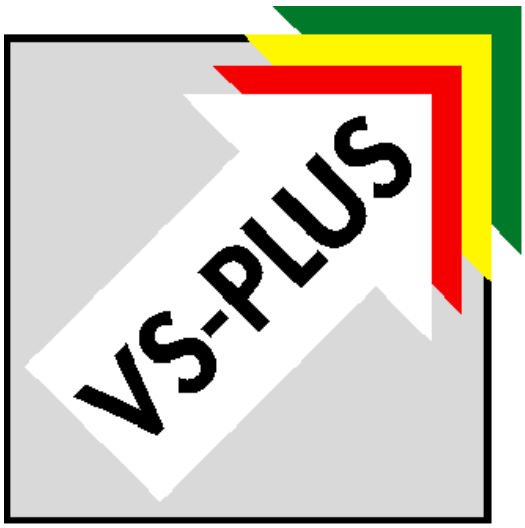
VS-PLUS IMPLEMENTATION

NEUTRAL, INDEPENDENT, STANDARDIZED



How to Implement VS-PLUS on a Controller

Implementation manual

Version of 12.03.2014

8585.0021B Implementation VS-PLUS EN v01-00-00 / [3] / 00-02-01PHe

Neue Bahnhofstrasse 160
CH-4132 Muttenz
Phone: +41 61 466 68 10
Fax +41 61 466 68 99
Mail info@vs-plus.com
http://www.vs-plus.com

Verkehrs-Systeme AG

DocName	Version	Version date	Status	Checked
8585.0021B Implementation VS-PLUS v00-00-01.docx	00-00-01	10.01.2014	Draft	Herren
8585.0021B Implementation VS-PLUS EN v00-00-04.docm	00-00-04	07.02.2014	Draft	Riedel
8585.0021B Implementation VS-PLUS EN v00-00-06.docm	00-00-06	10.02.2014	Internal check	
8585.0021B Implementation VS-PLUS EN v00-02-00.docm	00-02-00	25.02.2014	Internal check	Riedel
8585.0021B Implementation VS-PLUS EN v01-00-00.docm	01-00-00	12.03.2014	clearance	

Imprint

File: 8585.0021B Implementation VS-PLUS EN v01-00-00
Version: 01-00-00 [3]
Version data: 12.03.2014
Last save date: 12.03.2014
Number of pages: 114
Author(s): Peter Herren, Dr. Thomas Riedel

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ANNEX LIST

1 INTRODUCTION

This document describes the necessary requirements and functions in order must be implemented VS-PLUS on a controller.

1.1 Controller requirements

It is expected that the controller is able to run a fixed-time program independently. In addition, the following points must be fulfilled.

Keep and manage its own base parameters (as detectors, signal groups, conflicts, intergreens etc.)

Execute a signal programs with the correct transitions

Synchronize time with a central time base or a radio clock

Switch between programs and synchronize a program.

Switch controller on and off according to specified procedures

Provide interfaces to the "outside world" in order to exchange data and files.

Evaluate the detectors and provide their data to VS-PLUS in an aggregated state.

Provide a series of timers for VS-PLUS.

In order must be implemented VS-PLUS on the controller, an ANSI C compiler for the operating system of the controller is required.

The CPU speed should allow the VS-PLUS code to run at least once per second.

2 CONCEPT

2.1 VS-PLUS Structure

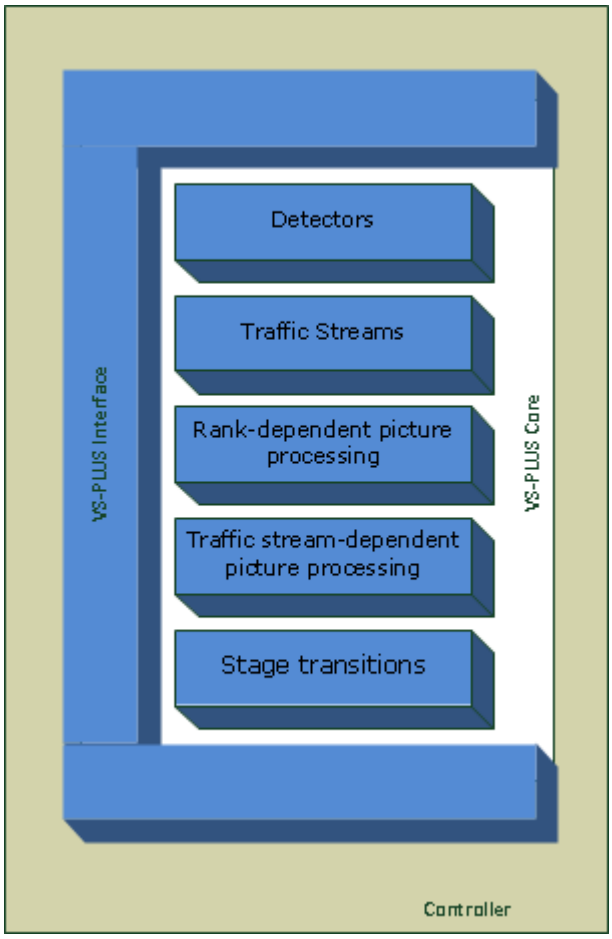


Figure 1: VS-PLUS modules inside the controller

The VS-PLUS core consists of several modules:

- Detectors
- Traffic Streams
- Rank-dependent picture processing
- Traffic stream-dependent picture processing
- Stage transitions

These modules are surrounded by the VS-PLUS interface that connects VS-PLUS to the controller.

All data exchange and communication is using this interface

2.2 General interface structure

The interface is based on functions. Depending on the situation these features can be called by VS-PLUS as well as by the controller.

That means also that for one part of the functions in VS-PLUS another part has to be implemented in the controller.

The linking of the functions, such as used in VS-PLUS and how the function is used in the controller, is done through the file "if626bas.inc" (626 refers to the VS-PLUS version 6.2.6).

Example of such a definition

```
#define f_prg()          ActualProgram()
```

f_prg() this is the function name in VS-PLUS.

ActualProgram() this is the function name in the controller.

This function name implementation has the advantage that the controller manufacturer is free for its function naming. So no adjustments in VS-PLUS are necessary between different controller manufacturers.

2.3 VS-PLUS Main

By the function VSPLUS(…) the controller calls VS-PLUS.

This function is not controlled by the "ifxxxbas.inc" file. This function name has to be called by the controller as is.

The controller controls VS-PLUS. It is in the sovereignty of the controller when to call VS-PLUS in the actual processing cycle.

How VS-PLUS is controlled is discussed in chapter 4 "VS-PLUS " (page 56).

2.4 VS-PLUS Parameters

The file format for any data transfer is the VCB format.

VS-PLUS **C**onfiguration **B**inary

Tools like VS-WorkSuite create such files. Currently, the VS-PLUS supply parameters are contained in one single file. In later versions, it should be ensured that there may also be several files.

The task of the controller is to transport this file and to make it available to VS-PLUS. The controller must also keep this file permanently in order to make sure that the supply parameters are not lost in case of power failure.

Reading a VS-PLUS parameter file into the controller comes in two steps. The VCB file is first checked whether it is valid and only in a second step the supply parameters will be deployed within VS-PLUS. After VS-PLUS has read the new supply parameters they are written back to the controller again.

A VCB file does not necessarily contain a complete supply parameter set. It can also contain only certain parts of it. In order to make sure that the supply parameter set is always complete, the whole set is always written back to the controller as a new version. This version must be kept by the controller and given to VS-PLUS in case of a restart.

In order to start VS-PLUS, there is always a complete supply parameter required. Besides this supply parameter VCB file, there are different kinds of VCB files that the controller must be able to receive and forward to VS-PLUS.

Command file: This file is a file that can contain various commands for VS-PLUS. For example, it is possible that VS-PLUS receives additional dynamic requirements by a network control. This type of files must not be stored on the device. After power failure, a new file must be transferred to the controller.

Special File: In the so called "Special File" additional program code can be transmitted to VS-PLUS. Since VS-PLUS follows standardized program steps, by this special programming file some extensions and modifications can be added. This type of files has to be stored on the controller.

Command file and Special File are VCB files. The controller must receive the files and make them available to VS-PLUS. Both files are optional. But the controller must always provide the interfaces for transmitting theses files any time.

3 VS-PLUS INTERFACE

3.1 General

The function descriptions are listed in the following chapters. The tables are structured as follows:

<table>
<tr><td colspan="2">Function as used in VS-PLUS</td><td>Function name</td></tr>
<tr><td colspan="3">Function description</td></tr>
<tr><td colspan="2" rowspan="3">Available starting with version</td><td>Compulsory name in the controller</td></tr>
<tr><td>or</td></tr>
<tr><td>Freely choosable name in the controller</td></tr>
<tr><td>☐ must be implemented</td><td>☐ obsolete</td><td>☐ for future use</td></tr>
<tr><td colspan="3">Prototype</td></tr>
<tr><td colspan="3">Parameter description</td></tr>
</table>

Table 1: VS-PLUS function definition prototype

- *Function as used in VS-PLUS:* How the function is used by VS-PLUS.
- *Function name:* Name or short name of the function.
- *Function description:* Full function description.
- *Available starting with version:* Starting with what version of VS-PLUS this function has been introduced.
- *Freely choosable name in the controller:* the function can be named differently in the control unit.
- *must be implemented:* Function must be implemented in order to have VS-PLUS correctly.

- *obsolete*: The function is no longer needed, but must be present in the interface with the specified settings.
- *for future use*: Functions that are not yet released and that will be used for future versions of VS-PLUS.
- *Prototype:* Function prototype with data types.
- *Parameter description*: Parameter description with their value ranges.

In case a controller does not provide a certain functionality (e.g. digital outputs), the functions has to be implemented anyway. But there is no function execution. Such functions have to log a line in the diary in order to tell the user that there has been no action executed.

3.2 Main Functions

The so called main functions are provided for controlling VS-PLUS. Normally they are called by the controller.

3.2.1 VS-PLUS Main

VSPLUS(…)		The VS-PLUS main function
The main function to call VS-PLUS		
VS-PLUS 2		VSPLUS
☑ must be implemented	☐ obsolete	☐ for future use
int VSPLUS(VSPSollTyp* VSPSoll, WBTyp* SchaltBild, WBReadyTyp* WunschBildBereit)		
Details in chapter 4.2 "Calling VS-PLUS"		

Table 2: VS-PLUS function: VSPLUS

3.2.2 Timer

Timers usually count any kind of waiting time.

t_twdet(x) t_twdet1(x) t_twdet2(x) t_twvs(y) t_anzt(y) t_vst(y) t_vsvort(y) t_spezt(y)		Read timer
With this function VS-PLUS reads a timer that is managed by the controller. The controller provides the necessary timers (for the necessary numbers of timers see 4.3 "Timer"). The return value is in units of 100ms. x, y: time indexes		
VS-PLUS 2.9		timer(1,x+(VSP_XX)) 1: read VSP_XX: offset constant, see 4.3 "Timer"
☑ must be implemented	☐ obsolete	☐ for future use
short timer(short funktion, short timer)		
return value: funktion: timer:	time in units of 100ms call type: 1 = read timer index	

Table 3: VS-PLUS function: timer read

m_stwdet(x,k)) m_stwdet1(x,k) m_stwdet2(x,k) m_stwvs(y,k) m_sanzt(y,k) m_svst(y,k) m_svsvort(y,k) m_sspezt(y,k)	Start timer with value k
These functions start a timer with the initial value k. The passed value k is in units of 100ms. After the start, the controller increases the value every 100ms. When the timer reaches the maximum value (32767), the timer is stopped and stays on this value. x, y: timer indices k: start value	
VS-PLUS 2.9	timer_2(2,x+(VSP_XX),k) 2: load and start VSP_XX: offset constant, see 4.3 "Timer"
☑ must be implemented	☐ obsolete ☐ for future use
short timer_2 (short funktion, short timer, short wert)	
return value: time in units of 100ms funktion: call type: 2 = load value and start timer: timer index wert: timer start value	

Table 4: VS-PLUS function: timer start

m_ltwdet (x) m_ltwdet1(x) m_ltwdet2(x) m_ltwvs(y) m_lanzt(y) m_lvst(y) m_lvsvort(y) m_lspezt(y)	Clear timer
This function clears a timer's value. The device sets the timer to 0. The timer will still be increased every 100ms. When the timer reaches the maximum value (32767), the timer is stopped and stays on this value. x, y: timer index	
VS-PLUS 2.9	timer(3,x+(VSP_XX)) 3: read VSP_XX: offset constant, see 4.3 "Timer"
☑ must be implemented	☐ obsolete ☐ for future use
short timer(short funktion, short timer)	
return value: time in units of 100ms funktion: call type: 3 = clear timer: timer index	

Table 5: VS-PLUS function: timer clear

m_altwdet(x) m_altwdet1(x) m_altwdet2(x) m_altwvs(y) m_alanzt(y) m_alvst(y) m_alvsvort(y) m_alspezt(y)	Stop timer and clear value
This function stops a time and clears the value. The controller sets the timer value to 0 and stops the timer x, y: timer index	
VS-PLUS 2	timer(4,x+(VSP_XX)) 4: stop and clear VSP_XX; offset constant, see 4.3 "Timer"
☑ must be implemented ☐ obsolete ☐ for future use	
short timer(short funktion, short timer)	
return value: time in units of 100ms funktion: call type: 4 = stop and clear timer: timer index	

Table 6: VS-PLUS function: timer stop and clear

m_atwdet (x) m_atwdet1(x) m_atwdet2(x) m_atwvs(y) m_aanzt (y) m_avst(y) m_avsvort(y) m_aspezt(y)	Stop timer
This function stops a timer. The actual value is kept. x, y: timer index	
VS-PLUS 2	timer(5,x+(VSP_XX)) 5: stop VSP_XX: offset constant, see 4.3 "Timer"
☑ must be implemented ☐ obsolete ☐ for future use	
short timer(short funktion, short timer)	
return value: time in units of 100ms funktion: call type: 5 = stop timer: timer index	

Table 7: VS-PLUS function: timer stop

3.3 Controller functions

These functions must be implemented by the controller.

3.3.1 Program functions

3.3.1.1 Actual program

f_prg()		Actual program
VS-PLUS is told the actual program number.		
VS-PLUS 2		ProgrammAktuell [translation: ProgramActual]
☑ must be implemented	☐ obsolete	☐ for future use
short ProgrammAktuell(void)		
return value:	Actual program number 0 – 255	

Table 8: VS-PLUS function: actual program

3.3.1.2 Selected program

f_prgwl()		Selected program
VS-PLUS is told a program change request. If there is no change request pending, the actual program is contained in the return value.		
VS-PLUS 2		ProgrammWahl [translation: ProgramChoice]
☑ must be implemented	☐ obsolete	☐ for future use
short ProgrammWahl(void)		
return value:	Number of the new, not yet activated program 0 – 255	

Table 9: VS-PLUS function: selected program

3.3.1.1 Cycle control

f_ur(vs,zeit)		Cycle control
VS-PLUS tells the controller the number of the traffic stream with the largest waiting time. This function is called once per second at the end of the VS-PLUS run. Tus the controller is enabled to stop calling VS-PLUS when this waiting time value has reaches a specified value. If this is the case, the controller automatically switches to cyclic control. At the same time a message is logged in order to enable to track this error.		
VS-PLUS 2		U_Kontrolle (vs, zeit) [translation: C_control]
☑ must be implemented	☐ obsolete	☐ for future use
void U_Kontrolle(short vs, short zeit)		
return value:	none	
vs:	number of traffic stream with largest waiting time 0 – VSMAX	
zeit:	waiting time in units of 100ms	

Table 10: VS-PLUS function: cycle control

3.3.1.2 Cycle second (TX)

l_zytim()()		Current cycle second
The current cycle second within the active signal program.		
VS-PLUS 2		Zykluszeit [translation: cycleSecond]
☑ must be implemented	☐ obsolete	☐ for future use
short Zykluszeit(void)		
return value:	Current cycle second in units of 100ms, rounded to seconds	

Table 11: VS-PLUS function: cycle second (TX)

3.3.1.1 Cycle time (TU)

l_umzt()		Cycle time
The cycle time of the active signal program (in units of 100ms)		
VS-PLUS 2		Umlaufzeit [translation: cycleTime]
☑ must be implemented	☐ obsolete	☐ for future use
short Umlaufzeit(void)		
return value:	Cycle time in units of 100ms	

Table 12: VS-PLUS function: cycle time (TU)

3.3.2 Detector functions

3.3.2.1 Impulse storage

l_imp(x)		Sum of rising slopes
Number of rising slopes counted by the controller, until clearing by VS-PLUS.		
VS-PLUS 2		d_imp
☑ must be implemented	☐ obsolete	☐ for future use
short d_imp(short det)		
return value:	number of rising slopes since last clear	
det:	detector channel number	

Table 13: VS-PLUS function: read sum of rising detector slopes

m_limp(x)		Clear sum of rising slopes
The rising slope counter is cleared by VS-PLUS.		
VS-PLUS 2		d_limp [translation: d_cimp]
☑ must be implemented	☐ obsolete	☐ for future use
void d_limp(short det)		
return value:	none	
det:	detector channel number	

Table 14: VS-PLUS function: clear sum of rising detector slopes

l_impss(x)		Impulse counter SS value
☐ must be implemented	☑ obsolete	☐ for future use

Table 15: VS-PLUS function: read impulse counter SS value (obsolete)

m_limpss (x)		Clear impulse counter SS value
☐ must be implemented	☑ obsolete	☐ for future use

Table 16: VS-PLUS function: clear impulse counter SS value (obsolete)

l_impab(x)		Sum of falling slopes
Number of falling slopes counted by the controller, until clearing by VS-PLUS.		
VS-PLUS 3		d_impab(1,x) [translation: d_impfall] 1: read counter (number of slopes)
☑ must be implemented	☐ obsolete	☐ for future use
short d_impab(short type, short det)		
return value:	Number of falling slopes since last clear	
type:	1: read counter (number of falling slopes)	
det:	detector channel number	

Table 17: VS-PLUS function: read sum of falling detector slopes

m_limpab(x)		Clear sum of falling slopes
Clear number of falling slopes.		
VS-PLUS 3		d_impab(2,x) [translation: d_impfall] 2: clear counter
☑ must be implemented	☐ obsolete	☐ for future use
short d_impab(short type, short det)		
return value:	number of falling slopes since last clear	
type:	2: clear counter	
det:	detector channel number	

Table 18: VS-PLUS function: clear sum of falling detector slopes

3.3.2.2 Occupancy degree

l_belga(x)		Current occupancy degree
The occupancy degree (in percent) indicates how long a detector has been occupied during the last second. (0 – 100%)		
VS-PLUS 2		d_belga [translation: d_occDeg]
☑ must be implemented	☐ obsolete	☐ for future use
short d_belga(short det)		
return value:	detector occupancy degree during the last second, in percent	
det:	detector channel number	

Table 19: VS-PLUS function: read current detector occupancy degree

l_belgg(x)		Smoothened occupancy degree
The smoothened occupancy degree is calculated over a defined interval. The formula is given in chapter 4.4.2 "Occupancy degree".		
VS-PLUS 2		d_belgg [translation: d_occSmoo]
☑ must be implemented	☐ obsolete	☐ for future use
short d_belgg(short det)		
return value:	smoothened occupancy degree, in percent (0 – 100%)	
det:	detector channel number	

Table 20: VS-PLUS function: read current smoothened detector occupancy degree

l_kvalue(x)		Load, derived from traffic situation
The actual traffic load in % (0 – 100%), calculated by the traffic situation module. The traffic situation module is a separate module that calculates traffic situations on segments basing on vehicle GPS positions.		
VS-PLUS 7		d_kvalue
☐ must be implemented	☐ obsolete	☑ for future use
short d_kvalue(short det)		
return value:	traffic load in percent (0 – 100%)	
det:	virtual detector (segment) channel number	

Table 21: VS-PLUS function: read detector load derived from traffic situation (for future use)

3.3.2.3 Occupancy

l_belza(x)		Occupancy state
Actual occupancy state of the detector		
VS-PLUS 2		d_blg [translation: d_occSta]
☑ must be implemented	☐ obsolete	☐ for future use
short d_blg(short det)		
return value: det:	0: not occupied; >0: occupied detector channel number	

Table 22: VS-PLUS function: read detector occupancy state

t_belzt(x)		Occupancy time
Actual occupancy time of the detector		
VS-PLUS 2		d_blgzt [translation: d_occTim]
☑ must be implemented	☐ obsolete	☐ for future use
short d_blgzt(short det)		
return value: det:	detector occupancy time in units of 100ms detector channel number	

Table 23: VS-PLUS function: read detector occupancy time

3.3.2.4 Time gap

t_zeitln(x)		Net time gap
The net time gap starts at the last falling slope		
VS-PLUS 2		d_ztlkn [translation: d_ntg]
☑ must be implemented	☐ obsolete	☐ for future use
short d_ztlkn(short det)		
return value: det:	net time gap in units of 100ms detector channel number	

Table 24: VS-PLUS function: read detector net time gap

t_zeitlb(x)		Gross time gap
The gross time gap starts at the last rising slope		
VS-PLUS 3		d_zeitlb [translation: d_gtg]
☑ must be implemented	☐ obsolete	☐ for future use
short d_ zeitlb (short det)		
return value: det:	gross time gap in units of 100ms detector channel number	

Table 25: VS-PLUS function: read gross net time gap

3.3.2.5 Fault

l_stoer(x)		Detector fault
This function tells VS-PLUS whether a detector fault has been detected by the controller		
VS-PLUS 3		d_stoer [translation: d_fault]
☑ must be implemented	☐ obsolete	☐ for future use
short d_stoer(short det)		
return value: det:	0: no fault; >0: fault detector channel number	

Table 26: VS-PLUS function: read detector fault

3.3.3 Signal groups and digital outputs

An overview over the signal group states is shown in chapter 4.6 "Signal groups".

3.3.3.1 Base controller times

t_min_rot(x) [translation: t_min_red]		Minimum red
The minimum red time (closed) defined in the controller.		
VS-PLUS 2		min_rot [translation: min_red]
☑ must be implemented	☐ obsolete	☐ for future use
short min_rot(short sg)		
return value:	minimum red time in units of 100ms	
sg:	signal group channel number	

Table 27: VS-PLUS function: read signal group minimum red time

T_vor(x) [translation: t_prep]		Preparation time
The preparation time expresses the transition time from closed to open. This function returns the value defined in the controller. The function returns 0 if no transition time is available. If the transition consists of several elements, the sum of all transition times hat to be returned.		
VS-PLUS 2		u_rot_gelb [translation: t_red_amber]
☑ must be implemented	☐ obsolete	☐ for future use
short u_rot_gelb(short sg)		
return value:	preparation time in units of 100ms	
sg:	signal group channel number	

Table 28: VS-PLUS function: read signal group preparation time (red-amber)

t_min_grun(x) [translation: t_min_green]		Minimum green
The minimum green time (open) defined in the controller.		
VS-PLUS 2		min_gruen [translation: min_green]
☑ must be implemented	☐ obsolete	☐ for future use
short min_gruen(short sg)		
return value: sg:	minimum green time in units of 100ms signal group channel number	

Table 29: VS-PLUS function: read signal group minimum green time

T_gelb(x) [translation: t_amber]		Amber time
The amber time expresses the transition time from open to closed. This function returns the value defined in the controller. The function returns 0 if no transition time is available. If the transition consist of several elements, the sum of all transition times hat to be returned		
VS-PLUS 2		u_gelb [translation: t_amber]
☑ must be implemented	☐ obsolete	☐ for future use
short u_gelb(short sg)		
return value: sg:	amber time in units of 100ms signal group channel number	

Table 30: VS-PLUS function: read signal group amber time

3.3.3.2 Signal group switching

EIN_Signal(x)		Switch signal group to open
Command for switching a signal to "open". The controller starts the transition from closed to open. The controller has to care about the correct transition sequence.		
VS-PLUS 2		SG_ein [translation: SG_open]
☑ must be implemented	☐ obsolete	☐ for future use
void SG_ein(short sg)		
return value: sg:	none signal group channel number	

Table 31: VS-PLUS function: command signal group to open

AUS_Signal(x)		Switch signal group to closed
Command for switching a signal to "closed". The controller starts the transition from closed to open. The controller has to care about the correct transition sequence.		
VS-PLUS 2		SG_aus(x) [translation: SG_close]
☑ must be implemented	☐ obsolete	☐ for future use
void SG_aus(short sg)		
return value: none sg: signal group channel number		

Table 32: VS-PLUS function: command signal group to close

3.3.3.3 Digital output switching

EIN_SState(x)		Switch digital output on
Command for switching a non-supervised output to "on". The output channel numbers can have a separate range outside the signal group range.		
VS-PLUS 2		Relais_ein [translation: relay_on]
☑ must be implemented	☐ obsolete	☐ for future use
void Relais_ein(short sg)		
return value: none sg: output channel number		

Table 33: VS-PLUS function: command digital output to on

AUS_SState(x)		Switch digital output off
Command for switching a non-supervised output to "off". The output channel numbers can have a separate range outside the signal group range.		
VS-PLUS 2		Relais_aus [translation: relay_off]
☑ must be implemented	☐ obsolete	☐ for future use
void Relais_aus(short sg)		
return value: none sg: output channel number		

Table 34: VS-PLUS function: command digital output to off

EIN_SBlink(x)		Switch blinker on
Command for switching a non-supervised blinker output (blinking digital output) to "on". The output channel numbers can have a separate range outside the signal group range.		
VS-PLUS 2		Blinker_ein [translation: blinker_on]
☑ must be implemented	☐ obsolete	☐ for future use
void Blinker_ein(short sg)		
return value: none sg: output channel number		

Table 35: VS-PLUS function: command blinker to on

AUS_SBlink(x)		Switch blinker off
Command for switching a non-supervised blinker output (blinking digital output) to "off". The output channel numbers can have a separate range outside the signal group range.		
VS-PLUS 2		Blinker_aus [translation: blinker_off]
☑ must be implemented	☐ obsolete	☐ for future use
void Blinker_aus(short sg)		
return value: none sg: output channel number		

Table 36: VS-PLUS function: command blinker to off

3.3.3.4 Extended signal group control

Signal_Dunkel(x)		Switch signal group to dark
		0
☐ must be implemented	☐ obsolete	☑ for future use

Table 37: VS-PLUS function: switch signal group to dark (for future use)

Signal_Blinken(x)	Switch signal group to blinking	
	0	
☐ must be implemented	☐ obsolete	☑ for future use

Table 38: VS-PLUS function: switch signal group to blinking (for future use)

Signal_FarbeRot(x)	Switch signal group to red	
	0	
☐ must be implemented	☐ obsolete	☑ for future use

Table 39: VS-PLUS function: switch signal group to red (for future use)

Signal_FarbeGruen(x)	Switch signal group to green	
	0	
☐ must be implemented	☐ obsolete	☑ for future use

Table 40: VS-PLUS function: switch signal group to green (for future use)

3.3.3.5 Current signal group states

HW_rot(x)	Signal group shows red	
This function enables VS-PLUS to check if a signal group shows red. The return value is 1 if this is the case. All other cases result in a return value of 0.		
VS-PLUS 2	s_rot [translation: s_red]	
☑ must be implemented	☐ obsolete	☐ for future use
short s_rot(short sg)		
return value: 1 = signal group shows red; 0 = not red sg: signal group channel number		

Table 41: VS-PLUS function: checks if signal group shows red

HW_min_rot(x)		Signal group is in minimum red
This function enables VS-PLUS to check if a signal group shows red and the elapsed red time is still within the minimum red time.		
VS-PLUS 2		s_min_rot [translation: s_min_red]
☑ must be implemented	☐ obsolete	☐ for future use
short s_min_rot(short sg)		
return value:	1 = signal group red and min red not reached; 0 = not red or min red elapsed	
sg:	signal group channel number	

Table 42: VS-PLUS function: checks if signal group is in minimum red

HW_gelb(x)		Signal group shows amber
This function enables VS-PLUS to check if a signal group shows amber or in general if the signal group is in the "open-closed" transition.		
VS-PLUS 2		s_gelb [translation: s_amber]
☑ must be implemented	☐ obsolete	☐ for future use
short s_gelb(short sg)		
return value:	1 = signal group is in transition "open-closed"; 0 = not in transition	
sg:	signal group channel number	

Table 43: VS-PLUS function: checks if signal group shows amber

HW_grun(x)		Signal group shows green
This function enables VS-PLUS to check if a signal group shows green. The return value is 1 if this is the case. All other cases result in a return value of 0.		
VS-PLUS 2		s_grun [translation: s_green]
☑ must be implemented	☐ obsolete	☐ for future use
short s_grun(short sg)		
return value:	1 = signal group shows green; 0 = not green	
sg:	signal group channel number	

Table 44: VS-PLUS function: checks if signal group shows green

HW_min_grun(x)		Signal group is in minimum green
This function enables VS-PLUS to check if a signal group shows green and the elapsed green time is still within the minimum green time.		
VS-PLUS 2		s_min_grun [translation: s_min_green]
☑ must be implemented	☐ obsolete	☐ for future use
short s_min_grun(short sg)		
return value:	1 = signal group green and min green not reached; 0 = not green or min green elapsed	
sg:	signal group channel number	

Table 45: VS-PLUS function: checks if signal group is in minimum green

HW_vor(x)		Signal group shows red-amber
This function enables VS-PLUS to check if a signal group shows red-amber or in general if the signal group is in the "closed-open" transition.		
VS-PLUS 2		s_vor [translation: s_prep]
☑ must be implemented	☐ obsolete	☐ for future use
short s_vor(short sg)		
return value:	1 = signal group is in transition "closed-open "; 0 = not in transition	
sg:	signal group channel number	

Table 46: VS-PLUS function: checks if signal group is in minimum green

3.3.3.6 Current status of digital outputs

HW_sr_aus(x)		Digital output is off
This function enables VS-PLUS to check if a digital output is switched off.		
VS-PLUS 2		s_sr_aus [translation: s_out_off]
☑ must be implemented	☐ obsolete	☐ for future use
short s_sr_aus(short re)		
return value:	1 = output is off; 0 = output is not off	
re:	output channel number	

Table 47: VS-PLUS function: checks if digital output is off

HW_sb_aus(x)	Digital blinker is off
This function enables VS-PLUS to check if a digital blinker output is switched off.	

VS-PLUS 2		s_sb_aus [translation: s_blink_off]
☑ must be implemented	☐ obsolete	☐ for future use
short s_sb_aus(short bli)		
return value: 1 = output is off; 0 = output is not off bli: output channel number		

Table 48: VS-PLUS function: checks if digital blinker is off

HW_sr_ein(x)		Digital output is on
This function enables VS-PLUS to check if a digital output is switched on.		
VS-PLUS 2		s_sr_ein [translation: s_out_on]
☑ must be implemented	☐ obsolete	☐ for future use
short s_sr_ein(short re)		
return value: 1 = output is on; 0 = output is not on re: output channel number		

Table 49: VS-PLUS function: checks if digital output is on

HW_sb_ein(x)		Digital blinker is on
This function enables VS-PLUS to check if a digital blinker output is switched on.		
VS-PLUS 2		s_sb_ein [translation: s_blink_on]
☑ must be implemented	☐ obsolete	☐ for future use
short s_sb_ein(short bli)		
return value: 1 = output is on; 0 = output is not on bli: output channel number		

Table 50: VS-PLUS function: checks if digital blinker is on

3.3.3.7 Switching command queries

HW_bef_rot(x)(		Is there a red command for the signal group?
☐ must be implemented	☑ obsolete	☐ for future use

Table 51: VS-PLUS function: checks if signal group can be switched to red (obsolete)

HW_bef_grun(x)		Is there a green command for the signal group?
☐ must be implemented	☑ obsolete	☐ for future use

Table 52: VS-PLUS function: checks if signal group can be switched to green (obsolete)

3.3.3.8 Current signal group times

HW_t_rot(x)		Current signal group red time
Tells how long the signal group is already red.		
VS-PLUS 2		s_t_rot [translation: s_t_red]
☑ must be implemented	☐ obsolete	☐ for future use
unsigned short s_t_rot(short sg)		
return value:	current red time in units of 100ms	
sg:	signal group channel number	

Table 53: VS-PLUS function: asks for current signal group red time

HW_t_gelb(x)		Current signal group amber time
Tells how long the signal group is already amber (transition state from open to closed).		
VS-PLUS 2		s_t_gelb [translation: s_t_amber]
☑ must be implemented	☐ obsolete	☐ for future use
unsigned short s_t_gelb(short sg)		
return value:	current amber time in units of 100ms	
sg:	signal group channel number	

Table 54: VS-PLUS function: asks for current signal group amber time

HW_t_min_rot(x)		Current signal group minimum red time
Tells how long the signal group is already in minimum red state. After the elapsing of the minimum red time the return value is 0.		
VS-PLUS 2		s_t_min_rot [translation: s_t_min_red]
☑ must be implemented	☐ obsolete	☐ for future use
unsigned short s_t_min_rot(short sg)		
return value:	actual minimum red time in units of 100ms	
sg:	signal group channel number	

Table 55: VS-PLUS function: asks for current signal group minimum red time

HW_t_grun(x)		Current signal group green time
Tells how long the signal group is already green.		
VS-PLUS 2		s_t_grun [translation: s_t_green]
☑ must be implemented	☐ obsolete	☐ for future use
unsigned short s_t_grun(short sg)		
return value:	current green time in units of 100ms	
sg:	signal group channel number	

Table 56: VS-PLUS function: asks for current signal group green time

HW_t_vor(x)		Current signal group red-amber time
Tells how long the signal group is already red-amber (transition from closed to open).		
VS-PLUS 2		s_t_vor [translation: s_t_prep]
☑ must be implemented	☐ obsolete	☐ for future use
unsigned short s_t_vor(short sg)		
return value:	current red-amber time in units of 100ms	
sg:	signal group channel number	

Table 57: VS-PLUS function: asks for current signal group red-amber time

HW_t_min_grun(x)		Current signal group minimum green time
Tells how long the signal group is already in minimum green state. After the elapsing of the minimum green time the return value is 0.		
VS-PLUS 2		s_t_min_grun [translation: s_t_min_green]
☑ must be implemented	☐ obsolete	☐ for future use
unsigned short s_t_min_grun(short sg)		
return value: actual minimum green time in units of 100ms sg: signal group channel number		

Table 58: VS-PLUS function: asks for current signal group minimum green time

3.3.3.9 Special status

HW_blink(x)		Signal group is in fault blinking mode
Indicates if the signal group is in fault mode		
VS-PLUS 2		s_stoeblink [translation: s_faultBlink]
☑ must be implemented	☐ obsolete	☐ for future use
short s_stoeblink (short sg)		
return value: 1 = signal group in fault mode; 0 = not in fault mode sg: signal group channel number		

Table 59: VS-PLUS function: checks if signal group is in fault blinking mode

HW_t_blink(x)		Fault duration
		0
☐ must be implemented	☐ obsolete	☑ for future use

Table 60: VS-PLUS function: asks for signal group fault duration

HW_Dunkel(x)	Signal group is dark	
	0	
☐ must be implemented	☐ obsolete	☑ for future use

Table 61: VS-PLUS function: checks if signal group is dark

HW_Blinkend(x)	Signal group is blinking	
	0	
☐ must be implemented	☐ obsolete	☑ for future use

Table 62: VS-PLUS function: checks if signal group is blinking

HW_VspFreigegeben(x)	Enabled for VS-PLUS	
Indicates is a signal group is enabled for VS-PLUS. If this is the case, the signal group is under VS-PLUS control. See chapter 4.6.3 "Enabled signal groups"		
VS-PLUS 6	s_HW_VspFreigegeben [translation: s_HW_VspEnabled]	
☑ must be implemented	☐ obsolete	☐ for future use
unsigned short s_HW_VspFreigegeben(short sg)		
return value: 1 = signal group is enabled; 0 = not enabled sg: signal group channel number		

Table 63: VS-PLUS function: checks if signal group is enabled for VS-PLUS

3.3.3.10 Controller intergreen times

l_zzV(x,y)		Controller intergreen
A controller intergreen value		
VS-PLUS 3		s_zwi_zeit [translation: s_intergreen]
☑ must be implemented	☐ obsolete	☐ for future use
short s_zwi_zeit (short sgr, short sge)		
return value:	Controller intergreen time in units of 100ms	
sgr:	Clearing signal group channel number	
sge:	Entering signal group channel number	

Table 64: VS-PLUS function: reads controller intergreen

3.3.3.11 Messages

MELDUNG(degree,nr,par1,par2,par3,par4)		VS-PLUS messages
Messages sent by VS-PLUS		
VS-PLUS 2		Meldung [translation: message]
☑ must be implemented	☐ obsolete	☐ for future use
void Meldung(short degree, short nr, short par1, short par2, short par3, short par4)		
return value:	none	
degree:	message criticality	
nr:	message number	
par1 – par4:	message parameter	

Table 65: VS-PLUS function: writes a predefined numerical message to the controller

MELDUNGnet(degree,nr,Anr,par1,par2,par3,par4,par5)		VS-PLUS messages from network control
Messages sent from the network control part within VS-PLUS		
VS-PLUS 6		MeldungNET [translation: messageNET]
☑ must be implemented	☐ obsolete	☐ for future use
void MeldungNET(unsigned char degree, unsigned char nr, unsigned short Anr, unsigned char par1, unsigned char par2, unsigned char par3, unsigned char par4, unsigned char par5)		
return value:	none	
degree:	message criticality	
nr:	message number	
Anr:	mission number	
par1 – par5:	message parameter	

Table 66: VS-PLUS function: writes a predefined numerical network control message to the controller

3.3.4 System functions

SteuerungNichtAktiv()		Control not active
The function returns 0 as long as a fixed time signal plan is being processed. When off or during switch on or switch off, this function returns 1		
VS-PLUS 6		s_SteuerungNichtAktiv [translation: s_controlNotActive]
☑ must be implemented	☐ obsolete	☐ for future use
unsigned short s_SteuerungNichtAktiv(void)		
return value:	1 = Off or switching on or off; 0 = signal program active	

Table 67: VS-PLUS function: checks if VS-PLUS control is not active

GibTelegramm(x)		Read serial telegrams
VS-PLUS reads the serial PT telegrams that have been received by the controller.		
VS-PLUS 6		TelegrammVomGeraet [translation: telegram_fm_controller]
☑ must be implemented	☐ obsolete	☐ for future use
short TelegrammVomGeraet(void* oev_tele_poi)		
return value:	1 = telegram available; 0 = none available	

oev_tele_poi:
struct R09serialTelegram
	pointe on a telegram
	{
	int MP (call point);
	int Linie (line);
	int Kurs (course);
	int Route (route);
	int Prioritaet (priority);
	int Laenge (vehicle length);
	int RichtungVonHand (direction by hand);
	int FahrplanAbweichnung (difference to schedule);
	};	

Table 68: VS-PLUS function: reads PT telegram

SetZwangsloschung(x,t,z,p)		Force check-out
		0
☐ must be implemented	☑ obsolete	☐ for future use

Table 69: VS-PLUS function: force PT check-out (obsolete)

m_Wunsch_VSPLUS(x,y)		Switch off VS-PLUS

Through this function VS-PLUS can ask the controller to be switched in a different mode.		
VS-PLUS 6.2	Wunsch_VSPLUS [translation: reqest_VSPLUS]	
☑ must be implemented	☐ obsolete	☐ for future use
int Wunsch_VSPLUS(int Wunsch, int Teiknoten)		
return value:	1 = request will be executed; 0 = function not supported	
Request:	0 = check if this function is supported by the controller 1 = switch into fix time mode 2 = switch controller off 3 = restart VS-PLUS (reset) 4 = switch program according to program timer 5 = predefined program (emergency program)	
Partial node:	0 = whole intersection 1 – 99 = partial node number 1 – 32 = program number for emergency program	

Table 70: VS-PLUS function: switch off VS-PLUS

3.3.5 OCIT functions

3.3.5.1 Public transport (PT) and Individual traffic (IT) on/off

I_OePNV_Ein_Aus()	OCIT PT on/off status	
VS-PLUS asks the controller if the OCIT Outstations status "OePNV" (PT) is on or off.		
VS-PLUS 6.2	OePNV_Ein_Aus [translation: PT_on_off]	
☑ must be implemented	☐ obsolete	☐ for future use
short OePNV_Ein_Aus(void)		
return value:	0 = not set 1 = PT off 2 = PT on	

Table 71: VS-PLUS function: reads OCIT PT on/off status

I_IV_Ein_Aus()		OCIT IT on/off
VS-PLUS asks the controller if the OCIT Outstations status "IV" (IT) is on or off.		
VS-PLUS 6.2		IV_Ein_Aus [translation: IT_on_off]
☑ must be implemented	☐ obsolete	☐ for future use
short IV_Ein_Aus(void)		
return value:	0 = not set 1 = IT off 2 = IT on	

Table 72: VS-PLUS function: reads OCIT IT on/off status

I_ZSondereingriffvn(S, E, vn)		OCIT special program modification
VS-PLUS reads the actual command "ZSondereingriff" (special program modification) and the corresponding job ID.		
VS-PLUS 6.2.6		ZSondereingriffvn [translation: SpecProgMod]
☑ must be implemented	☐ obsolete	☐ for future use
void ZSondereingriffvn(unsigned char* Sondereingriff, unsigned long* EndZeitpunkt, unsigned long* VorgangsNummer)		
return value:	none	
Sondereingriff:	number of special program modification (1 … 254)	
EndZeitpunkt:	end time (UTC time)	
VorgangsNummer:	job number	

Table 73: VS-PLUS function: reads OCIT special program modification status

I_UTCZeitstempel()		UTC timestamp
VS-PLUS needs the actual time from the controller in order to be able to check if the end of a program modification has been reached.		
VS-PLUS 6.2.6		UTCZeitstempel [translation: UTC_timestamp]
☑ must be implemented	☐ obsolete	☐ for future use
unsigned long UTCZeitstempel(void)		
return value:	actual time	

Table 74: VS-PLUS function: reads the UTC timestamp

3.3.6 Test functions

3.3.6.1 Detectors

l_Det_Aktiv(x)		Detector exists
Function to check if a detector with a certain channel number is defined in the controller		
VS-PLUS 6.2		Det_Aktiv [translation: det_exists]
☑ must be implemented	☐ obsolete	☐ for future use
short Det_Aktiv(short KanalNummer)		
return value: 1 = detector is defined in the controller; 0 = not defined KanalNummer: detector channel number		

Table 75: VS-PLUS function: checks for detector existence

3.3.6.2 Signal groups

l_Sg_Aktiv(x)		Signal group exists
Function to check if a signal group with a certain channel number is defined in the controller		
VS-PLUS 6.2		Sg_Aktiv [translation: sg_exists]
☑ must be implemented	☐ obsolete	☐ for future use
short Sg_Aktiv(short KanalNummer)		
return value: 1 = signal group is defined in the controller; 0 = not defined KanalNummer: signal group channel number		

Table 76: VS-PLUS function: checks for signal group existence

3.3.6.3 Date and time

l_AktuelleZeit(x,y,z)		Read current time
Diese Funktion liefert die aktuelle Uhrzeit		
VS-PLUS 6.2		AktuelleZeit [translation: CurrentTime]
☑ must be implemented	☐ obsolete	☐ for future use
int AktuelleZeit(int* Stunde, int* Minute, int* Sekunde)		
return value:	1 = time is set; 0 = not set	
Stunde:	current hour (0 … 23)	
Minute:	current minute (0 … 59)	
Sekunde:	current second (0 … 59)	

Table 77: VS-PLUS function: reads current time

l_AktuellesDatum(w,x,y,z)		Read current date
This function returns the actual date.		
VS-PLUS 6.2		AktuellesDatum [translation: CurrentDate]
☑ must be implemented	☐ obsolete	☐ for future use
int AktuellesDatum(int* Jahr, int* Monat, int* Tag, int* Wochentag)		
return value:	1 = date is set; 0 = not set	
Jahr:	current year (1970 .. 2999)	
Monat:	current month (1 .. 12)	
Tag:	current day (1 .. 31)	
Wochentag:	current weekday (1 = Monday; 2 = Tuesday; 3 = Wednesday; 4 = Thursday; 5 = Friday; 6 = Saturday; 7 = Sunday)	

Table 78: VS-PLUS function: reads current date

3.3.6.4 Nodes (intersections)

l_Get_OCITOutstationId(x,y,z)		OCIT-O identification
Returns the OCIT Outstations node identification		
VS-PLUS 6.2		Get_OCITOutstationId
☑ must be implemented	☐ obsolete	☐ for future use
int Get_OCITOutstationId(int* ZNr, int* FNr, int* Relknoten)		
return value:	1 = id is set; 0 = not set	
ZNr:	area number	
FNr:	controller number	
Relknoten:	relative node number	

Table 79: VS-PLUS function: reads OCIT-O node identification

l_ProgrammWahlZentrale()		Program number source
Returns the program number source (i.e. who has sent current valid program switch command)		
VS-PLUS 6.2		ProgrammWahlZentrale [translation: ProgramChoiceCentral]
☑ must be implemented	☐ obsolete	☐ for future use
int ProgrammWahlZentrale(void)		
return value:	1 = program number comes from central; 0 = program number comes locally from controller	

Table 80: VS-PLUS function: reads the program number source

3.3.7 Parameter supply

Chapter 5.1 "VS-PLUS parameter" explains in detail how to supply the parameters to the controller.

3.3.7.1 Storage Management

f_Alloziere_VSP_Speicher(x,y)		Memory allocation
VS-PLUS requests memory from the controller in order to save the VS-PLUS parameters.		
VS-PLUS 6.2		Allozieren_VSP_Speicher [translation: allocate_VSP_memory]
☑ must be implemented	☐ obsolete	☐ for future use
void* Allozieren_VSP_Speicher(int _sizeof, int id)		
return value:	pointer to memory area	
Sizeof:	size of memory to be allocated	
Id:	memory area identification (1 – 3)	

Table 81: VS-PLUS function: memory allocation

l_Freigeben_VSP_Speicher(x)		Free allocated memory area
VS-PLUS frees memory that has been requested from the controller.		
VS-PLUS 6.2		Freigeben_VSP_Speicher [translation: free_VSP_memory]
☑ must be implemented	☐ obsolete	☐ for future use
void Freigeben_VSP_Speicher(int id)		
return value:	none	
id:	memory area identification	

Table 82: VS-PLUS function: memory free

l_Gib_VSP_Zeiger(x)		Pointer to memory area
This function returns the pointer to the memory area		
VS-PLUS 6.2		Gib_VSP_Zeiger [translation: get_VSP_pointer]
☑ must be implemented	☐ obsolete	☐ for future use
void* Gib_VSP_Zeiger(int id)		
return value:	pointer to memory area	
Id:	memory area identification (1 – 3)	

Table 83: VS-PLUS function: memory pointer

3.3.7.2 Read supply file

l_Oeffnen_VSP_Parameter()	Open supply file (VCB)

This function opens the VCB file that has been sent to the controller. This function corresponds to a "fopen" in C.		
VS-PLUS 6.2		Oeffnen_VSP_Parameter [translation: open_VSP_parameters]
☑ must be implemented	☐ obsolete	☐ for future use
int Oeffnen_VSP_Parameter(void)		
return value:	1 = supply file was opened successfully, 0 = not opened	

Table 84: VS-PLUS function: open VCB supply file

l_Lesen_VSP_Parameter(x,y)	Read supply file
Data is read sequentially from the formerly opened VCB file. This function corresponds to a "fread" in C.	
VS-PLUS 6.2	Read_VSP_Parameter [translation: read_VSP_parameters]
☑ must be implemented / ☐ obsolete / ☐ for future use	
int Read_VSP_Parameter(char* data, int _sizeof)	
return value:	size of data read; -1 = error
data:	data read
sizeof:	size of data to be read

Table 85: VS-PLUS function: read VCB supply file

l_Schliessen_VSP_Parameter	Close supply file
This function closes the supply file. This function corresponds to a "fclose" in C.	
VS-PLUS 6.2	Schliessen_VSP_Parameter [translation: close_VSP_parameters]
☑ must be implemented / ☐ obsolete / ☐ for future use	
void Schliessen_VSP_Parameter(void)	
return value:	none

Table 86: VS-PLUS function: close VCB supply file

3.3.7.3 Save supply file

m_Oeffnen_Sichern_Parameter()		Open backup supply file (VCB)
This function opens the backup supply file that is stored on the controller. This function corresponds to a "fopen" in C.		
VS-PLUS 6.2		Oeffnen_Sichern_Parameter [translation: open_save_parameters]
☑ must be implemented	☐ obsolete	☐ for future use
int Oeffnen_Sichern_Parameter(void)		
return value:	1 = supply file was opened successfully, 0 = not opened	

Table 87: VS-PLUS function: open backup VCB supply file

m_Schreiben_Sichern_Parameter(x,y)		Write backup supply file
This function writes inthe backup supply file. This function corresponds to a "fwrite" in C.		
VS-PLUS 6.2		Schreiben_Sichern_Parameter [translation: write_save_parameters]
☑ must be implemented	☐ obsolete	☐ for future use
void Schreiben_Sichern_Parameter(char* data, int _sizeof)		
return value: data: sizeof:	none the value to be written size of the value to be written	

Table 88: VS-PLUS function: write backup VCB supply file

m_Schliessen_Sichern_Parameter()		Close backup supply file
This function closes the backup supply file. This function corresponds to a "fclose" in C.		
VS-PLUS 6.2		Schliessen_Sichern_Parameter [translation: close_save_parameters]
☑ must be implemented	☐ obsolete	☐ for future use
void Schliessen_Sichern_Parameter(void)		
return value:	none	

Table 89: VS-PLUS function: close backup VCB supply file

3.3.8 Read command file

Chapter 5.2 "VS-PLUS" explains in detail how reading of a command file works.

3.3.8.1 Check for new command file

l_Neue_Befehle()		Check for new file
VS-PLUS checks whether a new command file is available from the controller.		
VS-PLUS 6.2		Neue_Befehle [translation: new_commands]
☑ must be implemented	☐ obsolete	☐ for future use
int Neue_Befehle(void)		
return value:	1 = new command file is present 0 = no file present	

Table 90: VS-PLUS function: check for new command file

3.3.8.2 Read command file

l_Oeffnen_VSP_Befehle()		Open command file (VCB)
This function opens the command file which has arrived on the controller. This function corresponds to a "fopen" in C.		
VS-PLUS 6.2		Oeffnen_VSP_Befehle [translation: open_VSP_commands]
☑ must be implemented	☐ obsolete	☐ for future use
int Oeffnen_VSP_Befehle(void)		
return value:	1 = command file was opened successfully 0 = not opened	

Table 91: VS-PLUS function: open command file

l_Lesen_VSP_Befehle(x,y)		Read command file
Data is read sequentially from the formerly opened command file. This function corresponds to a "fread" in C.		
VS-PLUS 6.2		Lesen_VSP_Befehle [translation: read_VSP_commands]
☑ must be implemented	☐ obsolete	☐ for future use
int Lesen_VSP_Befehle(char* data, int _sizeof)		
return value:	size of data read; -1 = error	
data:	data read	
sizeof:	size of data to be read	

Table 92: VS-PLUS function: read command file

l_Schliessen_VSP_Befehle		Close command file
The formerly opened command file is closed. . This function corresponds to a "fclose" in C.		
VS-PLUS 6.2		Schliessen_VSP_Befehle [translation: close_VSP_commands]
☑ must be implemented	☐ obsolete	☐ for future use
void Schliessen_VSP_Befehle(void)		
return value:	none	

Table 93: VS-PLUS function: close command file

3.4 VS-PLUS functions

These VS-PLUS functions are called by the controller.

3.4.1 System functions

m_Prog_Schaltung_erlaubt()		Program change possible
This function tells the controller that a pending program switch has to be delayed. The controller has to delay the program switching until this function allows it. Only traffic-actuated VS-PLUS programs have to be delayed.		
VS-PLUS 6.2		Prog_Schaltung_erlaubt [translation: Prog_switch_allowed]
☑ must be implemented	☐ obsolete	☐ for future use
int Prog_Schaltung_erlaubt(void)		
return value:	1 = Switching to a new program allowed; 0 = not allowed	

Table 94: VS-PLUS function: controller checks if program change is possible

3.4.2 OCIT functions

3.4.2.1 PT and IT on/off

l_VSP_X_Ein_Aus_Ist(0)		Current status OCIT PT on/off
This function returns the value to the controller how PT on / off is set.		
VS-PLUS 6.2		OePNV_Ein_Aus_Ist [translation: PT_on_off_status]
☑ must be implemented	☐ obsolete	☐ for future use
short OePNV_Ein_Aus_Ist(void)		
return value:	1 = PT off 2 = PT on	

Table 95: VS-PLUS function: current OCIT status PT on/off

I_VSP_X_Ein_Aus_Ist(1)		Current status OCIT IT on/off
This function returns the value to the controller how IT on / off is set.		
VS-PLUS 6.2		IV_Ein_Aus_Ist [translation: IT_on_off_status]
☑ must be implemented	☐ obsolete	☐ for future use
short IV_Ein_Aus_Ist(void)		
return value:	1 = IT off 2 = IT on	

Table 96: VS-PLUS function: current OCIT status IT on/off

m_ZSondereingriffVSPvn(vn)		Active program modification with job id
This function returns the value to the controller of the active program modification and the job id.		
VS-PLUS 6.2.6		ZSondereingriffVSP [translation: ProgModifVSP]
☑ must be implemented	☐ obsolete	☐ for future use
unsigned char m_ZSondereingriffVSPvn(unsigend long* JobID)		
return value: JobID:	number of program modification corresponding job id	

Table 97: VS-PLUS function: active program modification with job id

3.4.3 Test functions

3.4.3.1 Programs

l_Prog_VSP(x)		VS-PLUS Program
This function enables the controller to check if a program number belongs to a VS-PLUS program or not		
VS-PLUS 6.2		Prog_VSP
☑ must be implemented	☐ obsolete	☐ for future use
short Prog_VSP(short prg)		
return value:	0 = not defined 1 = fixed time with VS-PLUS parameters 2 = fixed time without VS-PLUS parameters 5 = VS-PLUS with parameters 6 = VS-PLUS without parameters	
prg:	program number	

Table 98: VS-PLUS function: controller checks if a program number belongs to VS-PLUS

3.4.4 VS-PLUS supply

f_Initial_VSP_Parameter()		Initialize VS-PLUS parameters
With this function the controller starts the initialization of the VS-PLUS parameter area.		
VS-PLUS 6.2		Initial_VSP_Parameter
☑ must be implemented	☐ obsolete	☐ for future use
int Initial_VSP_Parameter(void)		
return value:	0 = no error -1 = error	

Table 99: VS-PLUS function: initialize VS-PLUS parameters

f_Pruefen_VSP_Parameter()		Check supply file
With this function the controller starts the parameter file checking by VS-PLUS. The meaning of the return value in case of an error is explained in chapter 4.2.3.2 "Checking of VS-PLUS parameters".		
VS-PLUS 6.2		Pruefen_VSP_Parameter [translation: check_VSP_parameter]
☑ must be implemented	☐ obsolete	☐ for future use
int Pruefen_VSP_Parameter(void)		
return value:	< 0 = error 0 = no error	

Table 100: VS-PLUS function: check VS-PLUS parameter supply file

f_Lesen_VSP_Parameter()		Read supply file
With this function the controller starts the parameter file reading by VS-PLUS. The meaning of the return value in case of an error is explained in chapter 4.2.3.3 "Reading of VS-PLUS parameters".		
VS-PLUS 6.2		Lesen_VSP_Parameter [translation: read_VSP_parameter]
☑ must be implemented	☐ obsolete	☐ for future use
int Lesen_VSP_Parameter(void)		
return value:	< 0 = error 0 = no error	

Table 101: VS-PLUS function: read VS-PLUS parameter supply file

f_Ende_VSP_Parameter()		Free parameter area
The controller requests VS-PLUS to give free the parameter area.		
VS-PLUS 6.2		Ende_VSP_Parameter [translation: end_VSP_parameter]
☑ must be implemented	☐ obsolete	☐ for future use
void Ende_VSP_Parameter(void)		
return value:	none	

Table 102: VS-PLUS function: free VS-PLUS parameter memory area

l_V_Build_Nr()		VS-PLUS build number
☐ must be implemented	☐ obsolete	☑ for future use

Table 103: VS-PLUS function: read VS-PLUS build number (for future use)

3.4.5 Reading from the command file

l_VABefehlPfad()		Provide OCIT path
Returns the path name for transmitting an OCIT command (default "57.520")		
VS-PLUS 6.2		VABefehlPfad [translation: OCIT_command_path]
☑ must be implemented	☐ obsolete	☐ for future use
char* VABefehlPfad(void)		
return value:	OCIT path name for command file	

Table 104: VS-PLUS function: provide OCIT path for command file

3.4.6 VS-PLUS version

f_versions_txt(x,y)		Read VS-PLUS version
With this function the controller can find out what VS-PLUS version is in use.		
VS-PLUS 6.2		versions_txt [translation: version_txt]
☑ must be implemented	☐ obsolete	☐ for future use
short versions_txt(char* text, int size)		
return value: text: size:	1 = version information available 0 = not available VS-PLUS version text maximum number of characters for version text (min. 10).	

Table 105: VS-PLUS function: read VS-PLUS version

3.4.7 Process Data

f_VSP_ProzessDaten(x,y)		Read process data
With this function the controller can read the process data from VS-PLUS according to OCIT-I		
VS-PLUS 6		VSP_ProzessDaten [translation: VSP_processData]
☑ must be implemented	☐ obsolete	☐ for future use
unsigned short VSP_ProzessDaten(void *px, void *py)		
The detailed functionality is described in chapter 5.3 "Pro".		

Table 106: VS-PLUS function: read VS-PLUS process data

3.4.8 PT module

l_OEVSpeicherLesenEin()		
☐ must be implemented	☑ obsolete	☐ for future use

f_OEVSpeicherAusgabe(x,y)		
☐ must be implemented	☑ obsolete	☐ for future use

4 VS-PLUS CONTROL

4.1 Global settings

4.1.1 Times

All times exchanged between VS-PLUS and the controller are in units of 100ms.

4.1.2 Compiler settings

If there are no special comments in the following descriptions, the settings are defines (#define) and have to be written into the make file.

4.1.3 GERAET_TEILKNOTEN_MAX

The global constant "GERAET_TEILKNOTEN_MAX" defines how many partial intersections are allowed for the controller.

If a controller does not support partial intersections, this value has to be 1.

If a controller supports more than one partial intersection, this value has to be

Number of partial intersections + 1

OCIT defines a maximum of 4 partial intersections.

This is defined in the include file named "ifxxxbas.inc".

4.1.4 SYSTEMLANGUAGE

The global constant "SYSTEMLANGUAGE" defines the output language of the PT memory module.

Actually the value "49" is for German and "no value for English.

This is defined in the include file named "ifxxxbas.inc"

4.1.4.1 _PROZESSOR_INTEL_

The global constant "_PROZESSOR_INTEL_" must be set when VS-PLUS runs on an operating system using "Little Endian" coding scheme for number.

4.1.4.2 _HELPER_MACROS_

Must be set if the macros

- LOBYTE
- HIBYTE
- LOWORD
- HIWORD

are not defined.

4.1.4.3 _VSPLUS_SIM_

May only be set if the VS-PLUS code is used in order to run a simulation. Option is to be used by Verkehrs-Systeme AG only.

4.1.4.4 *EMULATOR*

May only be set if the VS-PLUS code is used in order to run a simulation. Option is to be used by Verkehrs-Systeme AG only.

4.1.4.5 _SIEMENS_C800_

Only used on a Siemens C800.

4.1.4.6 _SIEMENS_C900

Only used on a Siemens C900.

4.1.4.7 _IOOEVDEK

Must be set if the PT memory / module declaration is contained within the VS-PLUS; only used by Siemens.

4.1.4.8 SIEMENS_VSP

Must be set if the "spezial()" calls are not done within the VS-PLUS code but externally; only used by Siemens.

4.1.4.9 _ADVANCED_MEMORY_

Must be set if two data areas for reading and holding of VS-PLUS parameters are used.

4.1.5 Number of timers

The maximum number of timers is defined in the file "ifxxxbas.inc".

The following table is based on VS-PLUS version 6.2.6. The number of elements can change for a new version and has to be adjusted accordingly. The maximum values of the constants can be found in the file "ifxxxmax.inc".

Name	Formula	Value	Number
VSP_T0	0	0	0
VSP_T1	VSP_T0+DETMAX	340	340
VSP_T2	VSP_T1+DETMAX	680	340
VSP_T3	VSP_T2+DETMAX	1020	340
VSP_T4	VSP_T3+VSMAX	1084	64
VSP_T5	VSP_T4+VSMAX	1148	64
VSP_T6	VSP_T5+VSMAX	1212	64
VSP_T7	VSP_T6+VSMAX	1276	64
VSP_TE	VSP_T7+SPEZMAX	1292	16
Total			***1292***

Table 107: Number of timers

4.2 Calling VS-PLUS

4.2.1 Main function

The controller controls VS-PLUS by calling the main function VSPLUS(….).

VS-PLUS is configured in such a way that it is called once per second. The called function terminates as soon as all necessary internal tasks have been executed. The execution duration cannot be predicted as the duration depends on the CPU power, the node size and the actual traffic demand.

The controller needs to monitor the VSPLUS main function for proper execution. If the function does not terminate within the intended time frame, the controller has to kill VS-PLUS and take back the commands over the controller. In such a case the controller switches VS-PLUS off and continues working in fixed time mode.

4.2.2 Call parameters

VS-PLUS is called with the following parameters:

VSPLUS(VSPSollTyp* VSPSoll, WBTyp* SchaltBild, WBReadyTyp* WunschBildBereit)

- VSPSoll: Desired operation mode of VS-PLUS
- SchaltBild: The off or change stage
- WunschBildBereit: Flag when the desired stage is reached

4.2.2.1 VSPSoll

The call parameters contain a pointer to a variable with the following structure:

VSPSOLLTYP VSPSOLL[GERAET_TEILKNOTEN_MAX]

On index 0 the array contains the parent control request (for controllers without partial intersections the control request for the entire node). Index positions 1 to n contain the partial intersection shutdown requests. They are needed in order to make a shut down or switch on partial intersections during operation.

```
typedef enum {
              VSP_AUS             = 0,
              VSP_EIN             = 1,
              VSP_NEU             = 2,
VSP_WUNSCH_AUS_UM  = 3,
              VSP_NEU_INI         = 4,
              VSP_ND              = 99
} VSPSollTyp;
```

This enumeration type defines the possible operation modes of VS-PLUS. The definition can be found in the file "vspxxxste.inc".

VSP_AUS	The whole intersection is not under VS-PLUS control any more. Only detectors and call points are treated by VS-PLUS, no switching commands are generated.
VSP_EIN	VS-PLUS runs in normal mode.
VSP_NEU	The whole intersections is again under VS-PLUS control.
VSP_WUNSCH_AUS_UM	The whole intersection is switched off or is changing modes.
VSP_NEU_INI	The whole intersection is initialized.
VSP_ND	The parent control request must never reach this value. This value is needed internally in VS-PLUS.

4.2.2.2 SchaltBild

The call parameters contain a pointer to a variable with the following structure:

WBTYP SCHALTBILD[SGMAX]

For each signal group a value can be given defining its behavior during mode change or switch off. The following values are possible

0 = the signal group shall switch to red

1 = the signal group shall switch to green

The signal group number is calculated from Index + 1 (Index 0 -= signal group 1).

On the base of the individual signal group requests VS-PLUS calculates the traffic stream requests. This does not happen without consideration of the VS-PLUS conflicts. I.e. the user must make sure that there are no parameter errors.

```
typedef   unsigned char    WBTyp;
```

4.2.2.3 WunschBildBereit

The call parameters contain a pointer to a variable with the following structure:

WBREADYTYP WUNSCHBILDBEREIT[GERAET_TEILKNOTEN_MAX]

This is a pointer on an array by whom VS-PLUS tells the controller when the desired stage has been reached. The desired stage is set to 0 (false) by the controller and VS-PLUS sets it to 1 (true) as soon as the desired program switch stage is reached.

The array structure is the same as for VSPSoll. If only 1 partial intersection shall be switched off, the WBReady flag will only be set for that partial intersection.

4.2.3 Return value

By the return value VS-PLUS tells the controller if the operating mode has been executed successfully or if there has been an error.

4.2.3.1 No error

Values >= 0: successful execution of VS-PLUS. No error has occurred. The return value contains the mode value in which VS-PLUS has been executed.

Return value	Meaning
0	VSP_AUS
1	VSP_EIN
2	VSP_NEU
3	VSP_WUNSCH_AUS_UM
4	VSP_NEU_INI
99	VSP_ND

Table 108: VS-PLUS execution mode codes

4.2.3.2 Checking of VS-PLUS parameters

Values < 0 mean errors.

Error message: In addition to the value, VS-PLUS sends a detailed error report through the MELDUNG interface function.

Return value	Meaning	Error message
-2	Controller number is not correct	13, 2, controller number, controller number in parameters, 0
-3	Signal group unknown to the controller	13, 3, signal group number, 0, 0
-4	Unknown detector	13, 4, detector number, 0, 0
-5	File could not be opened	13, 5, 0, 0, 0
-6	Not a VCB file	13, 6, 0, 0, 0
-7	VS-PLUS version is not correct	13, 7, VS-PLUS version, VS-PLUS version in parameters, 0
-8	Unexpected end of file	13, 8, 0, 0, 0
-9	Structure error	13, 9, group, number, 0
-10	Data sets defined incorrectly	13, 10, 63, data sets, 0
-99	Memory not available	13, 99, 0, 0, 0

Table 109: VS-PLUS parameter checking error codes

4.2.3.3 Reading of VS-PLUS parameters

Return value	Meaning	Error message
-12	Controller number is not correct	13, 2, controller number, controller number in parameters, 0
-13	Signal group unknown to the controller	13, 3, signal group number, 0, 0
-14	Unknown detector	13, 4, detector number, 0, 0
-15	File could not be opened	13, 5, 0, 0, 0
-16	Not a VCB file	13, 6, 0, 0, 0
-17	VS-PLUS version is not correct	13, 7, VS-PLUS version, VS-PLUS version in parameters, 0
-18	Unexpected end of file	13, 8, 0, 0, 0
-19	Structure error	13, 9, group, number, 0
-20	Data sets defined incorrectly	13, 10, 63, data sets, 0

Table 110: VS-PLUS parameter reading error codes

4.2.3.4 Writing of VS-PLUS parameters

Return value	Meaning	Error message
-20	File could not be opened	

Table 111: VS-PLUS parameter writing error codes

4.2.4 Turning on VS-PLUS for the entire node

The following descriptions always refer to the entire node. The commands are passed by the function "**VSPSoll[0]**". Commands for partial nodes (VSPSoll[tk]) are sent equally to those for the main node.

"**SchaltBild**" [translated: SwitchPattern] is only used for program changes or switch-off. Only in these cases the variable must contain a value. In all other case the variable should be set to 0 or contain a value as specified.

"**WunschBildBereit**" [translated: DesiredPatternReady] must be set to 0 for all elements. Only VS-PLUS changes this value as soon as the desired pattern is ready

4.2.4.1 Controller switch-on

When a controller switches on the partial nodes have no importance. All commands sent to VS-PLUS are addressed to the total node (VSPSoll[0]).

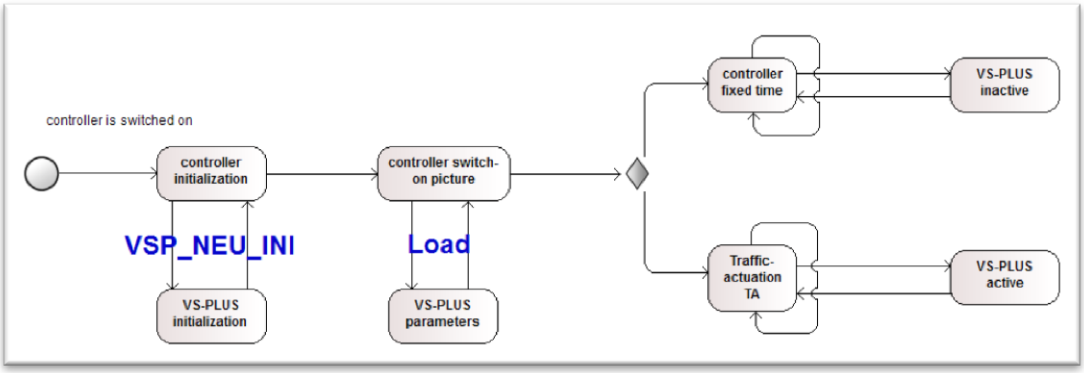


Figure 2: Controller switch-on state diagram

When the controller is switched on VS-PLUS must be initialized in order to guarantee proper operation. The following points should be considered:

- During controller initialization also VS-PLUS must be initialized. VS-PLUS is called once with "**VSP_NEU_INI**".
- During the remaining controller initialization process VS-PLUS is called with "**VSP_AUS"**.
- After this, the VS-PLUS-parameters can be read. The VS-PLUS parameter reading must be terminated by the end of the switch-on picture (for details for VS-PLUS parameter reading see chapter 5.1 "VS-PLUS parameter").
- When the controller has processed the switch-on picture, a decision has to be taken whether a fixed time program shall run or VS-PLUS. In order to find this out, the controller can call the VS-PLUS function "I_Prog_VSP" (see chapter 5.1 "VS-PLUS parameter").

Fixed time: The controller is in command of the lights according to its own parameters. VS-PLUS still needs to be called once per second.

- Traffic-actuation: VS-PLUS is called once per second.

4.2.4.2 Fixed time

Also when the controller runs in fixed-time mode VS-PLUS has to be called. During such calls only a part of the core is processed. The inputs (detectors) are read, but no outputs (signal groups) are generated.

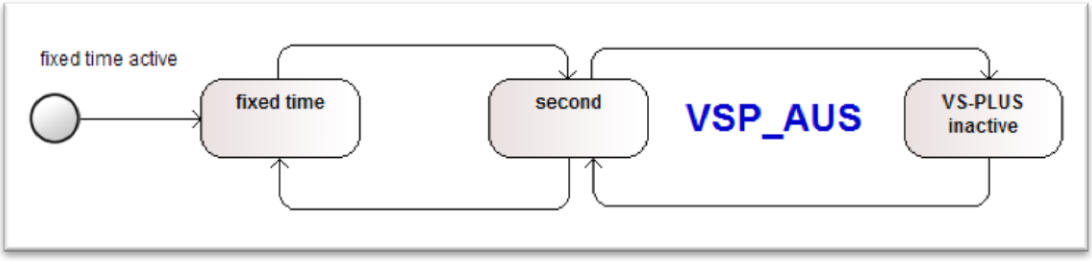


Figure 3: fixed time state diagram

VS-PLUS is called once per second with the command "**VSP_AUS**".

4.2.4.1 Traffic-actuation

When the controller runs in traffic-dependent mode, then VS-PLUS is fully invoked. The inputs are read by VS-PLUS and outputs are controlled by VS-PLUS as well.

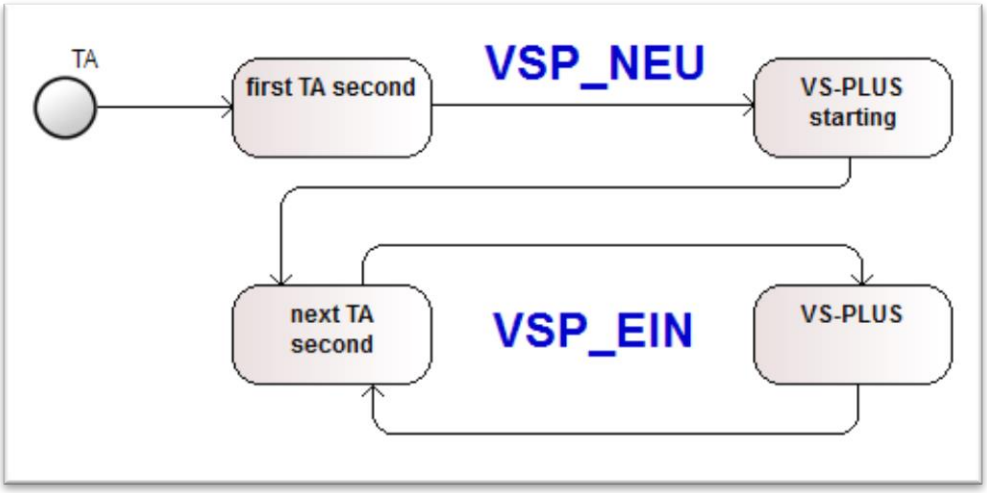


Figure 4: Traffic actuation mode- state diagram

The first time (when starting-up traffic-dependency), VS-PLUS must be called with the command "**VSP_NEU**". All further calls need the command "**VSP_EIN**".

4.2.5 VS-PLUS program switches for the whole node

4.2.5.1 Fixed time to fixed time

Switching from one fixed-time program to another must be executed entirely by the controller. VS-PLUS continues to be called as described in chapter 4.2.4.2 "Fixed time".

4.2.5.2 Fixed time to VS-PLUS

When switching from a fixed-time program to VS-PLUS, the controller switches depending on the switching definitions either immediately or after having reached the switchover point. Once VS-PLUS is to take over control, the procedure described in chapter 4.2.4.1 "Traffic-actuation" is applied.

4.2.5.3 VS-PLUS to VS-PLUS

In this case, no activities by the controller are needed. VS-PLUS is in charge of it.

4.2.5.4 VS-PLUS to fixed time

In this case, VS-PLUS must first be switched to a pre-defined image. When this image is reached, the controller can take over control.

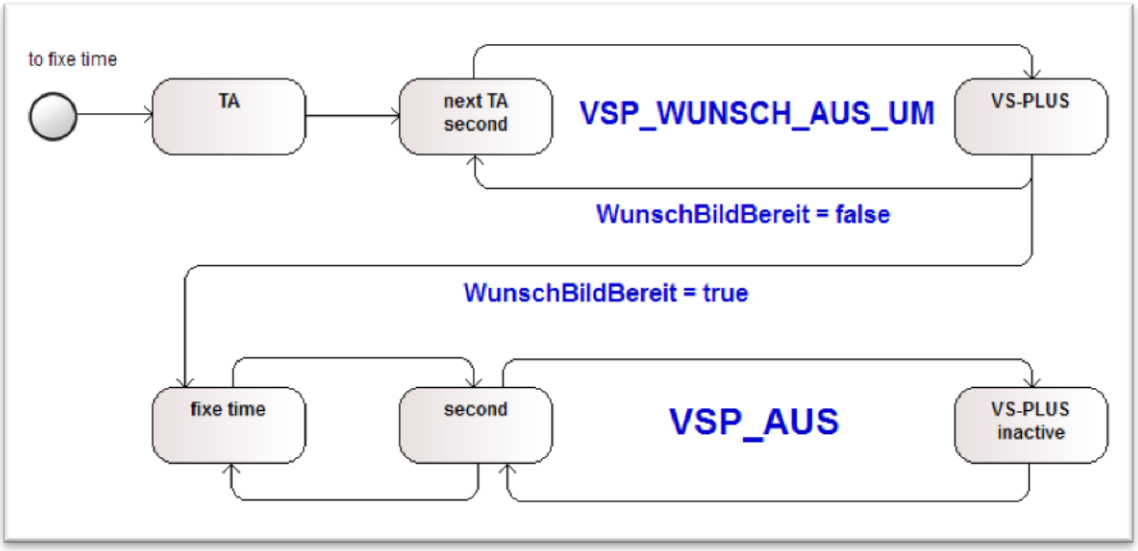


Figure 5: VS-PLUS to fixed time transition on state diagram

VS-PLUS is told with the command "**VSP_WUNSCH_AUS_UM**" to switch into the pre-defined switching image. This signal image is defined by "**SchaltBild**" by defining for each signal group if it has to be switched to red (closed) or green (open).

As long as the required signal image is not reached yet, VS-PLUS returns "false" to the controller by the parameter "**WunschBildBereit**". VS-PLUS needs to be called continuously with the same parameters once per second, until the image is reached.

As soon as the required signal image is reached, VS-PLUS returns "true" to the controller by the parameter "**WunschBildBereit**". Now the controller can take over control and needs to call VS-PLUS from now on in its "inactive" mode.

Due to the parameterization of VS-PLUS, it can happen that a given power-off signal image is never reached. This case should be avoided by initial field tests. In order to avoid any problems during operation, a security mechanism should be implemented: if after a certain period (e.g. a cycle) the desired image has not been reached, the controller takes over control.

4.2.6 Controller switch-off with VS-PLUS for the whole node

4.2.6.1 Switch off from fixed time

Switching off from fixed time is a pure controller task. Depending on the settings the controller is turned off immediately or waits until the cycle has reached the switch-off point. Once the switch-off signal images are displayed, VS-PLUS is no longer called.

4.2.6.2 Switching-off from VS-PLUS

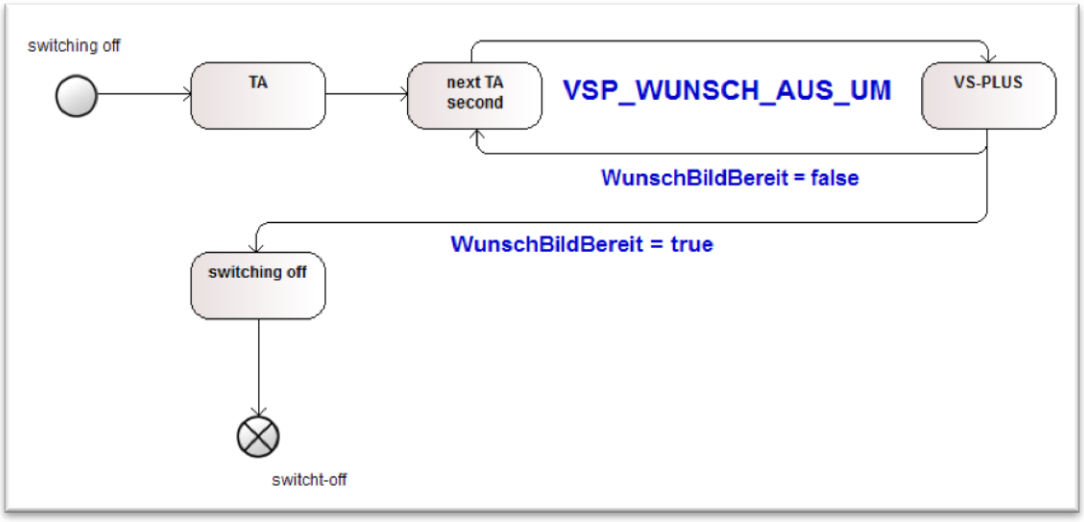


Figure 6: Controller switch-off from VS-PLUS state diagram

VS-PLUS is called with the command "**VSP_WUNSCH_AUS_UM**". Previously "**SchaltBild**" has to be set by the controller in order to tell VS-PLUS the requested switch-off signal image.

As soon as "**WunschBildBereit**" is returned "true" by VS-PLUS, the controller can take over control and switch-off according to a defined procedure. Once the controller has taken over control and has started the switch-off procedure, VS-PLUS needs not to be called any longer.

4.2.7 VS-PLUS program switches for the partial nodes

The VS-PLUS initialization as described in chapter 4.2.4.1 "Controller switch-on" has to be executed also when switching on partial nodes only.

VS-PLUS is not influenced by the controller configuration what partial nodes are allowed to be switched on. In all cases the following two requirements for switching partial node on and off have to be applied.

Important in this context is the function "**HW_VspFreigegeben**". This determines who has the authority to switch what signal groups. The function details are described in chapter 4.6.3 "Enabled signal groups".

VS-PLUS is not affected if a partial node shall be switched off in fixed time mode. Anyway the function "**HW_VspFreigegeben**" has to work as well in this case as described.

4.2.7.1 Partial node switch-off with VS-PLUS

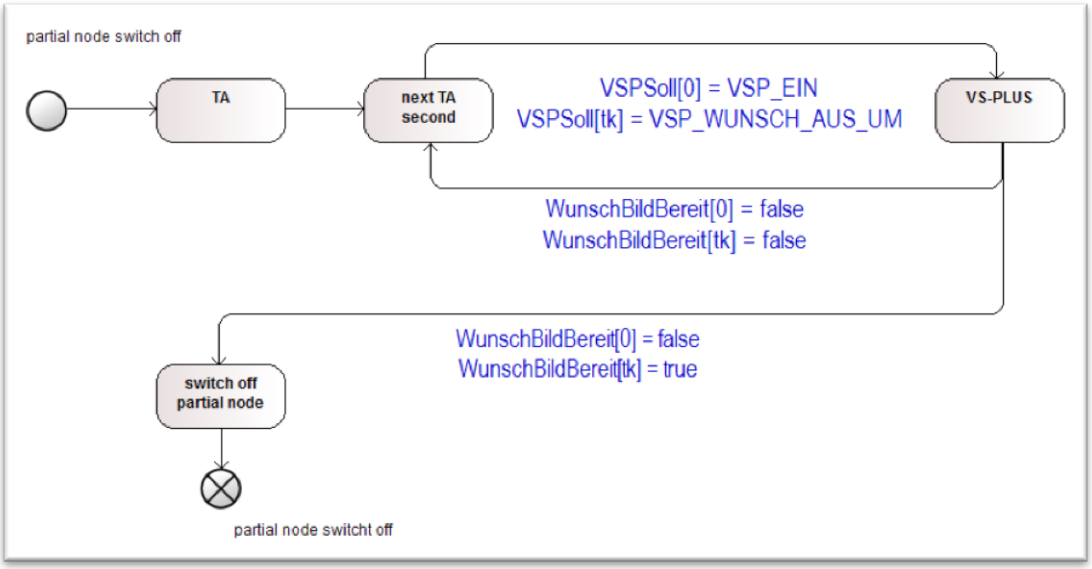


Figure 7: Partial node switch-off with VS-PLUS state diagram

Turning off a partial node is similar to turning off the entire node. The command "**VSP_WUNSCH_AUS_UM**" is called for the corresponding partial node (VSPSoll[tk]).

Following such a command, VS-PLUS switched the whole node into the switch-off signal image that has been defined. "**SchaltBild**" needs to contain a well-defined state for each signal group of the node.

As soon as VS-PLUS has reached the switch-off signal group pattern, the controller reads "true" in the calling parameter "**WunschBildBereit[tk]**". The controller can take over control over the specific partial node. The remaining part of the node can be switched on again as described in chapter 4.2.4.1 "Traffic-actuation". The corresponding partial node **VSPSoll[tk]** is noted as **VSP_AUS**.

4.2.7.2 Partial node switch-on with VS-PLUS

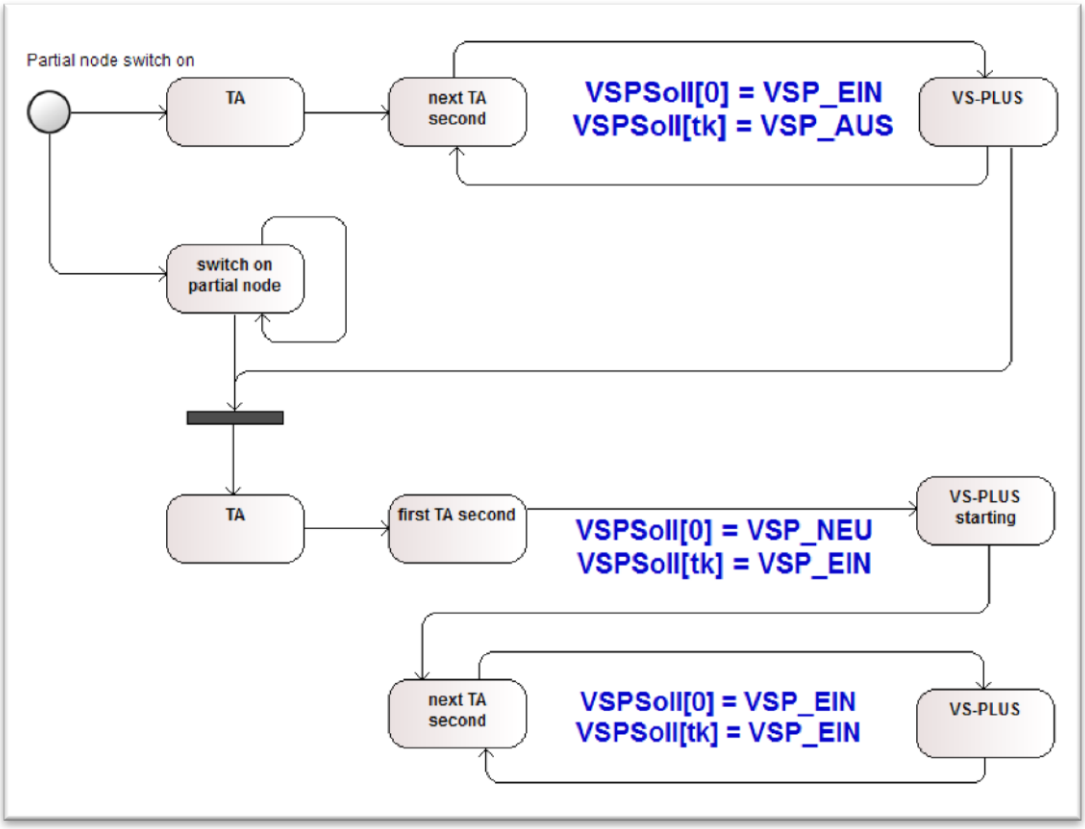


Figure 8: Partial node switch-on with VS-PLUS state diagram

If a partial node shall be switched-on while VS-PLUS is in control, the controller needs to switch the partial node on before and bring the signal groups into the pre-defined switch-on image. As soon as all signal groups have reached the final switch-on state red (closed) or green (open), VS-PLUS can take over the signal group image.

As soon as the partial node is ready to be taken over by VS-PLUS, VS-PLUS has to be called once with the command "**VSP_NEU**". "**VSP_EIN**" is given as parameter value for the corresponding partial node.

It is important to note that while switching-on a partial node, no conflicting signal groups must be shown, i.e. signal groups in conflict to other signal groups shown on the partial nodes that are already under VS-PLUS control. If this is not possible, VS-PLUS should be brought to a predefined switch-off signal image while the partial node is being switched on. This predefined switch-off image must not be in conflict on any signal groups that is used for switching on the partial node.

4.3 Timer

The controller must provide a number of timers. The maximum number is defined in chapter 4.1.5 "Number of timers ".

Each timer must provide the following functions:

- Read current value: returns the time elapsed since the timer start in units of 100ms.
- Load and run with defined value: the timer is started with a given value. From this point of time on the timer value is increased every 100ms.
- Clear: the timer value is set to 0, but the timer continues counting.
- Stop and clear: the timer is sopped and the value is set to 0.
- Stop: the timer is stopped. The value stays on the last value.

4.4 Detectors

4.4.1 Impulse counters

An impulse counter counts all rising slopes. This sum is cleared by VS-PLUS as soon as VS-PLUS has read the value.

4.4.2 Occupancy degree

Current occupancy degree: The current occupancy degree: to what percentage was the detector occupied during the last second?

Smoothened occupancy degree: The smoothened occupancy degree basing on the current occupancy degree according to the following formula:

$$A_n = A_{n-1} + \alpha_x (M - A_{n-1})$$

A_n = smoothened occupancy degree in %
A_{n-1} = last smoothened occupancy degree in %
M = current occupancy degree in %
α_x = factor
α_{up} = if $M > A_{n-1}$
α_{down} = if $M < A_{n-1}$

Figure 9: Occupancy degree calculation formula

4.5 Serial calling points

The PT module can process PT telegrams. The PT module collects calls and check-outs and keeps them in a ring buffer.

It is possible to process serial RBL telegrams (called "serial") and detectors (called "parallel").

The processing is limited to 32 PT groups. Each group can contain a maximum of 5 call points with an additional emergency check-in. Mixing of serial and parallel calls is possible within each group. Overtaking of PT vehicles can be detected with serial calls.

The PT module is considered as one single functional module. It contains call (check-in) and check-out evaluations, computation of PT travel times and data buffer management.

The serial telegrams are passed to VS-PLUS by the function

GibTelegramm(x)
[translation: GetTelegram(x)]

The PT module is capable of fetching a maximum of 5 PT telegrams per second from the controller. If more telegrams are available, the telegram reception unit in the controller has to buffer the telegrams and keep them until VS-PLUS fetches them in the next second(s).

The telegram data structure is as follows:

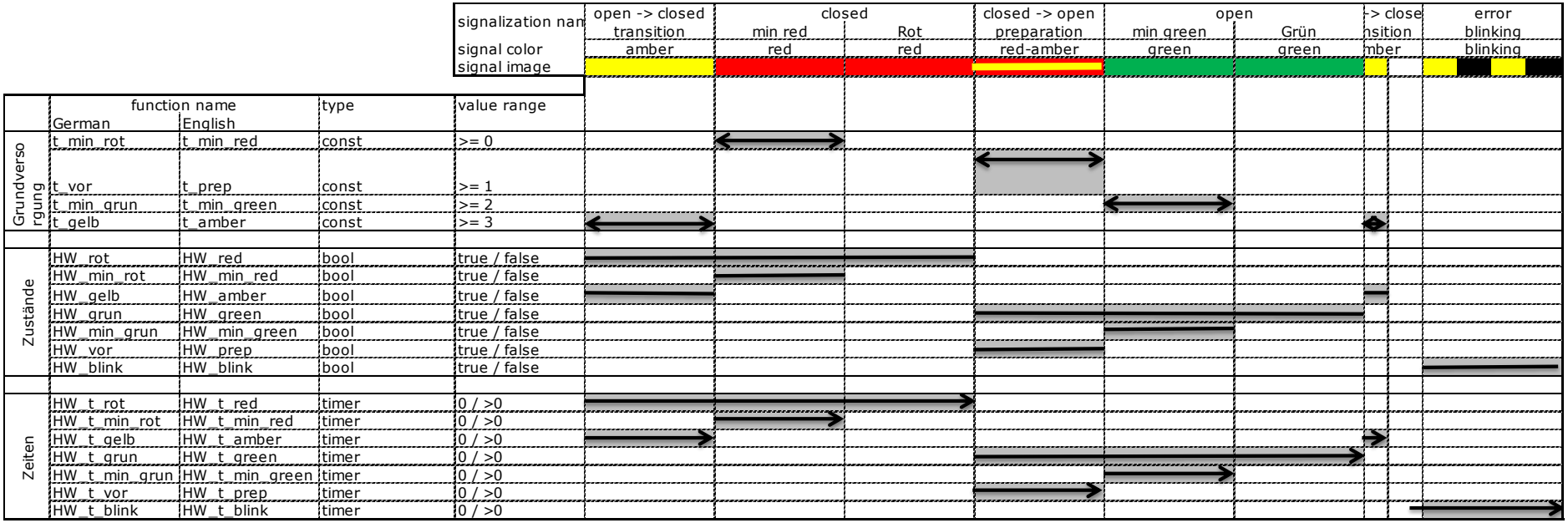
```
struct R09serialTelegram   {
                           int MP;
                           int Linie;
                           int Kurs;
                           int Route;
                           int Prioritaet;
                           int Laenge;
                           int RichtungVonHand;
                           int FahrplanAbweichnung;
                           };
```

- MP call point number (0 – 65535)
- Linie line number (0 - 999)
- Kurs course number (0 - 99)
- Route route number (0 – 999)
- Prioritaet priority (0 - 7)
- Laenge train length (0 – 5)
- RichtungVonHand direction by hand (0 – 3)
- FahrplanAbweichung schedule status in seconds (+ = late, - = early)

A telegram data structure must always be complete when transferred to VS-PLUS. Unknown values have to be initialized with 0.

4.6 Signal groups

4.6.1 Signal group states



	function name German	function name English	type	value range
Grundversorgung	t_min_rot	t_min_red	const	>= 0
	t_vor	t_prep	const	>= 1
	t_min_grun	t_min_green	const	>= 2
	t_gelb	t_amber	const	>= 3
Zustände	HW_rot	HW_red	bool	true / false
	HW_min_rot	HW_min_red	bool	true / false
	HW_gelb	HW_amber	bool	true / false
	HW_grun	HW_green	bool	true / false
	HW_min_grun	HW_min_green	bool	true / false
	HW_vor	HW_prep	bool	true / false
	HW_blink	HW_blink	bool	true / false
Zeiten	HW_t_rot	HW_t_red	timer	0 / >0
	HW_t_min_rot	HW_t_min_red	timer	0 / >0
	HW_t_gelb	HW_t_amber	timer	0 / >0
	HW_t_grun	HW_t_green	timer	0 / >0
	HW_t_min_grun	HW_t_min_green	timer	0 / >0
	HW_t_vor	HW_t_prep	timer	0 / >0
	HW_t_blink	HW_t_blink	timer	0 / >0

Figure 10: Signal group states

The signal group functions summarize all possible functions for the signal groups. The figure defines the return values of each function. In order to guarantee proper functioning of VS-PLUS, the intervals shown have to be respected.

4.6.2 Special case: green blinking

In some countries, the transition from free to closed contains an additional transitional element (for example, in Austria the green flashing). Depending on whether this additional element is part of the open or closed state, VS-PLUS expects different reporting.

4.6.2.1 Green blinking is a closed state

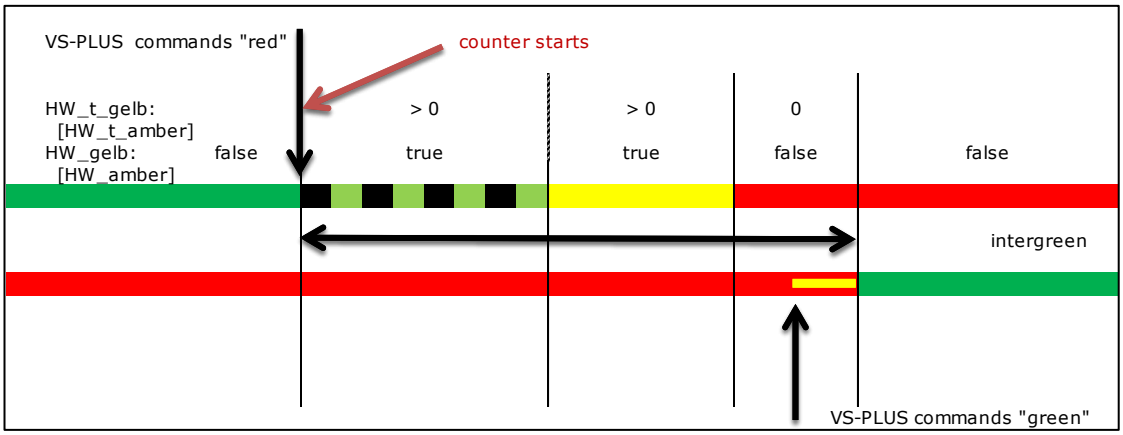


Figure 11: Green blinking is a closed state

If green blinking already belongs to the closed state, the intergreen starting point is at the green end. Therefore the amber time counter "HW_t_amber" starts counting with the beginning of green blinking. The counter is incremented until the end of the transition.

The amber state "HW_amber" becomes true as soon as the green blinking starts.

4.6.2.2 Green blinking is an open state

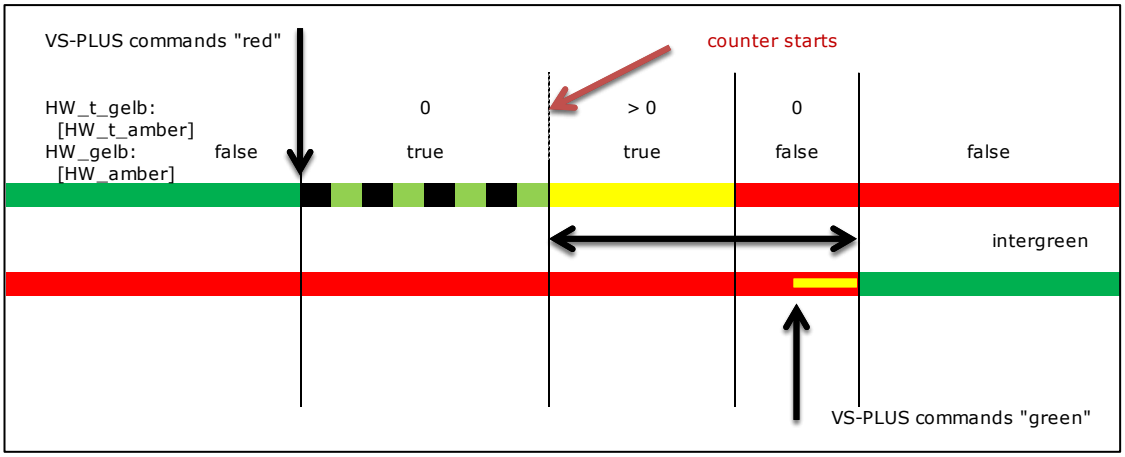


Figure 12: Green blinking is an open state

If green blinking still belongs to the open state, the intergreen starting point is at the green blinking end. Therefore the amber time counter "HW_t_amber" starts counting with the beginning of amber. The counter is incremented until the end of the transition. During green blinking the HW_t_amber counter contains 0.

The amber state "HW_amber" becomes already true as soon as the green blinking starts.

4.6.3 Enabled signal groups

Each signal group that is to be controlled by VS-PLUS needs a controller clearance. The function name is:

HW_VspFreigegeben(x)
[translation: HW_allowed(x)]

This function tells VS-PLUS whether a signal group can be used or not. This function is particularly important when switching partial nodes only.

If a signal group has VS-PLUS clearance (i.e. is allowed for VS-PLUS), the function returns 1, else 0.

If a signal group has no VS-PLUS clearance (any more), VS-PLUS does not control it and does not read its status information and counters.

When the controller is started, the signal groups should have VS-PLUS clearance as soon as the switch-on signal group image is shown. When VS-PLUS is called with the command "**VSP_NEU**", the signal groups must have clearance.

When the controller is switched off, the VS-PLUS clearance can be revoked as soon as the switch-off signal image is shown.

Even when a fixed time program is running on the controller, the VS-PLUS clearance is needed. In this case VS-PLUS does not control the lights, but VS-PLUS must be able to read the actual status and counters in order to be ready to take over control.

This function is particularly useful when switching partial nodes. If a partial node is switched off, the VS-PLUS clearance is canceled when the power-off signal group image is reached. At this point of time the control for these signal groups is taken over by the controller.

If a partial node is switched on again, the VS-PLUS clearance hat to be given before the VS-PLUS is called with the command "**VSP_NEU**". Thus VS-PLUS regains control of the corresponding signal groups.

4.7 Data types

The following data types are used in VS-PLUS:

- unsigned char — unsigned 8 bit variable
- signed char — signed 8 bit variable
- unsigned short — unsigned 16 bit variable
- signed short — signed 16 bit variable
- unsigned long — unsigned 32 bit variable
- signed long — signed 32 bit variable

5 VS-PLUS SUPPLY

5.1 VS-PLUS parameter

5.1.1 Principle

In order to avoid performance problems when reading a VCB file during operation, the corresponding functions are not contained within the VS-PLUS real-time process and placed in a separate source file. This allows checking and reading VCB files in a separate task or threat. This function is called in a separate time frame and thus does not influences the timing of the main function "VSPLUS", or at least just a little.

The memory requested by VS-PLUS must be global. Both the functions described below as well as VS-PLUS must be able to access it. Access is by means of functions which provide pointers to the data areas.

The VS-PLUS parameters read from the VCB file are normally organized in two data areas. Reading takes place in a passive data area. Only when reading has been completed the passive region is activated via pointer switching. This ensures that the parameters are not inconsistent during the reading process. Working with two data areas is important and should be implemented on the controller.

The compiler switch "_ADVANCED_MEMORY_" defines whether a "second data area" is used for VCB file reading. If this switch is not set, the system operates with one data area only for the VS-PLUS parameter.

The source code for this feature is in a separate file, VSPxxx_7.C.

5.1.2 Memory area initialization

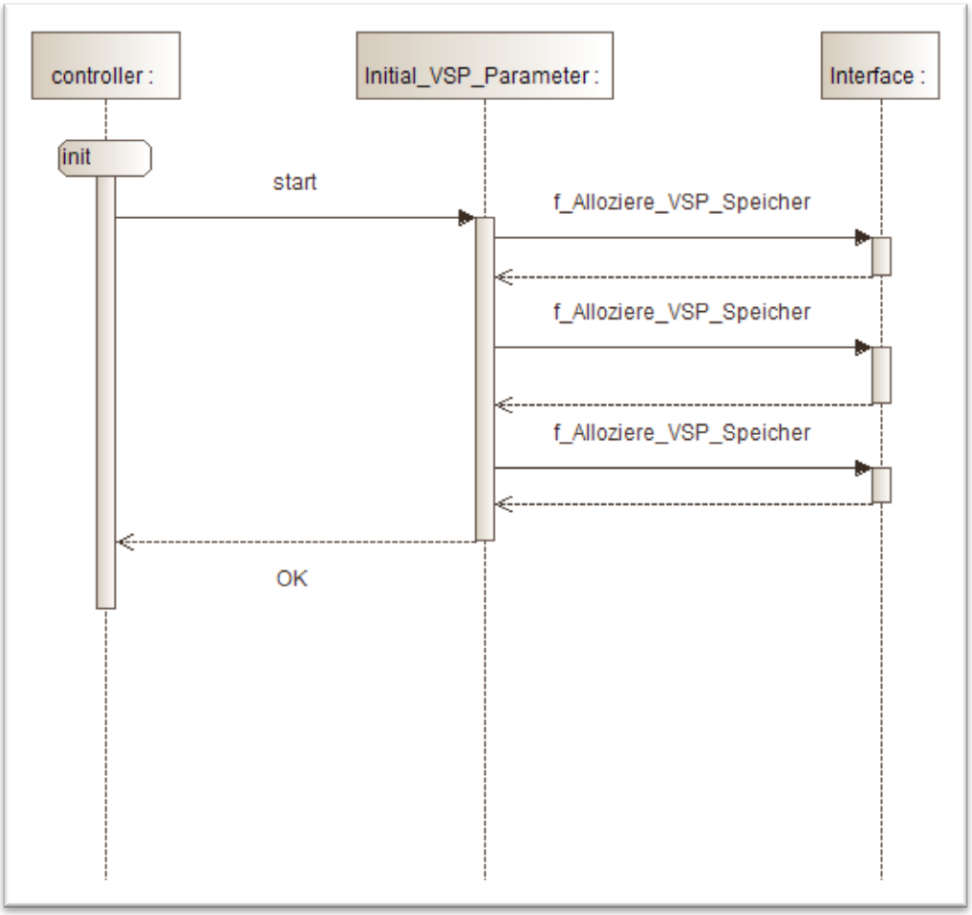


Figure 13: Memory area initialization state diagram

During the controller initialization phase, the VS-PLUS function "**Initial_VSP_Parameter**" has to be done by the controller. This function organizes and initializes the memory for the VS-PLUS parameter.

The memory space request is created by the function "**f_Alloziere_VSP_Speicher**" that VS-PLUS calls for the controller. This function is used 3 times:

- General memory area
- Data area 1
- Data area 2

The memory areas requested by this function must be persistent while the VS-PLUS process is alive.

Additionally VS-PLUS provides the function "**Ende_VSP_Parameter**". The controller can call this function when the system is stopped (shutdown). VS-PLUS answers with the function "**l_Freigeben_VSP_Speicher**" in order to mark the memory are for release.

5.1.3 VCB file

The controller has to handle the VCB file. The file can be transferred via a proprietary supply interface or via an OCIT VD interface. File access location and organization is the responsibility of the controller.

When a controller receives its first parameter supply, a complete supply parameter set has to be loaded onto the controller. While the controller is operating, it is possible to transmit via OCIT VD a new VCB file to the controller that contains only a part of the VS-PLUS parameters.

In order to make sure that there is always a complete set of supply parameter on the controller, VS-PLUS writes the data back to the controller after each reading or update. Each time the controller is restarted VS-PLUS needs to read its parameters from this file. Thus the controller must store the VCB file persistently.

5.1.4 Starting VS-PLUS

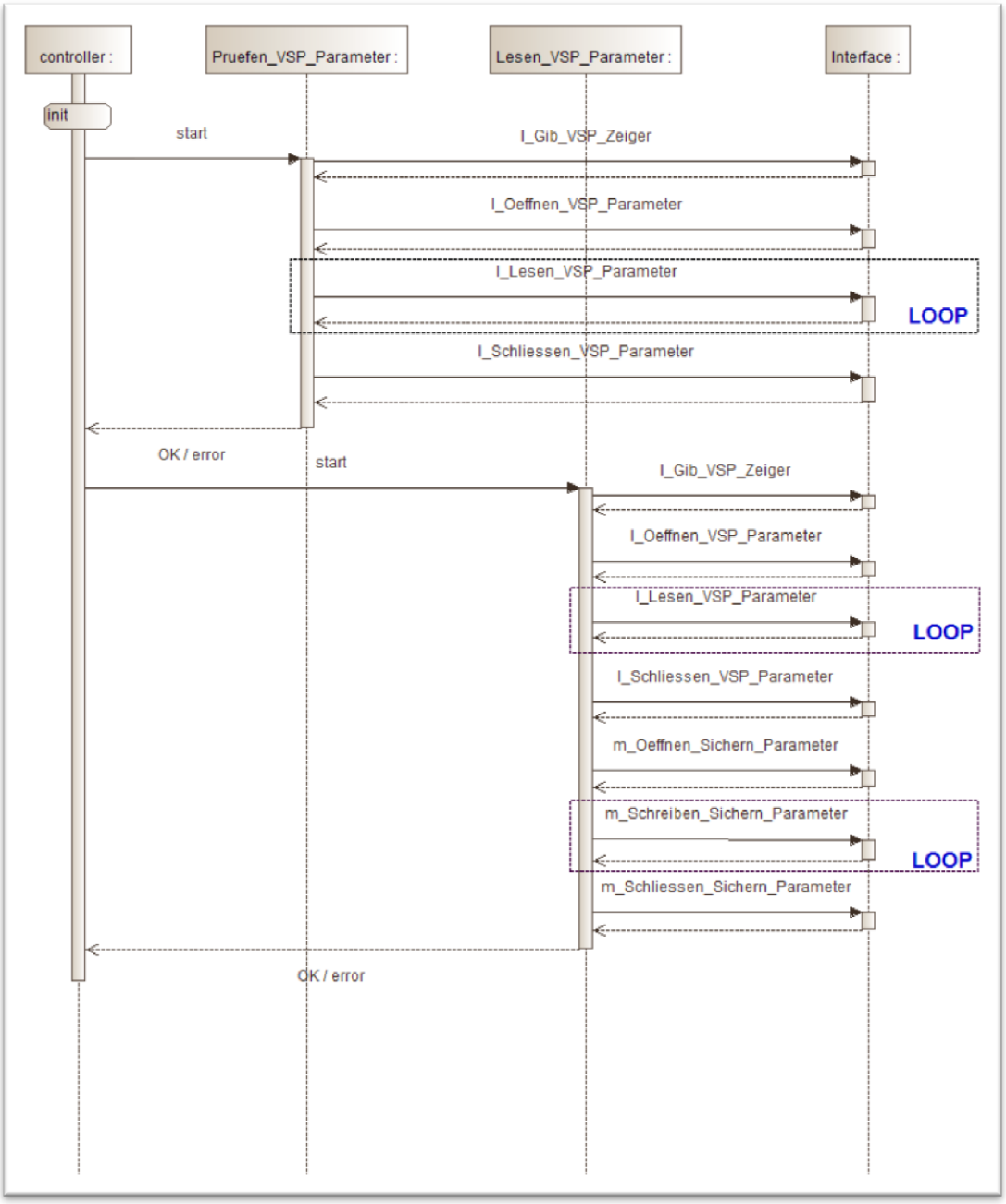


Figure 14: Starting VS-PLUS state diagram

The reading of the VS-PLUS parameter file must be terminated before calling VS-PLUS with the command "**VSP_NEU**". The best sequence is to start with "**VSP_NEU_INI**", then read the parameter file, and only then call VS-PLUS the next time.

The controller has to call 2 VS-PLUS functions in order to read the parameter file properly.

PRUEFEN_VSP_PARAMETER

This function checks VS-PLUS. The function checks if the VS-PLUS parameters belong to the basic supply data of the node. If the check fails, the parameter file cannot be read and VS-

PLUS cannot be started. All possible error messages are described in chapter 4.2.3.2 "Checking of VS-PLUS parameters".

If the check was successful, the function

LESEN_VSP_PARAMETER();

can read the parameter file. Errors at this step can also occur, they error messages are described in chapter 4.2.3.3 "Reading of VS-PLUS parameters".

After successful parameter file reading, a backup file is written back to the controller by VS-PLUS.

5.1.5 New VCB file

It is possible to load a new VCB file to the controller while VS-PLUS is running and to have VS-PLUS read this new data. VS-PLUS needs to check the file first, and then VS-PLUS can read the new data. The steps are identical to the steps when starting VS-PLUS.

The parameters can be activated when VS-PLUS is called the next time.

5.2 VS-PLUS commands

VS-PLUS provides an interface for command files. Command files are used by higher-level systems (as network controller) in order to send special commands to VS-PLUS. As an example a new frame signal plan might be sent to VS-PLUS and activated at once.

The command file format is VCB. The controller must receive such files and hand them over to VS-PLUS. As soon as VS-PLUS has read the command file, the controller can delete the file because there is no further use of it.

The interface must be implemented even when the controller does not provide a command interface to a higher lever system. The interface is necessary for the VS-PLUS certification process.

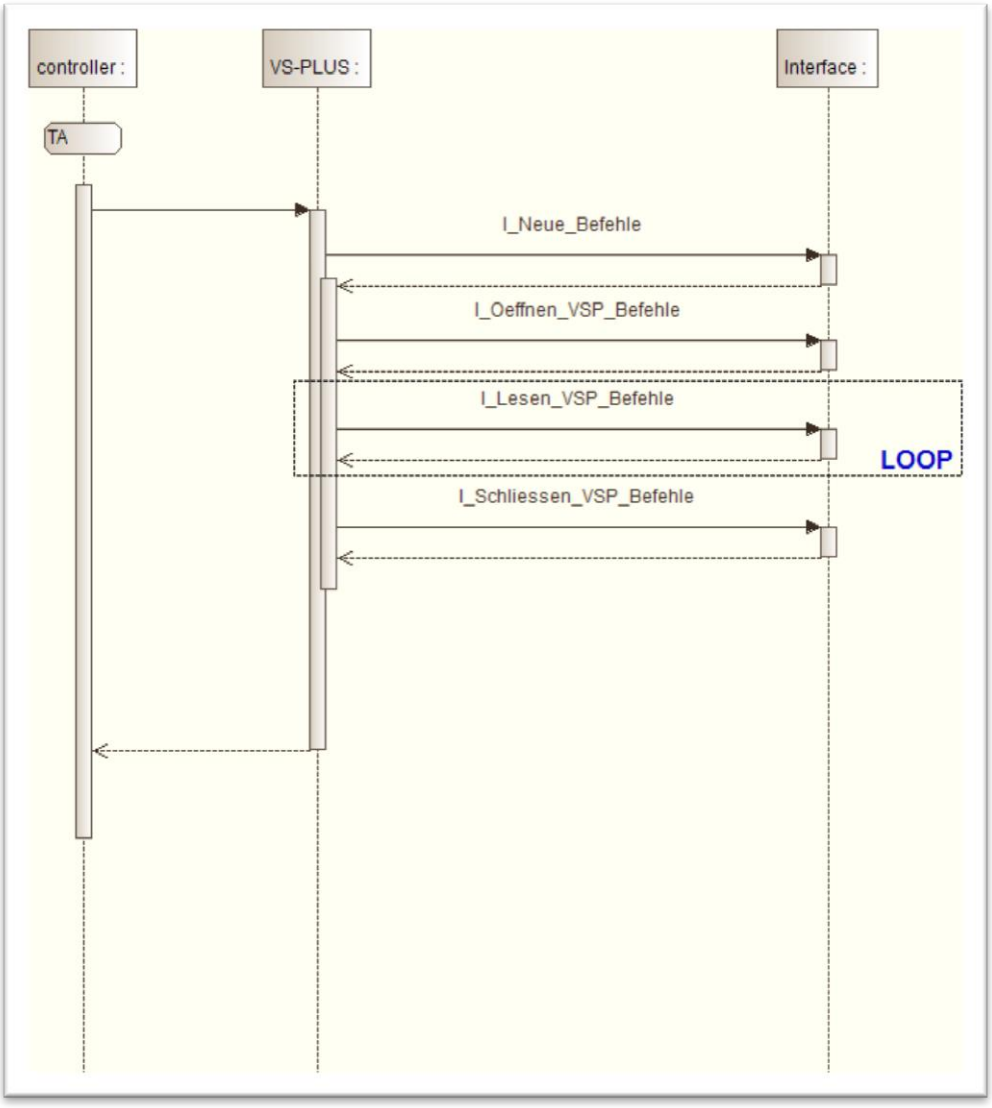


Figure 15: VS-PLUS command processing state diagram

VS-PLUS asks the controller every second if there is a new command file available. If this is true, VS-PLUS reads the file with the corresponding functions and processes the command. The controller may delete the file after VS-PLUS has read it.

5.3 Process data

There is an interface that enables the outside world to look inside VS-PLUS. Everything within VS-PLUS is variables. This interface reads VS-PLUS variables. These variables represent the internal state of the VS-PLUS state machine and also enable precise problem tracking. Controllers normally provide signal group and detector information to the outside world. A controller with VS-PLUS needs to provide the additional VS-PLUS variables to a central system.

VS-PLUS offers the function

f_VSP_ProzessDaten

to the controller for reading VS-PLUS process data.

The function declaration is as follows:

U_WORD f_VSP_ProzessDaten(void *px, void *py)

The function result is controlled by the calling parameters:

- px, NUL: Check for OITD number: checks if the process data point "px" is defined
- px, py: Fetch value: Read the value a particular instance
- NUL, py: Write all defined OITD numbers into a file whose pointer is given in "py"

5.3.1 Check for OITD number

If "px" only is set, px must contain a pointer to the following structure

```
typedef struct _      {
                      long Id;
                      unsigned short Inst
                      U_BYTE KmpInd
                      U_BYTE ErgLaenge
                      } PD_DEF;
```

- Id OITD number (process data type identification number)
- Inst instance number within OITD number (channel number)
- KmpInd for internal VS-PLUS use only
- ErgLaenge for internal VS-PLUS use only

Return value 0 → instance does not exist (OITD number plus channel number)

Return value 65535 → type does not exist (OITD number per se)

All other return values represent the size of the OITD instance in number of bytes.

5.3.2 Fetch value

"px" contains the same contents as before. "py" will contain the return value. Depending on the OITD size "py" must contain a pointer to a 1 or 2 byte memory.

A return value 0 indicates that the value in "py" is valid, 1 indicates invalid.

5.3.3 File

"py" contains a pointer to a FILE instance. VS-PLUS writes all OITD numbers into this file that are valid for the executed VS-PLUS version.

A return value 1 indicates that the file has been written successfully.

INDEX

APPENDIX

APPENDIX 1: IF626BAS.INC

```
/* VS-PLUS 6 Release 2 State 6 Edition 2014 */
/*

              VV         VV     SSSSSS     PPPPPPPPP
              VV         VV   SSS    SSS   PP       PP
              VV         VV   SS      SS   PP       PPP
              VV         VV    SS          PP       PP
               VV       VV       SS       PPPPPPPPP
                VV     VV          SS     PP
                 VV   VV      SS      SS   PP
                  VV VV       SSS    SSS   PP
                   VVV          SSSSSS     PPLUS

            INCLUDE FILE INTERFACE BASIS for VS-PLUS


      File name      : if626bas.inc
      Version        : 6.2.6
      Date           : 15.08.2012
      Programmer     : P. Herren


      Conents:
      ~~~~~~~~
      This file contains the controller-specific function calls

      #define _VSPLUS_SIM_    Definitions for version with simulation
      #define _VSPLUS_STG_    Definitions for version with controller

*/

#ifndef _if626bas
#define _if626bas

#ifdef _EMULATOR_
#include "ovstgdek.h"
#endif

#ifdef _SIEMENS_C900_
#define VSP_NR 11           /* Siemens (WFi) control scheme number */
#define VSP_FEHLER_NR 99    /* Siemens (WFi) own number for VS-PLUS */
#endif

#ifndef OEV_MODUL
#define OEV_MODUL    8
#endif

/*
TIMER DEFINITIONs (Timer base)
*/

#define VSP_T0      0u                  /* Base 0 for timer */
#define VSP_T1      VSP_T0+DETMAX       /* Base 1 for timer */
#define VSP_T2      VSP_T1+DETMAX       /* Base 2 for timer */
#define VSP_T3      VSP_T2+DETMAX       /* Base 3 for timer */
```

```
#define VSP_T4      VSP_T3+VSMAX        /* Base 4 for timer */
#define VSP_T5      VSP_T4+VSMAX        /* Base 5 for timer */
#define VSP_T6      VSP_T5+VSMAX        /* Base 6 for timer */
#define VSP_T7      VSP_T6+VSMAX        /* Base 7 for timer */
#define VSP_TE      VSP_T7+SPEZMAX      /* Last timer (Base 0-7)*/

/*
DATA TYPES VS-PLUS
*/

/*
Variable types
*/
typedef unsigned char  U_BYTE;  /*  8-Bit variable  without V */
typedef signed char    S_BYTE;  /*  8-Bit variable  with V */

#ifdef _EMULATOR_
typedef unsigned char  BYTE;    /* EMULATOR */
#endif

#define US_WORD U_WORD          /* 16-Bit variable without V */
#define SS_WORD S_WORD          /* 16-Bit variable with V */
typedef unsigned short U_WORD;  /* 16-Bit variable without V */
typedef signed short   S_WORD;  /* 16-Bit variable with V  */
#define UL_WORD U_WORD          /* 16-Bit variable without V */
#define SL_WORD S_WORD          /* 16-Bit variable with V */

#ifdef _EMULATOR_
#define WORD U_WORD             /* 16-Bit variable without V */
/* #define MAXWORD 0xFFFF */
#endif

typedef unsigned long  U_LONG;  /* 32-Bit variable without V */
typedef signed long    S_LONG;  /* 32-Bit variable with V */

#ifdef _EMULATOR_
typedef unsigned long  ULONG;   /* EMULATOR */
#endif

/*
Logical
*/
typedef enum {L_falsch, L_richtig} Logisch;

/*
Time
*/
typedef US_WORD Zeit;
typedef SS_WORD DZeit;
typedef S_WORD  LZeit;

/*
PROTOTYPES
*/

#ifdef _VSPLUS_SIM_
/*
Prototypes for simulation functions
*/
```

```
extern void Meldung (short int nr, short int par1, short int par2, short
int par3, short int par4);
extern void MeldungNET(unsigned char nr, unsigned short Anr, unsigned char
par1, unsigned char par2, unsigned char par3, unsigned char par4, unsigned
char par5);
extern int   OePNV_Ein_Aus(void);
extern void  OePNV_Ein_Aus_Ist (int OV_Bevorzugung);
extern int   IV_Ein_Aus(void);
extern void  IV_Ein_Aus_Ist (int IV_Bevorzugung);
extern int   Det_Aktiv(int KanalNummer);
extern int   Sg_Aktiv(int KanalNummer);
extern int   AktuelleZeit(int* Stunde, int* Minute, int* Sekunde);
extern int   AktuellesDatum(int* Jahr, int* Monat, int* Tag, int* Wochen-
tag);
extern int   Get_OCITOutstationId(int* ZNr, int* FNr, int* Realknoten);
extern int   ProgrammWahlZentrale(void);
extern int   Neue_Befehle(void);
extern int   Oeffnen_VSP_Befehle(void);
extern int   Lesen_VSP_Befehle(char* data, int _sizeof);
extern void  Schliessen_VSP_Befehle(void);
extern int   Wunsch_VSPLUS(int Wunsch, int Teiknoten);
extern void* Allozieren_VSP_Speicher(int _sizeof, int ID);
extern void* Gib_VSP_Zeiger(int ID);
extern int   Oeffnen_VSP_Parameter(void);
extern int   Read_VSP_Parameter(char* data, int _sizeof);
extern void  Schliessen_VSP_Parameter(void);
extern int   Oeffnen_Sichern_Parameter(void);
extern void  Schreiben_Sichern_Parameter(char* data, int _sizeof);
extern void  Schliessen_Sichern_Parameter(void);
extern void  Freigeben_VSP_Speicher(int ID);
#else
/*
Controller
*/
/*
Program functions
*/
short int ProgrammAktuell(void);
short int ProgrammWahl(void);
short int Zykluszeit(void);
short int Umlaufzeit(void);
int       ProgrammWahlZentrale(void);
short int timer(short int funktion, short int timer);
short int timer_2(short int funktion, short int timer, short int wert);
/*
Detector functions
*/
short int d_imp(short int det);
void      d_limp(short int det);
short int d_belga(short int det);
short int d_stoer(short int det);
short int d_belgg(short int det);
short int d_blg(short int det);
short int d_ztlkn(short int det);
short int d_blgzt(short int det);
short int d_impab(short int type, short int det);
short int d_kvalue(short int det);
short     d_zeitlb(short det);
```

```
/*
Signal-group functions
*/
short int          min_rot(short int sg);
short int          u_rot_gelb(short int sg);
short int          min_gruen(short int sg);
short int          u_gelb(short int sg);
void               SG_ein(short int sg);
void               SG_aus(short int sg);
void               Relais_ein(short int sg);
void               Relais_aus(short int sg);
void               Blinker_ein(short int sg);
void               Blinker_aus(short int sg);
short int          s_rot(short int sg);
short int          s_sr_aus(short int re);
short int          s_sb_aus(short int bli);
short int          s_min_rot(short int sg);
short int          s_gelb(short int sg);
short int          s_grun(short int sg);
short int          s_sr_ein(short int re);
short int          s_sb_ein(short int bli);
short int          s_min_grun(short int sg);
short int          s_vor(short int sg);
short int          s_stoeblink(short int sg);
unsigned short     s_t_rot(short int sg);
unsigned short int s_t_gelb(short int sg);
unsigned short int s_t_min_rot(short int sg);
unsigned short int s_t_grun(short int sg);
unsigned short int s_t_vor(short int sg);
unsigned short int s_t_min_grun(short int sg);
unsigned short int s_HW_VspFreigegeben(short int sg);
short int          s_zwi_zeit(short sgr, short sge);
/*
Error messages
*/
void Meldung(short int degree, short int nr, short int par1, short int
par2, short int par3, short int par4);
void MeldungNET(short int degree, unsigned char nr, unsigned short Anr,
unsigned char par1, unsigned char par2, unsigned char par3, unsigned char
par4, unsigned char par5);
void U_Kontrolle(short int vs, short int zeit);
/*
System functions
*/
short              TelegrammVomGeraet(void* oev_tele_poi);
unsigned short int s_SteuerungNichtAktiv(void);
/*
Data handling
*/
void*              Allozieren_VSP_Speicher(int _sizeof, int id);
void*              Gib_VSP_Zeiger(int id);
void               Freigeben_VSP_Speicher(int id);
int                Oeffnen_VSP_Parameter(void);
int                Read_VSP_Parameter(char* data, int _sizeof);
void               Schliessen_VSP_Parameter(void);
int                Oeffnen_Sichern_Parameter(void);
void               Schreiben_Sichern_Parameter(char* data, int _sizeof);
void               Schliessen_Sichern_Parameter(void);
int                Oeffnen_VSP_Befehle(void);
int                Lesen_VSP_Befehle(char* data, int _sizeof);
```

```
void                Schliessen_VSP_Befehle(void);
/*
OCIT functions
*/
int                 OePNV_Ein_Aus(void);
int                 IV_Ein_Aus(void);
int                 Det_Aktiv(int KanalNummer);
int                 Sg_Aktiv(int KanalNummer);
int                 AktuelleZeit(int* Stunde, int* Minute, int* Sekunde);
int AktuellesDatum(int* Jahr, int* Monat, int* Tag, int* Wochentag);
int Get_OCITOutstationId(int* ZNr, int* FNr, int* Realknoten);
int                 Wunsch_VSPLUS(int Wunsch, int Teiknoten);
int                 Neue_Befehle(void);
void ZSondereingriffvn(unsigned char* Sondereingriff, unsigned long* End-
Zeitpunkt, unsigned long* VorgangsNummer);
unsigned long       UTCZeitstempel(void);
#endif
/*
Prototypes for functions within VS-PLUS
*/
int                 l_Det_Wert(int KanalNummer, int Typ);
int                 m_Prog_Schaltung_erlaubt(void);
int                 l_Prog_VSP(int ProgNummer);
int                 f_Initial_VSP_Parameter(void);
int                 f_Pruefen_VSP_Parameter(void);
int                 f_Lesen_VSP_Parameter(void);
void                f_Ende_VSP_Parameter(void);
int                 l_VSP_X_Ein_Aus_Ist(int T);
char*               l_VABefehlPfad(void);
/*
Specific to controller manufacturer
*/
/*
Siemens C900
*/
#ifdef _SIEMENS_C900_
/*
Example for "typdef struct" (GNU Compiler Siemens)
"typedef _PACKED_V_ struct _PACKED_N_" --> "struct __attribute__((packed)"
*/
#define _PACKED_V_
#define _PACKED_N_ __attribute__((packed))
#else
#define _PACKED_V_
#define _PACKED_N_
#endif

/*
DEFINITIONS for PT module
*/

#define OMAXBYTE 255 /* in case of unsigned char */
#define OMAXWORD 65535U /* in case of unsigned short (2byte) */
/*
Language for PT module
*/
#ifdef _VSPLUS_SIM_
#define SYSTEMLAGUAGE 49
#else
#define SYSTEMLAGUAGE 49
```

```
#endif

/*
DEFINITIONS for partial intersections
*/

#ifdef _VSPLUS_SIM_
#define GERAET_TEILKNOTEN_MAX 1
#else
#define GERAET_TEILKNOTEN_MAX 4
#endif

/*
TIMER
*/

/*
Detector waiting time
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define t_twdet(x)     timer(1,x+(VSP_T0),0) /* -read                      */
#define m_stwdet(x,k)  timer(2,x+(VSP_T0),k) /* -load and start with k     */
#define m_ltwdet(x)    timer(3,x+(VSP_T0),0) /* -clear                     */
#define m_altwdet(x)   timer(4,x+(VSP_T0),0) /* -stop and clear            */
#define m_atwdet(x)    timer(5,x+(VSP_T0),0) /* -stop                      */
#else
/*
Controller
*/
#define t_twdet(x)     timer(1,x+(VSP_T0))     /* -read                   */
#define m_stwdet(x,k)  timer_2(2,x+(VSP_T0),k) /* -load and start with k  */
#define m_ltwdet(x)    timer(3,x+(VSP_T0))     /* -clear                  */
#define m_altwdet(x)   timer(4,x+(VSP_T0))     /* -stop and clear         */
#define m_atwdet(x)    timer(5,x+(VSP_T0))     /* -stop                   */
#endif
/*
Detector waiting time for 1st follow-up haul
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define t_twdet1(x)    timer(1,x+(VSP_T1),0) /* -read                      */
#define m_stwdet1(x,k) timer(2,x+(VSP_T1),k) /* -load and start with k     */
#define m_ltwdet1(x)   timer(3,x+(VSP_T1),0) /* -clear                     */
#define m_altwdet1(x)  timer(4,x+(VSP_T1),0) /* -stop and clear            */
#define m_atwdet1(x)   timer(5,x+(VSP_T1),0) /* -stop                      */
#else
/*
Controller
*/
#define t_twdet1(x)    timer(1,x+(VSP_T1))     /* -read                   */
#define m_stwdet1(x,k) timer_2(2,x+(VSP_T1),k) /* -load and start with k  */
#define m_ltwdet1(x)   timer(3,x+(VSP_T1))     /* -clear                  */
#define m_altwdet1(x)  timer(4,x+(VSP_T1))     /* -stop and clear         */
#define m_atwdet1(x)   timer(5,x+(VSP_T1))     /* -stop                   */
#endif
```

```
/*
Detector waiting time for 2nd follow-up haul
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define t_twdet2(x)    timer(1,x+(VSP_T2),0) /* -read                      */
#define m_stwdet2(x,k) timer(2,x+(VSP_T2),k) /* -load and start with k     */
#define m_ltwdet2(x)   timer(3,x+(VSP_T2),0) /* -clear                     */
#define m_altwdet2(x)  timer(4,x+(VSP_T2),0) /* -stop and clear            */
#define m_atwdet2(x)   timer(5,x+(VSP_T2),0) /* -stop                      */
#else
/*
Controller
*/
#define t_twdet2(x)    timer(1,x+(VSP_T2))     /* -read                   */
#define m_stwdet2(x,k) timer_2(2,x+(VSP_T2),k) /* -load and start with k  */
#define m_ltwdet2(x)   timer(3,x+(VSP_T2))     /* -clear                  */
#define m_altwdet2(x)  timer(4,x+(VSP_T2))     /* -stop and clear         */
#define m_atwdet2(x)   timer(5,x+(VSP_T2))     /* -stop                   */
#endif
/*
Traffic stream waiting time
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define t_twvs(y)      timer(1,y+(VSP_T3),0) /* -read                      */
#define m_stwvs(y,k)   timer(2,y+(VSP_T3),k) /* -load and start with k     */
#define m_ltwvs(y)     timer(3,y+(VSP_T3),0) /* -clear                     */
#define m_altwvs(y)    timer(4,y+(VSP_T3),0) /* -stop and clear            */
#define m_atwvs(y)     timer(5,y+(VSP_T3),0) /* -stop                      */
#else
/*
Controller
*/
#define t_twvs(y)      timer(1,y+(VSP_T3))     /* -read                   */
#define m_stwvs(y,k)   timer_2(2,y+(VSP_T3),k) /* -load and start k       */
#define m_ltwvs(y)     timer(3,y+(VSP_T3))     /* -clear                  */
#define m_altwvs(y)    timer(4,y+(VSP_T3))     /* -stop and clear         */
#define m_atwvs(y)     timer(5,y+(VSP_T3))     /* -stop                   */
#endif
/*
Digital output switch-on - switch-off time
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define t_anzt(y)      timer(1,y+(VSP_T4),0) /* -read                      */
#define m_sanzt(y,k)   timer(2,y+(VSP_T4),k) /* -load and start with k     */
#define m_lanzt(y)     timer(3,y+(VSP_T4),0) /* -clear                     */
#define m_alanzt(y)    timer(4,y+(VSP_T4),0) /* -stop and clear            */
#define m_aanzt(y)     timer(5,y+(VSP_T4),0) /* -stop                      */
#else
/*
Controller
*/
```

```
#define t_anzt(y)      timer(1,y+(VSP_T4))     /* -read                    */
#define m_sanzt(y,k)   timer_2(2,y+(VSP_T4),k) /* -load and start with k   */
#define m_lanzt(y)     timer(3,y+(VSP_T4))     /* -clear                   */
#define m_alanzt(y)    timer(4,y+(VSP_T4))     /* -stop and clear          */
#define m_aanzt(y)     timer(5,y+(VSP_T4))     /* -stop                    */
#endif
/*
Signal time counter (red green) for traffic stream
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define t_vst(y)       timer(1,y+(VSP_T5),0) /* -read                     */
#define m_svst(y,k)    timer(2,y+(VSP_T5),k) /* -load and start with k    */
#define m_lvst(y)      timer(3,y+(VSP_T5),0) /* -clear                    */
#define m_alvst(y)     timer(4,y+(VSP_T5),0) /* -stop and clear           */
#define m_avst(y)      timer(5,y+(VSP_T5),0) /* -stop                     */
#else
/*
Controller
*/
#define t_vst(y)       timer(1,y+(VSP_T5))     /* -read                    */
#define m_svst(y,k)    timer_2(2,y+(VSP_T5),k) /* -load and start with k   */
#define m_lvst(y)      timer(3,y+(VSP_T5))     /* -clear                   */
#define m_alvst(y)     timer(4,y+(VSP_T5))     /* -stop and clear          */
#define m_avst(y)      timer(5,y+(VSP_T5))     /* -stop                    */
#endif
/*
Signal time counter (prepraration/amber) for traffic stream
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define t_vsvort(y)    timer(1,y+(VSP_T6),0) /* -read                     */
#define m_svsvort(y,k) timer(2,y+(VSP_T6),k) /* -load and start with k    */
#define m_lvsvort(y)   timer(3,y+(VSP_T6),0) /* -clear                    */
#define m_alvsvort(y)  timer(4,y+(VSP_T6),0) /* -stop and clear           */
#define m_avsvort(y)   timer(5,y+(VSP_T6),0) /* -stop                     */
#else
/*
Controller
*/
#define t_vsvort(y)    timer(1,y+(VSP_T6))     /* -read                    */
#define m_svsvort(y,k) timer_2(2,y+(VSP_T6),k) /* -load and start with k   */
#define m_lvsvort(y)   timer(3,y+(VSP_T6))     /* -clear                   */
#define m_alvsvort(y)  timer(4,y+(VSP_T6))     /* -stop and clear          */
#define m_avsvort(y)   timer(5,y+(VSP_T6))     /* -stop                    */
#endif
/*
Special timer (for special functions)
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define t_spezt(y)     timer(1,y+(VSP_T7),0) /* -read                     */
#define m_sspezt(y,k)  timer(2,y+(VSP_T7),k) /* -load and start with k    */
#define m_lspezt(y)    timer(3,y+(VSP_T7),0) /* -clear                    */
```

```
#define m_alspezt(y)   timer(4,y+(VSP_T7),0) /* -stop and clear          */
#define m_aspezt(y)    timer(5,y+(VSP_T7),0) /* -stop                    */
#else
/*
Controller
*/
#define t_spezt(y)     timer(1,y+(VSP_T7))     /* -read                  */
#define m_sspezt(y,k)  timer_2(2,y+(VSP_T7),k) /* -load and start with k */
#define m_lspezt(y)    timer(3,y+(VSP_T7))     /* -clear                 */
#define m_alspezt(y)   timer(4,y+(VSP_T7))     /* -stop and clear        */
#define m_aspezt(y)    timer(5,y+(VSP_T7))     /* -stop                  */
#endif

/*
BASE FUNCTIONS
*/

/*
Program functions
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
/* Actual signal-program */
#define f_prg()              ProgrammAktuell()
/* Selected signal-program */
#define f_prgwl()            ProgrammWahl()
#else
/*
Controller
*/
/* Actual signal-program */
#define f_prg()              ProgrammAktuell()
/* Selected signal-program */
#define f_prgwl()            ProgrammWahl()
#endif
/*
Cycle control
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define f_ur(vs,zeit)        U_Kontrolle(vs, zeit)
#else
/*
Controller
*/
#define f_ur(vs,zeit)       U_Kontrolle(vs,zeit)
#endif
/*
Framework signal-plan functions
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
/* Current cycle second (TX) */
#define l_zytim()            Zykluszeit()
```

```
/* Cycle time (TU) */
#define l_umzt()            Umlaufzeit()
#else
/*
Controller
*/
/* Current cycle second (TX) */
#define l_zytim()           Zykluszeit()
/* Cycle time (TU) */
#define l_umzt()            Umlaufzeit()
#endif
/*
Detector functions
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
/* Sum impulse storage (ZS-Value) */
#define l_imp(x)            d_imp(x)
/* Reset impulse storage (ZS-Value) */
#define m_limp(x)           d_limp(x)
/* Sum impulse storage (SS-Value) */
#define l_impss(x)          (0)
/* Reset impulse storage (SS-Value) */
#define m_limpss(x)         (0)
/* Sum impulse falling slopes  */
#define l_impab(x)          d_impab(1,x)
/* Reset sum of impulse falling slopes */
#define m_limpab(x)         d_impab(2,x)
#else
/*
Controller
*/
/* Sum impulse storage (ZS-Value) */
#define l_imp(x)            d_imp(x)
/* Reset impulse storage (ZS-Value) */
#define m_limp(x)           d_limp(x)
/* Sum impulse storage (SS-Value) */
#define l_impss(x)          0
/* Reset impulse storage (SS-Value) */
#define m_limpss(x)         0
/* Sum impulse falling slopes   */
#define l_impab(x)          d_impab(1,x)
/* Reset sum of impulse falling slopes */
#define m_limpab(x)         d_impab(2,x)
#endif
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
/* Current percentage of occupancy */
#define l_belga(x)          d_belga(x)
/* Smoothened occupancy degree (percentage) */
#define l_belgg(x)          d_belgg(x)
/*
Load, derived from traffic situation (GPS data, in %) */
#define l_kvalue(x)         d_kvalue(x)
#else
```

```
/*
Controller
*/
/* Current percentage of occupancy */
#define l_belga(x)          d_belga(x)
/* Smoothened occupancy degree (percentage) */
#define l_belgg(x)          d_belgg(x)
/*
Load, derived from traffic situation (GPS data, in %) */
#define l_kvalue(x)         d_kvalue(x)
#endif
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
/* Current state of occupancy */
#define l_belza(x)          d_blg(x)
#else
/*
Controller
*/
/* Current state of occupancy */
#define l_belza(x)          d_blg(x)
#endif
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
/* Net time gap in 1/10 s    */
#define t_zeitln(x)         d_ztlkn(x)
/* Gross time gap in 1/10 s    */
#define t_zeitlb(x)     (0)
#else
/*
Controller
*/
/* Net time gap in 1/10 s */
#define t_zeitln(x)         d_ztlkn(x)
/* Gross time gap in 1/10 s */
#define t_zeitlb(x)         d_zeitlb(x)
#endif
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
/* Occupancy time in 1/10 s */
#define t_belzt(x)          d_blgzt(x)
#else
/*
Controller
*/
/* Occupancy time in 1/10 s */
#define t_belzt(x)          d_blgzt(x)
#endif
/*
Hardware detector fault
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
```

```
*/
/* Hardware detector fault */
#define l_stoer(x)          d_stoer(x)
#else
/*
Controller
*/
/* Hardware detector fault */
#define l_stoer(x)          d_stoer(x)
#endif
/*
Signal-group functions
*/
/*
Base values
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
/* minimum red time */
#define t_min_rot(x)         min_rot(x)
/* preparation time */
#define t_vor(x)             u_rot_gelb(x)
/* minimum green time */
#define t_min_grun(x)        min_gruen(x)
/* amber time */
#define t_gelb(x)            u_gelb(x)
#else
/*
Controller
*/
/* minimum red time */
#define t_min_rot(x)          min_rot(x)
/* preparation time */
#define t_vor(x)              u_rot_gelb(x)
/* minimum green time */
#define t_min_grun(x)         min_gruen(x)
/* amber time */
#define t_gelb(x)             u_gelb(x)
#endif
/*
Switching commands: using signal colors
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
/* signal group on */
#define EIN_Signal(x)       SG_ein(x)
/* signal group off */
#define AUS_Signal(x)       SG_aus(x)
/* digital output on */
#define EIN_SState(x)       Relais_ein(x)
/* digital output off */
#define AUS_SState(x)       Relais_aus(x)
/* digital blinker on */
#define EIN_SBlink(x)       Blinker_ein(x)
/* digital blinker off */
#define AUS_SBlink(x)       Blinker_aus(x)
```

```
#else
/*
Controller
*/
/* signal group on */
#define EIN_Signal(x)       SG_ein(x)
/* signal group off */
#define AUS_Signal(x)       SG_aus(x)
/* digitl output on */
#define EIN_SState(x)       Relais_ein(x)
/* digital output off */
#define AUS_SState(x)       Relais_aus(x)
/* digital blinker on */
#define EIN_SBlink(x)       Blinker_ein(x)
/* digital blinker off */
#define AUS_SBlink(x)       Blinker_aus(x)
#endif
/*
Switching commands: special operations, these settings are valid for both
contoller and simulation
*/
#define Signal_Dunkel(x) 0  /* Switch signal group to dark */
#define Signal_Blinken(x) 0 /* Switch signal group to blinking */
#define Signal_FarbeRot(x) 0 /* Switch signal group to red after dark or
blinking */
#define Signal_FarbeGruen(x) 0 /* Switch signal group to green after dark
or blinking */
/*
Current signal state
*/
/*
Current state
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
/* Get current state whether signal-group shows red */
#define HW_rot(x)           s_rot(x)
/* Get current state whether digital output is off */
#define HW_sr_aus(x)        s_sr_aus(x)
/* Get current state whether digital blinker is off */
#define HW_sb_aus(x)        s_sb_aus(x)
/* Get current state whether minimal red time elapsed */
#define HW_min_rot(x)       s_min_rot(x)
/* Get current state whether signal-group shows amber */
#define HW_gelb(x)          s_gelb(x)
/* Get current state whether signal-group shows green */
#define HW_grun(x)          s_grun(x)
/* Get current state whether digital output is on */
#define HW_sr_ein(x)        s_sr_ein(x)
/* Get current state whether digital blinker is on */
#define HW_sb_ein(x)        s_sb_ein(x)

/* Get current state whether minimal green time elapsed */
#define HW_min_grun(x)      s_min_grun(x)
/* Get current state whether signal-group is in transition open-closed */
#define HW_vor(x)           s_vor(x)
#else
```

```
/*
Controller
*/
/* Get current state whether signal-group shows red */
#define HW_rot(x)           s_rot(x)
/* Get current state whether digital output is off */
#define HW_sr_aus(x)        s_sr_aus(x)
/* Get current state whether digital blinker is off */
#define HW_sb_aus(x)        s_sb_aus(x)
/* Get current state whether minimal red time elapsed */
#define HW_min_rot(x)       s_min_rot(x)
/* Get current state whether signal-group shows amber */
#define HW_gelb(x)          s_gelb(x)
/*  Get current state whether signal-group shows green */
#define HW_grun(x)          s_grun(x)
/* Get current state whether digital output is on */
#define HW_sr_ein(x)        s_sr_ein(x)
/* Get current state whether digital blinker is on */
#define HW_sb_ein(x)        s_sb_ein(x)
/* Get current state whether minimal green time elapsed */
#define HW_min_grun(x)      s_min_grun(x)
/* Get current state whether signal-group is in transition open-closed */
#define HW_vor(x)           s_vor(x)
#endif
/*
Check switching commands: special operations, these settings are valid for
both contoller and simulation
*/
/* Check if a red command is pending */
#define HW_bef_rot(x)       1
/* Check if a green command is pending */
#define HW_bef_grun(x)      1
/*
Current signal time
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
/* current red duration */
#define HW_t_rot(x)          s_t_rot(x)
/* current amber duration */
#define HW_t_gelb(x)         s_t_gelb(x)
/* current minimum red duration */
#define HW_t_min_rot(x)      s_t_min_rot(x)
/* current green duration */
#define HW_t_grun(x)         s_t_grun(x)
/* current preparation time */
#define HW_t_vor(x)          s_t_vor(x)
/* current minimum green duration */
#define HW_t_min_grun(x)     s_t_min_grun(x)
#else
/*
Controller
*/
/* current red duration                    */
#define HW_t_rot(x)          s_t_rot(x)
/* current amber duration                  */
#define HW_t_gelb(x)         s_t_gelb(x)
/* current minimum red duration */
```

```
#define HW_t_min_rot(x)      s_t_min_rot(x)
/* current green duration                */
#define HW_t_grun(x)         s_t_grun(x)
/* current preparation time              */
#define HW_t_vor(x)          s_t_vor(x)
/* current minimum green duration        */
#define HW_t_min_grun(x)     s_t_min_grun(x)
#endif
/*
Special (extraordinary) state
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
/* Get current state: signal-group is in fault mode       */
#define HW_blink(x)            0
/* Get current state: fault duration                      */
#define HW_t_blink(x)          0
/* Get current state: signal-group is dark                */
#define HW_Dunkel(x)           0
/* Get current state: signal-group is blinking            */
#define HW_Blinkend(x)         0
/* Get current state: signal-group is enabled for VS-PLUS */
#define HW_VspFreigegeben(x)   s_HW_VspFreigegeben(x)
#else
/*
Controller
*/
/* Get current state: signal-group is in fault mode       */
#define HW_blink(x)            s_stoeblink(x)
/* Get current state: signal-group is dark                */
#define HW_Dunkel(x)           0
/* Get current state: signal-group is blinking            */
#define HW_Blinkend(x)         0
/* Get current state: signal-group is enabled for VS-Plus */
#define HW_VspFreigegeben(x)   s_HW_VspFreigegeben(x)
#endif
/*
Access to shortened intergreens
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
/* Read intergreen from controller t */
#define l_zzV(x,y)           pANZE_Zwizt_V[x-1].MElement[y-1]
#else
/*
Controller
*/
/* Read intergreen from controller  */
#define l_zzV(x,y)           s_zwi_zeit(x,y)
#endif
/*
Error messages
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
```

```
*/
#define MELDUNG(degree,nr,par1,par2,par3,par4)     Meldung(nr, par1, par2,
par3, par4)
#define MELDUNGnet(degree,nr,Anr,par1,par2,par3,par4,par5) MeldungNET(nr,
Anr, par1, par2, par3, par4, par5)
#else
/*
Controller
*/
#define MELDUNG(degree,nr,par1,par2,par3,par4)                    Mel-
dung(degree, nr, par1, par2, par3, par4)
#define MELDUNGnet(degree,nr,Anr,par1,par2,par3,par4,par5) Mel-
dungNET(degree, nr, Anr, par1, par2, par3, par4, par5)
#endif
/*
System functions
*/
/*
Check for controller off and for controller fault
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
/*
The function returns 0 as long a a signal plan is being processed.
When the controller is off or during switch-on or switch-off, this function
must return 1
*/
#define SteuerungNichtAktiv()      s_SteuerungNichtAktiv()
#else
/*
Controller
*/
/*
The function returns 0 as long a a signal plan is being processed.
When the controller is off or during switch-on or switch-off, this function
must return 1
*/
#define SteuerungNichtAktiv()    s_SteuerungNichtAktiv()
#endif
/*
Hand over received telegram
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define GibTelegramm(x)            TelegrammVomGeraet(x)
#else
/*
Controller
*/
#define GibTelegramm(x)        TelegrammVomGeraet(x)
#endif
/*
Switch off VS-PLUS
*/
#ifdef _VSPLUS_SIM_
/*
```

```
Simulation
*/
#define m_Wunsch_VSPLUS(x,y)    Wunsch_VSPLUS(x,y)
#else
/*
Controller
*/
#define m_Wunsch_VSPLUS(x,y)    Wunsch_VSPLUS(x,y)
#endif
/*
Delay program switch (set by adaptive control type "Vmod")
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define Prog_Schaltung_erlaubt() m_Prog_Schaltung_erlaubt()
#else
/*
Controller
*/
#define _#_()           m_Prog_Schaltung_erlaubt
#endif

/*
OCIT FUNCTIONS
*/

/*
PT, IT on/off (preset in Controller, OCIT central-controller) and PT, IT
on-/off-state (current in VSPlus)
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define l_OePNV_Ein_Aus()       OePNV_Ein_Aus()
#define l_IV_Ein_Aus()          IV_Ein_Aus()
#define OePNV_Ein_Aus_Ist()     OePNV_Ein_Aus()
#define IV_Ein_Aus_Ist()        IV_Ein_Aus()
#else
/*
Controller
*/
#define l_OePNV_Ein_Aus()    OePNV_Ein_Aus()   /* Read PT on/off */
#define l_IV_Ein_Aus()       IV_Ein_Aus()   /* Read IT on/off */
#define _#_()  (l_VSP_X_Ein_Aus_Ist(0)    /* Read PT on/off state of VSP */
#define _#_()  (l_VSP_X_Ein_Aus_Ist(1))   /* Read IT on/off state of VSP */
#endif
/*
Detector exists
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define l_Det_Aktiv(x)          Det_Aktiv(x)
#else
```

```
/*
Controller
*/
#define l_Det_Aktiv(x)          Det_Aktiv(x)
#endif
/*
Signal-group exists
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define l_Sg_Aktiv(x)           Sg_Aktiv(x)
#else
/*
Controller
*/
#define l_Sg_Aktiv(x)           Sg_Aktiv(x)
#endif
/*
Get current time
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define l_AktuelleZeit(x,y,z)   AktuelleZeit(x,y,z)
#else
/*
Controller
*/
#define l_AktuelleZeit(x,y,z)   AktuelleZeit(x,y,z)
#endif
/*
Get current date
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define l_AktuellesDatum(w,x,y,z)  AktuellesDatum(w,x,y,z)
#else
/*
Controller
*/
#define l_AktuellesDatum(w,x,y,z)  AktuellesDatum(w,x,y,z)
#endif
/*
Get unit-number (intersection): OCIT outstationID
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define l_Get_OCITOutstationId(x,y,z)   Get_OCITOutstationId(x,y,z)
#else
/*
Controller
*/
#define l_Get_OCITOutstationId(x,y,z)   Get_OCITOutstationId(x,y,z)
```

```
#endif
/*
Get central-controller signal program selection
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define l_ProgrammWahlZentrale()         ProgrammWahlZentrale()
#else
/*
Controller
*/
#define l_ProgrammWahlZentrale()         ProgrammWahlZentrale()
#endif
/*
Central-controller special intervention
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
/*
Read special intervention Version includes job-number (vn)
*/
#define l_ZSondereingriffvn(S, E, vn)    ZSondereingriffvn(S, E, vn)
/*
Get UTC time-stamp
*/
#define l_UTCZeitstempel()               UTCZeitstempel()
/*
Write special intervention Version includes job-number (vn)
*/
#define ZSondereingriffVSP(vn)           m_ZSondereingriffVSPvn(vn)
#else
/*
Controller
*/
/*
Read special intervention
Version includes job-number (vn)
*/
#define l_ZSondereingriffvn(S, E, vn)    ZSondereingriffvn(S, E, vn)
/*
Get UTC time-stamp
*/
#define l_UTCZeitstempel()               UTCZeitstempel()
/*
Write special intervention Version includes job-number (vn)
*/
#define _#_(vn)                       m_ZSondereingriffVSPvn(vn)
#endif

/*
FUNCTIONS FOR SUPPLY PARAMETER FILES
*/

/*
Request memory
*/
```

```
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define f_Alloziere_VSP_Speicher(x,y) Allozieren_VSP_Speicher(x,y)
#else
/*
Controller
*/
#define f_Alloziere_VSP_Speicher(x,y) Allozieren_VSP_Speicher(x,y)
#endif
/*
Free VS-PLUS supply parameter memory
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define l_Freigeben_VSP_Speicher(x)   Freigeben_VSP_Speicher(x)
#else
/*
Controller
*/
#define l_Freigeben_VSP_Speicher(x)   Freigeben_VSP_Speicher(x)
#endif
/*
Request VS-PLUS supply parameter memory pointer
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define l_Gib_VSP_Zeiger(x)           Gib_VSP_Zeiger(x)
#else
/*
Controller
*/
#define l_Gib_VSP_Zeiger(x)           Gib_VSP_Zeiger(x)
#endif
/*
Read supply parameters from controller
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define l_Oeffnen_VSP_Parameter()     Oeffnen_VSP_Parameter()
#define l_Lesen_VSP_Parameter(x,y)    Read_VSP_Parameter(x,y)
#define l_Schliessen_VSP_Parameter()  Schliessen_VSP_Parameter()
#else
/*
Controller
*/
#define l_Oeffnen_VSP_Parameter()     Oeffnen_VSP_Parameter()
#define l_Lesen_VSP_Parameter(x,y)    Read_VSP_Parameter(x,y)
#define l_Schliessen_VSP_Parameter()  Schliessen_VSP_Parameter()
#endif
/*
Save supply parameters to controller
*/
```

```
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define m_Oeffnen_Sichern_Parameter()      Oeffnen_Sichern_Parameter()
#define m_Schreiben_Sichern_Parameter(x,y) Schreiben_Sichern_Parameter(x,y)
#define m_Schliessen_Sichern_Parameter()   Schliessen_Sichern_Parameter()
#else
/*
Controller
*/
#define m_Oeffnen_Sichern_Parameter()      Oeffnen_Sichern_Parameter()
#define m_Schreiben_Sichern_Parameter(x,y) Schreiben_Sichern_Parameter(x,y)
#define m_Schliessen_Sichern_Parameter()   Schliessen_Sichern_Parameter()
#endif
/*
Initialize supply process
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define Initial_VSP_Parameter()       f_Initial_VSP_Parameter()
#else
/*
Controller
*/
#define _#_()                         f_Initial_VSP_Parameter()
#endif
/*
Check supply parameter file
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define Pruefen_VSP_Parameter()       f_Pruefen_VSP_Parameter()
#else
/*
Controller
*/
#define _#_()                         f_Pruefen_VSP_Parameter()
#endif
/*
Read supply parameter file
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define Lesen_VSP_Parameter()         f_Lesen_VSP_Parameter()
#else
/*
Controller
*/
#define _#_()                         f_Lesen_VSP_Parameter()
#endif
/*
End VS-PLUS parameters - Release memory on controlled shut-down
*/
```

```
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define Ende_VSP_Parameter()          f_Ende_VSP_Parameter()
#else
/*
Controller
*/
#define _#_()                         f_Ende_VSP_Parameter()
#endif

/*
FUNCTIONS FOR VS-PLUS COMMANDS
*/

/*
New command file?
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define l_Neue_Befehle()              Neue_Befehle()
#else
/*
Controller
*/
#define l_Neue_Befehle()              Neue_Befehle()
#endif
/*
Accessing the command file
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define l_Oeffnen_VSP_Befehle()       Oeffnen_VSP_Befehle()
#define l_Lesen_VSP_Befehle(x,y)      Lesen_VSP_Befehle(x,y)
#define l_Schliessen_VSP_Befehle()    Schliessen_VSP_Befehle()
#define VABefehlPfad()                l_VABefehlPfad()
#else
/*
Controller
*/
#define l_Oeffnen_VSP_Befehle()       Oeffnen_VSP_Befehle()
#define l_Lesen_VSP_Befehle(x,y)      Lesen_VSP_Befehle(x,y)
#define l_Schliessen_VSP_Befehle()    Schliessen_VSP_Befehle()
#define _#_()                         l_VABefehlPfad()
#endif

/*
ACCESSING PROCESS DATA
*/

/*
Is there a program that is parameterized for VS-PLUS?
*/
#ifdef _VSPLUS_SIM_
/*
```

```
Simulation
*/
#define Prog_VSP(x)                    l_Prog_VSP(x)
#else
/*
Controller
*/
#define  _#_(x)                        l_Prog_VSP(x)
#endif
/*
Controller asks for VS-PLUS version text
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define f_versions_txt(x,y)            f_versions_txt(x,y)
#else
/*
Controller
*/
/*
short f_versions_txt(char* text, int size)
*/
#define _#_()                          f_versions_txt(x,y)
#endif
/*
Asking for PD values
*/
#ifdef _VSPLUS_SIM_
/*
Simulation
*/
#define f_VSP_ProzessDaten(x,y)        f_VSP_ProzessDaten(x,y)
#else
/*
Controller
*/
/*
U_WORD f_VSP_ProzessDaten(void *px, void *py)
*/
#define _#_()                          f_VSP_ProzessDaten(x,y)
#endif
/*
READ PT MEMORY
*/
#ifdef _VSPLUS_SIM_
/*
Simulation (inactive)
*/
#define l_OEVSpeicherLesenEin()        OEVSpeicherLesenEin()
#define f_OEVSpeicherAusgabe(x,y)      OEVSpeicherAusgabe(x,y)
#else
/*
Controller
*/
/*
Controller defines if reading of PT memory (AMLi telegram) is requires
0 = don't read, 1 = read
*/
```

```
#define l_OEVSpeicherLesenEin()       0
/*
AMLi telegram
void f_OEVSpeicherAusgabe(char* oev_daten, short typ)
type 1 = AMLi telegram header (single output only)
0 = AMLi telegram data
Format: NR  TT.MO HH:MM:SS  MPN  LLLKK RRR PZH FAHRP  TX SP PH-UE TWF RTE
GNE
*/
#define f_OEVSpeicherAusgabe(x,y)     0
#endif

/*
SWAP BYTES
*/

#ifdef _PROZESSOR_INTEL_
/*
Intel processor
*/
/*
Call m_BC in order to swap bytes
*/
#define m_BCS(a) (signed short) (((((unsigned short)(a))<<8)|(((unsigned
short)(a))>>8)))
#define m_BCL(a) \
(signed long) (( ((unsigned long)(a)) <<24 ) | \
( ( ( ((unsigned long)(a)) >>8  )<<24 )>>8  )| \
( ( ( ((unsigned long)(a)) >>16 )<<24 )>>16 )| \
( ((unsigned long)(a)) >> 24))
#define m_BC(a) (  (sizeof (a) == 2) ? (m_BCS (a)) : (m_BCL (a)) )
#else
/*
Not an Intel processor
*/
#define m_BC(a)             a
#endif

/*
HELPER_MACROS
*/

#ifdef _HELPER_MACROS_
#define LOBYTE(w)           ((unsigned char)(w))
#define HIBYTE(w)           ((unsigned char)((unsigned int)(w) >> 8))
#define LOWORD(l)           ((unsigned short)(l))
#define HIWORD(l)           ((unsigned short)((unsigned long)(l) >> 16))
#endif
#endif
```

APPENDIX 2: VS-PLUS OITD NUMBERS

Name	OITD2 value	OITD4 value	Symbol (German)	Data type	Description	Index range
57.0	58368	3735552	OITD_VSP_VERSION	USHORT	VS-PLUS version	1
57.1	58369	3735553	OITD_VSP_SYSTEM1	USHORT	System variable VS-PLUS (TX, TU)	1 - 2
57.2	58370	3735554	OITD_VSP_SYSTEM2	USHORT	System variable VS-PLUS (frame signal plan: current, target)	1 - 2
57.3	58371	3735555	OITD_VSP_SYSTEM3	USHORT	System variable VS-PLUS (program number: current, target)	1 - 2
57.4	58372	3735556	OITD_VSP_SYSTEM4	USHORT	System variable VS-PLUS	1
57.5	58373	3735557	OITD_VSP_SYSTEM5	USHORT	System variable VS-PLUS	1
57.6	58374	3735558	OITD_VSP_SYSTEM6	USHORT	System variable VS-PLUS	1
57.7	58375	3735559	OITD_VSP_SYSTEM7	USHORT	System variable VS-PLUS	1
57.8	58376	3735560	OITD_VSP_SYSTEM8	USHORT	System variable VS-PLUS	1
57.9	58377	3735561	OITD_VSP_SYSTEM9	USHORT	System variable VS-PLUS	1
57.10	58378	3735562	OITD_VSP_ADAPTIV1	UBYTE	System variable for adaptive control	1
57.11	58379	3735563	OITD_VSP_ADAPTIV2	UBYTE	System variable for adaptive control (MTS current)	1 - 64
57.12	58380	3735564	OITD_VSP_ADAPTIV3	USHORT	System variable for adaptive control (MTSs tate)	1
57.13	58381	3735565	OITD_VSP_ADAPTIV4	USHORT	System variable for adaptive control	1
57.14	58382	3735566	OITD_VSP_ADAPTIV5	USHORT	System variable for adaptive control	1
57.15	58383	3735567	OITD_VSP_ADAPTIV6	USHORT	System variable for adaptive control	1
57.16	58384	3735568	OITD_VSP_ADAPTIV7	USHORT	System variable for adaptive control	1
57.17	58385	3735569	OITD_VSP_ADAPTIV8	USHORT	System variable for adaptive control	1
57.18	58386	3735570	OITD_VSP_ADAPTIV9	USHORT	System variable for adaptive control	1
57.19	58387	3735571	OITD_VSP_ADAPTIV10	USHORT	System variable for adaptive control	1
57.101	58469	3735653	OITD_DET_WARTEZEIT	USHORT	Detector waiting time (Det_ID)	1 - 280
57.102	58470	3735654	OITD_VS_ZUSTAND	USHORT	Current traffic stream state	1 - 64
57.103	58471	3735655	OITD_VS_WARTEZEIT	USHORT	Traffic stream waiting time	1 - 64
57.104	58472	3735656	OITD_VS_STUFE	UBYTE	Current traffic stream priority level	1 - 64
57.105	58473	3735657	OITD_VS_KONTROLLZEIT	UBYTE	Traffic stream control time	1 - 64
57.106	58474	3735658	OITD_VS_G_MAX	USHORT	Maximum reachable green time	1 - 64
57.107	58475	3735659	OITD_VS_PRIOKLASSE	UBYTE	Current traffic stream priority class	1 - 64
57.108	58476	3735660	OITD_VS_PF_WERT	USHORT	Traffic stream priority value	1 - 64
57.109	58477	3735661	OITD_VS_ANFO_GUELTIG	UBYTE	Is the traffic stream call valid?	1 - 64

Name	OITD2 value	OITD4 value	Symbol (German)	Data type	Description	Index range
57.110	58478	3735662	OITD_VS_ANFO_TYP	UBYTE	Traffic stream call type	1 - 64
57.111	58479	3735663	OITD_VS_WUNSCH	UBYTE	What zone the traffic stream is in?	1 - 64
57.112	58480	3735664	OITD_VS_RAHMENSIGNAL	UBYTE	Actual traffic stream frame signal	1 - 64
57.113	58481	3735665	OITD_VS_STATUS	UBYTE	Actual traffic stream status	1 - 64
57.114	58482	3735666	OITD_VS_DT_RAHMEN	USHORT	Actual time difference until next frame	1 - 64
57.115	58483	3735667	OITD_VS_RESTFAHRZEIT	SHORT	Remaining travel time for PT traffic stream	1 - 64
57.116	58484	3735668	OITD_VS_OEV_PRIO	UBYTE	PT traffic stream priority	1 - 64
57.117	58485	3735669	OITD_VS_GRUEN_WEGEN	UBYTE	Traffic stream has obtained green as…	1 - 64
57.118	58486	3735670	OITD_VS_HAT_STAU	UBYTE	Traffic stream is in spillback mode	1 - 64
57.119	58487	3735671	OITD_VS_ZZ_KURZ	UBYTE	Traffic streams uses shortened intergreens	1 - 64
57.120	58488	3735672	OITD_VS_VERRIGELUNG	UBYTE	Traffic stream is locked	1 - 64
57.121	58489	3735673	OITD_VS_OEV_RANG	UBYTE	Rank sequence for PT traffic streams	1 - 10
57.122	58490	3735674	OITD_VS_ANKUNFT_EW	USHORT	Expected arrival second of a PT traffic stream	1 - 64
57.123	58491	3735675	OITD_VS_ROT_GRUN	USHORT	Actual green / red time of a traffic stream	1 - 64
57.124	58492	3735676	OITD_VS_WSUMM	UBYTE	Repetition sum of a traffic stream	1 - 64
57.125	58493	3735677	OITD_HAUPTZEIGER	UBYTE	Priority element main pointer	1 - 6
57.126	58494	3735678	OITD_WUNSCHBILD	UBYTE	Priority element target image	1 - 10
57.127	58495	3735679	OITD_TK_AKTIVFLAG	UBYTE	Partial node active flag	1 - 3
57.128	58496	3735680	OITD_OEV_WARTEZEIT_VS	USHORT	Waiting time of a PT traffic stream	1 - 64
57.130	58498	3735682	OITD_OEV_LINIE_VS	USHORT	Line id of a PT traffic stream	1 - 64
57.131	58499	3735683	OITD_OEV_ROUTE_VS	USHORT	Route id of a PT traffic stream	1 - 64
57.132	58500	3735684	OITD_OEV_KURS_VS	USHORT	Course id of a PT traffic stream	1 - 64
57.133	58501	3735685	OITD_OEV_FOLGEZUEGE_VS	UBYTE	Follower train flag of a PT traffic stream	1 - 64
57.134	58502	3735686	OITD_OEV_ABMELDEZEIT_VS	USHORT	Check-out time of a PT traffic stream	1 - 64
57.135	58503	3735687	OITD_OEV_NOTAN_WARTEZEIT_VS	USHORT	Emergence check-in waiting time of a PT traffic stream	1 - 64
57.136	58504	3735688	OITD_OEV_PRIORITAET_VS	UBYTE	Priority of a PT traffic stream	1 - 64
57.137	58505	3735689	OITD_OEV_ZWANGSABM_VS	UBYTE	Forced check-out time of a PT traffic stream	1 - 64
57.138	58506	3735690	OITD_DET_S_IMP_SUMME	UBYTE	Sum of rising slopes	1 - 120
57.139	58507	3735691	OITD_DET_F_IMP_SUMME	UBYTE	Sum of falling slopes	1 - 120
57.140	58508	3735692	OITD_DET_BELGRAD	UBYTE	Actual occupancy degree	1 - 120

Name	OITD2 value	OITD4 value	Symbol (German)	Data type	Description	Index range
57.141	58509	3735693	OITD_DET_BELGRAD_GEGLAETTET	UBYTE	Smoothened occupancy degree	1 - 120
57.142	58510	3735694	OITD_DET_AKT_BELZEIT	USHORT	Actual occupancy time	1 - 120
57.143	58511	3735695	OITD_DET_AKT_BELZUST	UBYTE	Actual occupancy state	1 - 120
57.144	58512	3735696	OITD_DET_LUECKE	USHORT	Time since last falling slope	1 - 120
57.145	58513	3735697	OITD_DET_BRUTTO_LUECKE	USHORT	Time since last rising slope	1 - 120
57.146	58514	3735698	OITD_DET_STOERUNG	UBYTE	Actual error state	1 - 120

Table 112: VS-PLUS OITD numbers

APPENDIX 3: VS-PLUS VERSIONS

		VS-PLUS import version												
		6.0.0	6.1.0	6.1.2	6.1.3	6.1.4	6.2.0	6.2.1	6.2.2	6.2.3	6.2.4	6.2.5	6.2.6	7.0.0
VS-PLUS version	6.0.0	X												
	6.1.0		X											
	6.1.2			X										
	6.1.3		X	X	X									
	6.1.4				X	X								
	6.2.0						X							
	6.2.1						X	X						
	6.2.2						X	X	X					
	6.2.3						X	X	X	X				
	6.2.4						X	X	X	X	X			
	6.2.5						X	X	X	X	X	X		
	6.2.6						X	X	X	X	X	X	X	
	7.0.0											X	X	X

Table 113: VS-PLUS versions