user program has to have preprocessed the buffer according to the structure MSAC_C2_INITIATE_RES_PDU. The user is supported with the function MSAC_C2_INITIATE_REQ_TO_RES; it generates the response structure from the request structure. This applies only if the slave is the endpoint of the connection. If the macro MSAC_C2_INITIATE_REQ_TO_RES returns the value DPV1_NOK, the PDU that was received remains unchanged. The user has to either make the evaluation himself, or reject the request for establishing a connection.
The firmware sends the response PDU when the application program leaves the function with DPV1_OK. If the application program can't establish the connection (for example, profile is not supported), the application program has to fill in the response PDU according to the structure DPV1_ABORT_PDU, and exit the function with DPV1_ABORT. The firmware will then set the correct function number, and send the PDU as response. In this case, the firmware does not open the connection, and marks the corresponding SAP as free again. The request for establishing a connection may also be refused with negative response data (DPV1_ERROR_RES).

Comment: The application is not to change the function number received.

DPV1_INITIATE_REQ_PDU		Initiate Request Structure
function_num	uint8_t	0x57
reserved1	uint8_t	Reserved byte
reserved2	uint8_t	Reserved byte
reserved3	uint8_t	Reserved byte
send_timeout	uint16_t	Time control for MSAC_C2
features_supported1	uint8_t	0x01 (Read/Write service)
features_supported2	uint8_t	Reserved
profile_features_supported1	uint8_t	Profile-,vendor specific
profile_features_supported2	uint8_t	Profile-,vendor specific
profile_ident_number	uint16_t	Vendor specific
s_type	uint8_t	
s_len	uint8_t	
d_type	uint8_t	
d_len	uint8_t	
addr_data	BYTE[s_len + d_len]	Structure according to DPV1_INITIATE_SUB_PARAM

Figure 5-21 : Structure DPV1_INITIATE_REQUEST

S-Type:
This subparameter indicates the presence (S-Type=1) of the optional Network/MAC address in the Add_Addr_Param of the source.

S-Len:
This subparameter indicates the length of the S_Addr subparameter.

D-Type:
This subparameter indicates the presence (D-Type=1) of the optional Network/MAC address in the Add_Addr_Param of the destination.

D-Len:
This subparameter indicates the length of the D_Addr subparameter.

addr_data:
Contains the additional address information of the source and of the destination.

DPV1_INITIATE_RES_PDU		Initiate Response Structure
function_num	uint8_t	0x57
max_len_data_unit	uint8_t	Length data unit
features_supported1	uint8_t	0x01 (Read/Write service)
features_supported2	uint8_t	Reserved

profile_features_supported1	uint8_t	Profile-,vendor specific
profile_features_supported2	uint8_t	Profile-,vendor specific
profile_ident_number	uint16_t	Vendor specific
s_type	uint8_t	See above
s_len	uint8_t	See above
d_type	uint8_t	See above
d_len	uint8_t	See above
addr_data	BYTE[s_len + d_len]	Structure according to DPV1_INITIATE_SUB_PARAM

Figure 5-22 : Structure DPV1_INITIATE_RESPONSE

addr_data[]		
S_api	uint8_t	
S_reserved	uint8_t	
S_net_addr	uint8_t[6]	
S_mac_addr	uint8_t[]	
D_api	uint8_t	
D_reserved	uint8_t	
D_net_addr	uint8_t[6]	
D_mac_addr	uint8_t[]	

Figure 5-23 : Structure addr_data

S_API:
This subparameter identifies the application process instance of the source.

S_Network_Address: (S-Type=1)
This subparameter identifies the network address of the source according to ISO/OSI-Network addresses.

S_MAC_Address: (S-Type=1)
This subparameter identifies the MAC_Address of the source.
D_api:
This subparameter identifies the application process instance of the destination.

D_Network_Address: (D-Type=1)
This subparameter identifies the network address of the destination according to ISO/OSI-Network addresses.

D_MAC_Address: (D-Type=1)
This subparameter identifies the MAC_Address of the destination.

5.3.2 MSAC_C2_INITIATE_REQ_TO_RES (MSAC_C2)

This function relieves the application program of copying the data that is located at different locations at the initiate request and the response PDU. In addition, standard settings are entered in the response PDU.

MSAC_C2_INITIATE_REQ_TO_RES			
Transfer	PDU	MSAC_C2_INITIATE_REQ_PDU *	Request PDU
Return	DPV1_OK DPV1_NOK		Response PDU was generated The user has to handle the Response PDU himself since the device is not the endpoint of the connection. The PDU that has been transferred is not changed.

Figure 5-24 : Function MSAC_C2_INITIATE_REQ_TO_RES

Function Description:

- ♦ A check is made in the connection buffer whether the endpoint (D type = 0) of a connection has been reached. Only then will the response PDU be generated; that is, the buffer that was received is changed.
- ♦ The following response PDU is generated:
  - ▪ As length for the PDU, the length entry for the MSAC_C2 PDU transferred with vpc3_init() is used.
  - ▪ Only READ and WRITE is specified for supported services
  - ▪ The profile attributes and the profile number are set to default values (defined in dp_cfg.h).
  - ▪ The data for destination- and source addressing is copied from the request PDU and entered in the response PDU; destination and source are exchanged.

5.3.3 Dpv1_Msac2AbortInd

The firmware calls this function if a MSAC_C2 connection was aborted by the master, or the firmware detects a reason for canceling it (for example, timeout). A MSAC_C1 connection is coupled to the processing mode (cyclical state machine) of the slave. In the case of LEAVE_DATA_EXCHANGE, the MSAC_C1 connection is cancelled automatically.

USER_C2_ ABORT_IND		ABORT Indication Callback Function	
Transfer	SAP PDU	uint8_t DPV1_PTR *	SAP number
Return	DPV1_OK		See above

Figure 5-25 : Function USER_C2_ABORT_IND

DPV1_ABORT_PDU		Abort Structure
function_num	uint8_t	
Subnet	uint8_t	
instance_reason	uint8_t	

Figure 5-26 : Function DPV1_ABORT_PDU

Subnet		
MSAC_C2_SUBNET_NO	0	
MSAC_C2_SUBNET_LOCAL	1	
MSAC_C2_SUBNET_REMOTE	2	

Figure 5-27 : Description Subnet

Instance		
MSAC_C2_INSTANCE_FDL	0x00	
MSAC_C2_INSTANCE_MSAC_C2	0x10	
MSAC_C2_INSTANCE_USER	0x20	
MSAC_C2_INSTANCE_RESERVED	0x30	

Figure 5-28 : Description Instance

reason		
MSAC_C2_ABT_SE	0x01	Sequence error
MSAC_C2_ABT_FE	0x02	Invalid request PDU received
MSAC_C2_ABT_TO	0x03	Timeout of the connection
MSAC_C2_ABT_RE	0x04	Invalid response PDU received
MSAC_C2_ABT_IV	0x05	Invalid service from USER
MSAC_C2_ABT_STO	0x06	Send_Timeout requested was too small
MSAC_C2_ABT_IA	0x07	Invalid additional address information
MSAC_C2_ABT_OC	0x08	waiting for FDL_DATA_REPLY.con
MSAC_C2_ABT_RES	0x0F	Resource error

Figure 5-29 : Description Reason

5.3.4 Dpv1_ReadReq (MSAC_C1+MSAC_C2)

The firmware calls this function when a read request is pending.

DPV1_RET_VAL Dpv1_ReadReq(uint8_t bSapNr, MSG_HEADER_PTR psMsgHeader,VPC3_UNSIGNED8_PTR pToDpv1Data)		
Function	DP-V1 read request	
Parameter	bSapNr	PROFIBUS service access point
	psMsgHeader	Pointer to message header
	pToDpv1Data	Pointer to DPV1 data
Return Value	DPV1_OK DPV1_NOK DPV1_DELAY	See DPV1_RET_VAL

Figure 5-30 : Function Dpv1_ReadReq

The firmware calls this function when a Read request has been received. The array pToDpv1Data[] is undefined when the function is called. The application program has to fill in the array pToDpv1Data[], and enter the corresponding length in the field 'length'. The firmware handles the function number. If there is an error, the user normally provides a negative response PDU. This retains the connection. If the connection is to be cancelled also, an ABORT PDU is to be generated.

DPV1_READ_PDU		**Read Structure**
Function_num	uint8_t	0x5E
Slot_num	uint8_t	
Index	uint8_t	
Length	uint8_t	
Pdu_data	uint8_t[]	

Figure 5-31 : Description DPV1_READ_PDU

Example for Read Processing:

- Read.req(length ≤ 40) for a data set with the length 40 octets => the length indicated in the request is read
- Read.req(length > 40) for a data set with the length 40 octets => the genuine length of the data set (40 bytes) is read

5.3.5 DpV1_WriteReq (MSAC_C1+MSAC_C2)

The firmware calls this function if a write request was received. The firmware manages the function number. If there is an error, the user normally sets up a negative response PDU. This retains the connection. If the connection is to be cancelled also, an ABORT PDU is to be generated.

DPV1_RET_VAL Dpv1_WriteReq(uint8_t bSapNr, MSG_HEADER_PTR psMsgHeader,VPC3_UNSIGNED8_PTR pToDpv1Data)		
Function	DP-V1 write request	
Parameter	bSapNr	PROFIBUS service access point
	psMsgHeader	Pointer to message header
	pToDpv1Data	Pointer to DPV1 data
Return Value	DPV1_OK DPV1_NOK DPV1_DELAY	See DPV1_RET_VAL

Figure 5-32 : Function Dpv1_WriteReq

Example for Write Processing:

- Write.req(length $\leq$ 40) for a data set with the length 40 octets => the length of data indicated in the request is written, and the length is mirrored in the reply.
- Write.req(length > 40) for a data set with the length 40 octets => there is to be no writing; an error message has to be transmitted.

DPV1_WRITE_PDU		**Write Structure**
Function_num	uint8_t	0x5F
Slot_num	uint8_t	
Index	uint8_t	
Length	uint8_t	
Pdu_data	uint8_t[]	

Figure 5-33 : Description DPV1_WRITE_PDU

5.3.6 Dpv1_Msac2DataTransportReq (MSAC_C2)

The firmware calls this function if a data transport request was received. When the function is called, the array pToDpv1Data[] contains the received data. The application program has to fill the array pToDpv1Data[] with the data that is to be sent, and set the field 'length' correspondingly. The firmware handles the function number. If there is an error, the user normally sets up a negative response PDU. This retains the connection. If the connection is to be cancelled also, an ABORT PDU is generated.

DPV1_RET_VAL Dpv1_Msac2DataTransportReq(uint8_t bSapNr, MSG_HEADER_PTR psMsgHeader,VPC3_UNSIGNED8_PTR pToDpv1Data)		
Function	DP-V1 data transport request	
Parameter	bSapNr	PROFIBUS service access point
	psMsgHeader	Pointer to message header
	pToDpv1Data	Pointer to DPV1 data
Return Value	DPV1_OK DPV1_NOK DPV1_DLAY	See DPV1_RET_VAL

Figure 5-34 : Function Dpv1_Msac2DataTransportReq

DATA_TRANSPORT_PDU		**Data Transport Structure**
Function_num	uint8_t	0x51
Slot_num	uint8_t	
Index	uint8_t	
Length	uint8_t	
Pdu_data	uint8_t[]	

Figure 5-35 : Description DATA_TRANSPORT_PDU

5.4 DPV1 Alarm-Handling

The alarm and status messages will be transferred within the Ext_Diag_Data and replaces the device related diagnosis of EN 50170. The Ext_Diag_Data can consist of one, multiple or all of the following components:

- ♦ Alarm-PDU (only one)
- ♦ Status-PDU
- ♦ Identification-related diagnosis
- ♦ Channel-related diagnosis
- ♦ Revision-Number (only one)

The structure of the PDUs for alarm and status is as follows:

Byte	Description
0	Headerbyte
1	Alarm_Type / Status_Type
2	Slot_Number
3	Specifier
4 :	Diagnostic User Data

Figure 5-36 : Structure of the device-related diagnosis for alarm / status

5.4.1 Coding of the Alarm PDU

Byte	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	**0**	**0**	Block length in byte (4 to 63)					
1	**0**	Alarm Type						
2	Slot Number							
3	Seq_Nr					ACK	SPEC	
4 : 62	Diagnostic User Data							

Figure 5-37 : Alarm-Pdu

The Alarm_Type describes the alarm itself. The necessary reaction of the control application in the DPV1-Master (Class 1) is manufacturer- or application-specific.

Alarm Type	
0	Reserved
1	Diagnostic Alarm
2	Process Alarm
3	Pull Alarm
4	Plug Alarm
5	Status Alarm
6	Update Alarm
7-31	Reserved
32-126	Manufacturer specific Alarm
127	Reserved

Figure 5-38 : Coding Alarm Type

Alarm_specifier:

Coding	Designation	
00	No further differentiation	
01	Error appears and Slot disturbed	the slot generates an alarm due to an error
10	Error disappears and Slot is okay	the slot generates an alarm and indicates that the slot has no further errors
11	Error disappears and Slot is still disturbed	the slot generates an alarm and indicates that the slot has still further errors

Figure 5-39 : Coding Alarm Specifier

Add_Ack:
When setting this bit the slave indicates to the DPV1-Master (Class 1) that this alarm requires in addition to the MSAC1_Alarm_Ack a separate user acknowledgement. This can be done for instance by means of a Write service.

Seq_Nr:
By means of the Seq_Nr an unique identification of an alarm message is accomplished.

5.4.2 Coding of the Status PDU

Byte	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	**0**	**0**	Block length in byte (4 to 63)					
1	**1**	Status Type						
2	Slot Number							
3	reserved						SPEC	
4 : 62	Diagnostic User Data							

Figure 5-40 : Status Pdu

Status Type	
0	Reserved
1	Status Message
2	Modul Status
3-31	Reserved
32-126	Manufacturer specific Status
127	Reserved

Figure 5-41 : Coding Status Type

Status_specifier:

Coding	Designation
00	No further differentiation
01	Status appears
10	Status disappears
11	Reserved

Figure 5-42 : Coding Status Specifier

Coding of Modul Status

The Modul_Status contains information whether the modules/slots of a DPV1-Slave delivers valid data or not and the information whether there is a wrong module or no module in place. For each module/slot 2 bits are designated. The Modul_Status is padded to byte limits and not used bits are fixed to zero. The Modul_Status is typically generated by the device module (Slot_Number = 0).

Structure of the Modul_Status:

Byte	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	Headerbyte							
1	Status_Type = Modul_Status							
2	Slot Number = 0							
3	Specifier							
4	Modul_Status 4		Modul_Status 3		Modul_Status 2		Modul_Status 1	
:	.....							
m	Modul_Status m		Modul_Status m-1		.		.	

Figure 5-43 : Structure Modul Status

Modul Status:

Coding	Designation
00	data valid
01	data invalid: the data of the corresponding module are not valid due to an error (e.g. short circuit)
10	data invalid/wrong module: the data of the corresponding module are not valid, due to a wrong module in place
11	data invalid/no module: the data of the corresponding module are not valid, because there is no module in place

Figure 5-44 : Coding Modul Status

5.4.3 Example for Ext_Diag_Data (Alarm and Status PDU)

MSB						LSB		
7	6	5	4	3	2	1	0	
0	**0**	0	0	0	1	1	1	**Header: Device** related diagnostic
1	0	0	0	0	0	0	1	Statustype: Status Message
0	0	0	0	0	0	1	0	Slotnumber: 2 (sensor A)
0	0	0	0	0	0	0	0	Specifier: no further differentiation
0	0	0	0	0	1	0	1	Diag. User Data: average temperature
								Temperature value
								Unsigned16
0	**0**	0	0	1	0	0	1	**Header: Device** related diagnostic
0	0	0	0	0	0	1	0	Alarmtype: Process Alarm
0	0	0	0	0	0	1	1	Slotnumber: 3 (valve B)
0	0	0	0	0	0	0	1	Specifier: alarm appears
0	1	0	1	0	0	0	0	Diag. User Data: 0x50 (upper limit ex...)
								Time stamp
								4 bytes
0	**1**	0	0	0	0	1	0	**Header: Identification** related diagn.
0	0	0	0	0	0	0	1	1st Identification number with diagn.

Figure 5-45 : Example

Correspondending GSD-part:

```
;text assignments for sensor A and valve B

Unit_Diag_Area = 24-27
        Value(1) = "Minimum temperature"
        Value(2) = "Maximum temperature"
        Value(5) = "Average temperature"
Unit_Diag_Area_End

Unit_Diag_Area = 28-31
        Value(1) = "lower limit exceeded pressure"
        Value(5) = "upper limit exceeded pressure"
Unit_Diag_Area_End

Unit_Diag_Area = 8-15
        Value(2) = "senor A"
        Value(3) = "valve B"
Unit_Diag_Area_End

Unit_Diag_Area = 16-17
        Value(1) = "alarm/status appearing"
        Value(2) = "alarm/status disappearing"
Unit_Diag_Area_End
```

Since these definitions are used for both alarms and status messages their values should be different. That means different values for alarms and status messages should be used at the same position within the diagnostic field.

5.4.4 Coding of the Alarm_Ack-PDU

ALARM_ACK_PDU		
Function_num	uint8_t	0x5C
Slot_num	uint8_t	
Alarmtype	uint8_t	
Specifier	uint8_t	
Seq_Nr	uint8_t[]	

Figure 5-46 : Description ALARM_ACK_PDU

5.4.5 Alarm User Callback Functions

VPC3_SetAlarm

uint8_t VPC3_SetAlarm(ALARM_STATUS_PDU_PTR psAlarm, uint8_t bCallback)		
Function	By calling this function, the user can send alarms to the master	
Parameter	psAlarm	Pointer to alarm structure
	bCallback	0:the stack sends directly alarm data 1:the stack calls the function DpDiag_Alarm
Return Value	SET_ALARM_OK	
	SET_ALARM_AL_STATE_CLOSED	Alarm state machine not started
	SET_ALARM_ALARMTYPE_NOTSUPP	Alarm type not supported
	SET_ALARM_SEQ_NR_ERROR	The values of the transfer parameters are not in the specified value range
	SET_ALARM_SPECIFIER_ERROR	The values of the transfer parameters are not in the specified value range

Figure 5-50 : Function VPC3_SetAlarm ()

If the parameter callback is “FALSE” the alarm will be send directly. If the parameter callback is “TRUE” the alarm will be send over the function user_alarm (dp_user.c). In this function the user can add e.g. ModuleStatus or Channel related diagnostic.

Acknowledge Alarm

void DpDiag_AlarmAckReq(ALARM_STATUS_PDU_PTR psAlarm)		
Function	The slave acknowledges an alarm to the user that was set previously: The slave receives the acknowledgement in DPV1 operation from the parameterization master, and tranfers it to the user.	
Parameter	psAlarm	Pointer to alarm structure
Return Value	None	

Figure 5-51 : Function DpDiag_AlarmAckReq()

Notes:

DPV2 Services 6

6.1 Isochron Mode (IsoM)

6.1.1 General

The IsoM synchronize DP-Master, DP-Slave and DP-Cycle. The isochron cycle time starts with the transmission of the SYNCH telegram by the IsoM Master. If the VPC 3+ supports the IsoM, a **synchronization signal at Pin 13** is generated by reception of a SYNCH telegram.

Byte	Bit Position								Designation
	7	6	5	4	3	2	1	0	
0	0	0	1	0	1	0	0	0	Control_Command
1	1	0	0	0	0	0	0	0	Group_Select

Figure 6-1 : SYNCH telegram

There are two operation modes for cyclic synchronization available in VPC3+:

- **Isochron Mode:** Each SYNCH telegram causes an impulse on the SYNC output and a New_GC_Command interrupt.

- **Poor Sync:** A Data_Exchange telegram no longer causes an DX_Out interrupt immediately, rather the event is stored in a flag. By a following SYNCH message reception, the DX_Out interrupt and a synchronization signal are generated at the same time. Additionally a New_GC_Command interrupt is produced, as the SYNCH telegram behaves like a regular Global_Control telegram to the DP state machine. If no Data_Exchange telegram precedes the SYNCH telegram, only the New_GC_Command interrupt is generated.

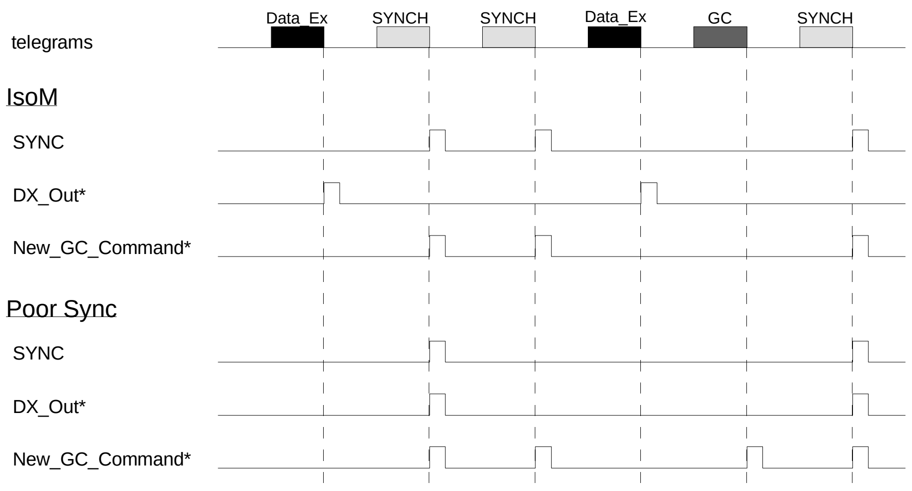


Figure 6-2 : SYNC-signal and interrupts for synchronization modes

6.1.2 Isochron Mode

Settings for Isochron mode in the DpCfg.h

The user connects the different services via #define in “DpCfg.h”, so that the program code is adapted to the required services respectively. SYNC_Ena in Mode Register 2 must be set. Furthermore the polarity (SYNC_Pol) can be adjusted. Sync_PW Register contains a multiplicator with base of 1/12 µs to adapt the pulse width. Additionally the Spec_Clear_Mode in Mode Register 0 must be set.

Service	
#define DP_ISOCHRON_MODE	Activation of the functionality for the expansion services of the isochron mode.

Figure 6-3 : PROFIBUS Services

Settings for Isochron Mode		
#define SYNCH_PULSEWIDTH	uint8_t	Width of Synchpulse in 1/12µs

Figure 6-4 : Settings for Isochron Mode

	Mode Register 2, Address 0CH:
bit 7 - 5	
bit 4	**SYNC_Ena:** Enable generation of SYNC pulse (for Isochron Mode only) 0 = SYNC pulse generation is disabled (default). **1 = SYNC pulse generation is enabled.**
bit 3 - 0	

Figure 6-5 : General Slave Parameter

	Mode Register 0, High-Byte, Address 07H (Intel):
Bit 15 - 14	
Bit 13	**Spec_Clear_Mode:** Special Clear Mode (Fail Safe Mode) 0 = No special clear mode. **1 = Special clear mode. VPC3+ will accept data telegrams with data unit = 0**
Bit 12 - 8	

Figure 6-6 : Coding of Mode Register 0, High-Byte

Settings in Set_Param telegram are shown below (Master configuration).

Byte	Bit Position								Designation
	7	6	5	4	3	2	1	0	
0			Sync_Req = 0	Freeze_Req = 0					Station_Status
1									WD_Fact_1
2									WD_Fact_2
3									minT$_{SDR}$
4									Ident_Number_High
5									Ident_Number_Low
6	Group_8 = 0								Group_Ident
7		Fail_Safe = 1							DPV1_Status_1
8									DPV1_Status_2
9				IsoM_Req = 1					DPV1_Status_3
10 : 246									User_Prm_Data

Figure 6-7 : Format of Set_Param for IsoM

6.1.3 Poor Sync Mode

Settings for Poor Sync mode in the DpCfg.h

DX_Int_Port in Mode Register 2 must be set and SYNC_Ena need not to be set. The setting of polarity and pulse width are the same as by IsoM. Also the Fail Safe Mode must be supported.

Service	
#define DP_ISOCHRON_MODE	Activation of the functionality for the expansion services of the isochron mode.

Figure 6-8 : PROFIBUS Services

Settings for Isochron Mode		
#define SYNCH_PULSEWIDTH	uint8_t	Width of synch pulse in 1/12µs

Figure 6-9 : Settings for Isochron Mode

	Mode Register 2, Address 0CH:
bit 7 - 5	
bit 4	**SYNC_Ena:** Enable generation of SYNC pulse (for Isochron Mode only) **0 = SYNC pulse generation is disabled (default).** 1 = SYNC pulse generation is enabled.
bit 3	**DX_Int_Port:** Port mode for Dataexchange Interrupt 0 = DX Interrupt not assigned to port DATA_EXCH (default). **1 = DX Interrupt (synchronized to GC-SYNC) assigned to port DATA_EXCH.**
bit 2 - 0	

Figure 6-10 : General Slave Parameter

	Mode Register 0, High-Byte, Address 07H (Intel):
Bit 15 - 14	
Bit 13	**Spec_Clear_Mode:** Special Clear Mode (Fail Safe Mode) 0 = No special clear mode. **1 = Special clear mode. VPC3+ will accept data telegrams with data unit = 0**
Bit 12 - 8	

Figure 6-11 : Coding of Mode Register 0, High-Byte

Settings in Set_Param telegram are shown below (Master configuration).

Byte	Bit Position								Designation
	7	6	5	4	3	2	1	0	
0			Sync_Req = 1	Freeze_Req = 1					Station_Status
1									WD_Fact_1
2									WD_Fact_2
3									minT_{SDR}
4									Ident_Number_High
5									Ident_Number_Low
6	Group_8 = 1								Group_Ident
7									DPV1_Status_1
8									DPV1_Status_2
9									DPV1_Status_3
2 : 246									User_Prm_Data

Figure 6-12 : Format of Set_Prm for DP-Slave using isochrones cycles

In opposite to IsoM the DX_Out interrupt first generated by receiving of SYNCH telegram. If no Data_Exchange telegram received before a SYNCH occurred, no synchronization signal is generated.

6.1.4 Structured Prm-Data for Isochron Mode

Byte		Value range	Description
0	Structured Length	28	
1	Structure Type	4	
2	Slotnumber	0	
3	Reserved	0	
4	Version	1	
5 - 8	T_{BASE_DP}	375, 750, 1500 (default), 3000, 6000. All other values are reserved an shall not be used.	
9 - 10	T_{DP}	154 to 2^{16}-1	
11	T_{MAPC}	0 to255	
12 - 15	T_{BASE_IO}	375, 750, 1500 (default), 3000, 6000. All other values are reserved an shall not be used.	
16 - 17	T_I	0 to 2^{16}-1	
18 - 19	T_O	0 to 2^{16}-1	
20 - 23	T_{DX}	0 to 2^{32}-1	
24 - 25	T_{PLL_W}	1 to 2^{16}-1	
26 - 27	T_{PLL_D}	0 to 2^{16}-1	

Figure 6-13 : Structured Isochron Mode Parameter

6.2 Data-eXchange-Broadcast (DXB)

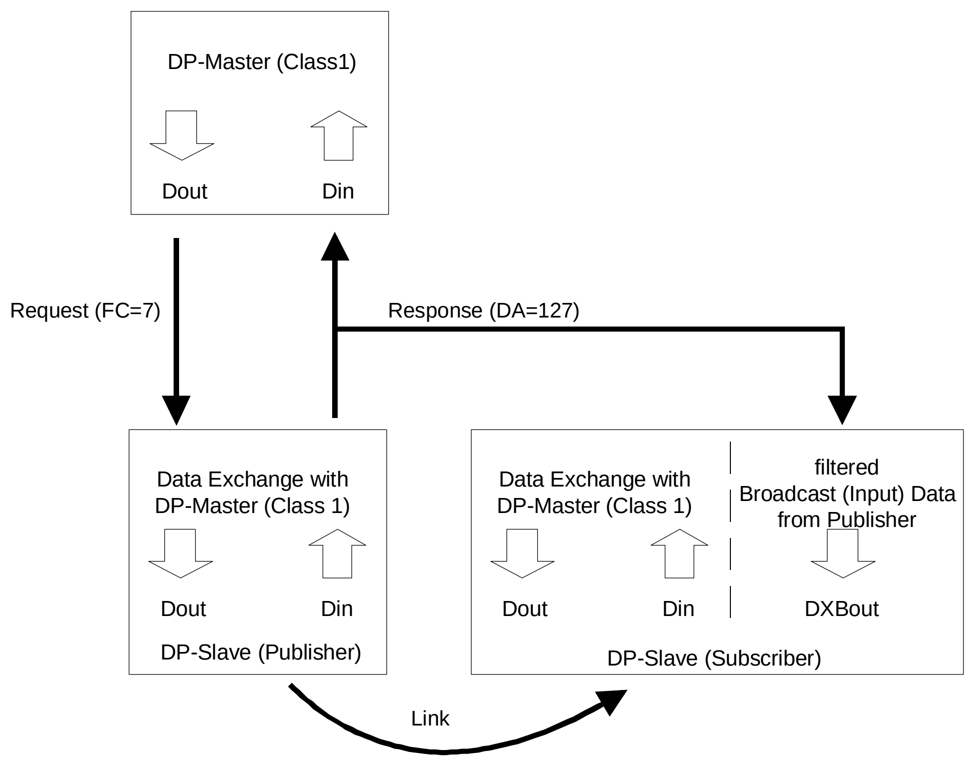

DP-Master (Class1)
Dout
Din
Request (FC=7)
Response (DA=127)
Data Exchange with DP-Master (Class 1)
Dout
Din
DP-Slave (Publisher)
Data Exchange with DP-Master (Class 1)
Dout
Din
filtered Broadcast (Input) Data from Publisher
DXBout
DP-Slave (Subscriber)
Link

Figure 6-14 : Overview DXB

The DXB mechanism enables a fast slave-to-slave communication. A slave which holds input data significant for other slaves, works as a Publisher. The Publisher can handle a special kind of Data Exchange request from the master and sends its answer as a broadcast telegram. Other slaves, that are parameterized as Subscribers, can monitor this telegram. A link is opened to the Publisher if the address of the Publisher is registered in the link table of the Subscriber. If the link were established correctly, the Subscriber can fetch the input data from the Publisher. The VPC 3+ can handle a maximum of 29 links.

6.2.1 Publisher

The VPC3+ handles the publisher mode automatically. In the firmware no adjustments need to be made. A Publisher is activated with 'Publisher_Enable = 1' in DPV1_Status_1. The time minT_{SDR} must be set to 'T_{ID1} = 37 t_{bit} + 2 T_{SET} + T_{QUI}'.

All Data_Exchange telegrams containing the function code 7 (Send and Request Data Brct) are responded with destination address 127. If Publisher mode is not enabled, these requests are ignored.

6.2.2 Subscriber

A Subscriber requires information about the links to its Publishers. These settings are contained in a DXB Linktable or DXB Subscribertable and transferred via the Structured_Prm_Data in a Set_Param or Set_Ext_Prm telegram. Each Structured_Prm_Data is treated like the User_Prm_Data and therefore evaluated by the user. From the received data the user must generate DXB_Link_Buf and DXB_Status Buf entries. The watchdog must be enabled to make use of the monitoring mechanism. This must be checked by the user.

6.2.3 Structured PRM-Data: DXB Linktable

Byte	**Bit Position**								**Designation**
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
0									Structured_Length
1	0	0	0	0	0	0	1	1	Structure_Type
2	0	0	0	0	0	0	0	0	Slot_Number
3	0	0	0	0	0	0	0	0	Reserved
4	0	0	0	0	0	0	0	1	Version
5									Publisher_Addr
6									Publisher_Length
7									Sample_Offset
8									Sample_Length
9 : 120									Further link entries

Figure 6-15 : Format of the Structured_Prm_Data with DXB-Linktable (specific link is grey scaled)

6.2.4 Structured PRM-Data: DXB Subscribertable

Byte	Bit Position								Designation
	7	6	5	4	3	2	1	0	
0									Structured_Length
1	0	0	0	0	0	1	1	1	Structure_Type
2	0	0	0	0	0	0	0	0	Slot_Number
3	0	0	0	0	0	0	0	0	Reserved
4	0	0	0	0	0	0	0	1	Version
5									Publisher_Addr
6									Publisher_Length
7									Sample_Offset
8									Dest_Slot_Number
9									Offset_Data_Area
10									Sample_Length
11 : 120									further link entries

Figure 6-16: Format of the Structured_Prm_Data with DXB-Subscribertable (specific link is grey scaled)

The user must copy the link entries of DXB-Linktable or DXB-Subscribertable, without Dest_Slot_Number and Offset_Data_Area, in the DXB_Link_Buf and set R_Len_DXB_Link_Buf. Also the user must enter the default status message in DXB_Status_Buf from the DXB-Linktable and write the appropriate values to R_Len_DXB_Status_Buf. After that, the parameterization interrupt can be acknowledged.

6.2.5 Structure of VPC3+ DXB-Link Table

Byte	Entry
0	Publisher_Addr (= 0...125)
1	Publisher_Length (= 1...244)
2	Sample_Offset (= 0...243)
3	Sample_Length (= 1..244)
...	...
m - 3	Publisher_Addr (= 0..125)
m - 2	Publisher_Length (= 1..244)
m - 1	Sample_Offset (= 0..243)
m	Sample_Length (= 1..244)

Figure 6-17 : Structure of VPC3+ DXB_LINK_TABLE

6.2.6 Structure of VPC3+ DXB Link Status

Byte	Bit Position								Designation
	7	6	5	4	3	2	1	0	
0	0	0	Block_Length						Header_Byte
1	1	0	0	0	0	0	1	1	Status_Type
2	0	0	0	0	0	0	0	0	Slot_Number
3	0	0	0	0	0	0	0	0	Status_Specifier
4									Publisher_Addr
5	Link_ Failure	Link_ Error	0	0	0	0	0	Data_ Exist	Link_Status
6 : 61									Further link entries

	Link_Status:
Bit 7	**Link_Status :** 1 = active, valid data receipt during last monitoring period 0 = not active, no valid data receipt during last monitoring period (DEFAULT)
Bit 6	**Link_Error:** 0 = no faulty Broadcast data receipt (DEFAULT) 1 = wrong length, error occurred by reception
Bit 0	**Data_Exist:** 0 = no correct Broadcast data receipt during current monitoring period (DEFAULT) 1 = error free reception of Broadcast data during current monitoring period

Figure 6-18 : DXB_Link_Status_Buf (specific link is grey scaled)

6.2.7 Functional Description of the DXB Services

VPC3_SET_DXB_LINK_TABLE_LEN (uint8_t link_len)		
Function	Set the length of the DXB-Link Table buffer	
Parameter	Length of DXB-Link Table buffer	
Return Value	None	

Figure 6-19 : Function VPC3_SET_DXB_LINK_TABLE_LEN

uint8_t VPC3_GET_DXB_LINK_TABLE_LEN ()		
Function	Get the length of the DXB-Link Table buffer	
Parameter	None	
Return Value	Length of DXB-Link Table buffer	

Figure 6-20 : Function VPC3_GET_DXB_LINK_TABLE_LEN

VPC3_UNSIGNED8_PTR VPC3_GET_DXB_LINK_TABLE_BUF_PTR ()		
Function	Fetch buffer pointer of the DXB-Link Table buffer.	
Parameter	None	
Return Value	pointer to the DXB-Link Table buffer	

Figure 6-21 : Function VPC3_GET_DXB_LINK_BUF_PTR

VPC3_SET_DXB_LINK_STATUS_LEN (uint8_t status_len)		
Function	Set the length of the DXB-Link Status buffer	
Parameter	Length of DXB-Link Status buffer	
Return Value	None	

Figure 6-22 : Function VPC3_SET_DXB_LINK_STATUS_LEN

uint8_t VPC3_GET_DXB_LINK_STATUS_LEN ()		
Function	Get the length of the DXB-Link Status buffer	
Parameter	None	
Return Value	Length of DXB-Link Status buffer	

Figure 6-23 : Function VPC3_GET_DXB_LINK_STATUS_LEN

VPC3_UNSIGNED8_PTR VPC3_GET_DXB_LINK_STATUS_BUF_PTR(void)		
Function	Fetch buffer pointer of the DXB-Link Status buffer.	
Parameter	None	
Return Value	pointer to the DXB-Link Status data buffer	

Figure 6-24 : Function VPC3_GET_DXB_LINK_STATUS_BUF_PTR()

void VPC3_SubscriberToLinkTable (PRM_SUBSCRIBER_TABLE_PTR psDxb, uint8_t bNrOfLinks)		
Function	Converts the dxb-subscriber table format to the dxb-link table format and initialize the VPC3+ with the dxb-link table.	
Parameter	psDxb bNrOfLinks	
Return Value	None	

Figure 6-25 : Function VPC3_SubscriberToLinkTable ()

uint8_t VPC3_CheckDxbLinkTable(void)		
Function	Checks the dxb-link table.	
Parameter	None	
Return Value	DP_OK DP_PRM_DXB_ERROR	

Figure 6-26 : Function VPC3_CheckDxbLinkTable ()

void VPC3_BuildDxbLinkStatus(void)		
Function	Generate from the dxb-link table the dxb link status table and initialize the VPC3+ with the dxb-link status table.	
Parameter	Valid DXB-Link Table	
Return Value	None	

Figure 6-27 : Function Vpc3_BuildDxbLinkStatus()

Processing Sequence

The VPC 3+ processes DXBout buffers like the Dout buffers. The only difference is, that the DXBout buffers are not cleared by the VPC 3+.

The VPC 3+ writes the received and filtered broadcast data in the DXBout buffer. The buffer contains also the Publisher_Address and the Sample_Length.

Byte	Bit Position								Designation
	7	6	5	4	3	2	1	0	
0									Publisher_Addr
1									Sample_Length
2 : 246									Sample_Data

Figure 6-28 : Structure of DXBout Buffer

VPC3_UNSIGNED8_PTR VPC3_GetDxbOutBufPtr ()		
Function	Fetch buffer pointer of the DXB output buffer.	
Parameter	None	
Return Value	Pointer to the DXB data buffer NIL, if no diagnostics buffer in the ‘U’ state	

Figure 6-29 : Function VPC3_GetDxbOutBufPtr()

Monitoring

After receiving the DXB data the Link_Status in DXB_Status_Buf of the concerning Publisher is updated. In case of an error the bit Link_Error is set. If the processing is finished without errors, the bit Data_Exist is set.

In state Data_Exchange the links are monitored in intervals defined by the parameterized watchdog time. After the monitoring time runs out, the VPC 3+ evaluates the Link_Status of each Publisher and updates the bit Link_Failure. The timer restarts again automatically.

Event	Link_Status	Link_Error	Data_Exist
WD_Time elapsed AND Data_Exist = 1	0	0	0
WD_Time elapsed AND (Data_Exist = 0 OR Link_Error = 1)	1		
faulty DXB data receipt		1	0
valid DXB data receipt		0	1

Figure 6-30 : Link_Status handling

⚠ **To enable the monitoring of Publisher-Subscriber links the watchdog timer must be enabled in the Set_Param telegram. This must be checked by user.**

Notes:

Trouble-shooting 7

Notes:

Appendix 8

8.1 Revision History

Version	Date	Remarks
V6.00	05.06.2012	First release

8 Appendix

Notes:

proficip GmbH
Einsteinstrasse 6a
91074 Herzogenaurach
Germany

Phone : +49.9132.744-200
Fax: +49.9132.744-2164

www.profichip.com

profichip®
The Clever Alternative