DBonDP

Standard-Software for Simatic S7

Inhaltsverzeichnis

The actual list of products can be found on http://sites.inka.de/heisch

Version 04.05.2009 HW

Heisch Automation

Ingenieurbüro für Industrieautomatisierung

Büro:	Ostring 15	D 76829 Landau / Pfalz
Postadresse:	Im Vorderen Großthal 4	D 76857 Albersweiler /Pfalz
Tel:	+49 6341 890117	Fax: +49 6341 890118
e-mail:	hwauto@heisch-automation.de	www.heisch-automation.de

1.Scope of delivery

The library DBonDP contains of

1. this **manual**

2. the FB 120 „DBonDP“

3. Examples and demonstration programs

containing of:

- OB 1 main program for the examples
- FC 9 Generating time pulses and Log0,Log1
- FC 120 example: Simulation of the local and the remote plc, German documentation
- FC 120 example: Simulation of the local and the remote plc, English documentation
- FB 121 Simulation-Version of FB120

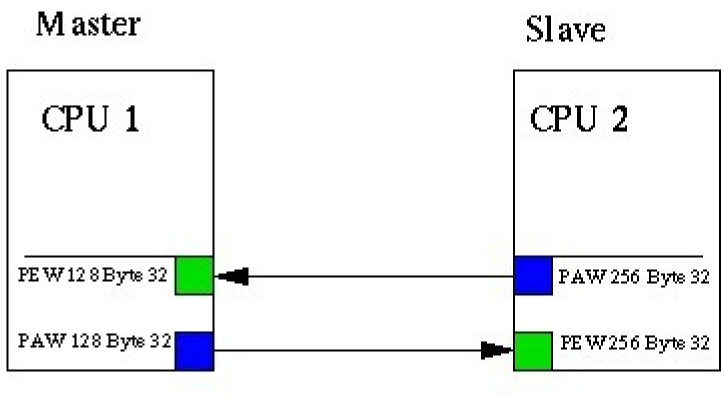
- DB120 Instance-DB for FB 121 (local plc)
- DB121 Instance-DB for FB 121 (remote plc)
- DB1120 source DB local plc
- DB1121 destination DB in the remote plc

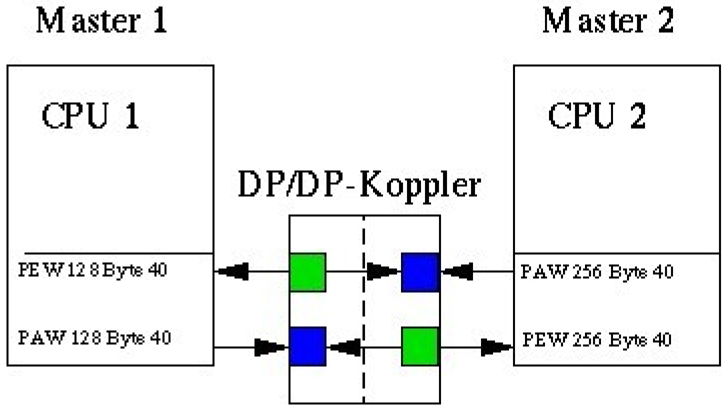
The demo version does not contain the FB 120. It is for sale.

2. Functionality

Bidirektional transfer of data blocks between 2 Simatic-S7-CPUs over IO-link.

Qualified for the data exchange between machines which are dedicated for block wise IO but cannot exchange telegrams.





Generally these are:

- CPUs with Profibus interface, which are in the ssame net and may be coupled by I/O in master slave mode.
  (one CPU is Master, 2. CPU is Slave)
- CPUs, which are linked over a DP/DP-coupler.
- Any I/O link, which contains the following conditions.

Conditions:

- Data consistence possible on I/O, block wise adjustable. (*)
- minimal block size 8 Bytes (*)
- Couple area: PIW / POW of each machine need to be adjustable to the same numbers (i.e: PIW 256, POW 256) (*)
- The peripherial addresses of the both machines may be different. (*)

(*) All S7 CPUs with Profibus-DP interface comply with these conditions.

Remark:

The core of this link method, the FB120, was created in 2001 for a client's project.
In this project, data are transfered between 8 CPUs, each FB-FB-link transfers different DBs by multiplexing the parameters.

No problems are known. Says: This link mode is proved by practice.

Later changes for this library:

- Profibus I/O block size was made flexible from 8 up to 224 Byte.
- Error messages improved.

The demo programs have been greated for this for-sale-library. The FB 121 was created for pre installation tests and for the following demo programs.

3. Implementation and operation

Operation:
FB 120 transfers data blocks to a remote partner by using decentral periphery, situated in a DP/DP-coupler or in a Profibus slave system.
The FB is working full duplex, says: send and receive jobs are processed simultaneously.

Implemented: Write-Aktiv, Receive-Passiv. Fetch-jobs are not implemented.

Data consistency of the transferred DBs: ("IO_Length_Word" -1) words.

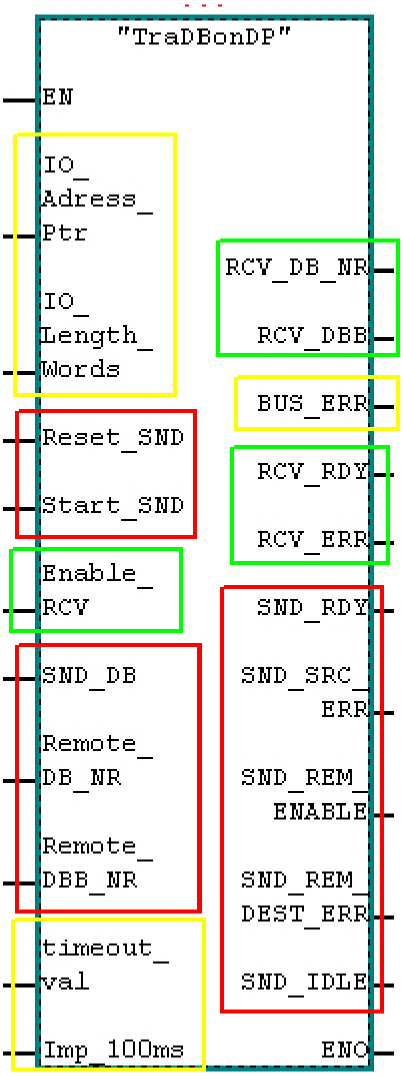
The FB will be used in local and remote machine, it is working symmetrically.

Conditions
Used I/O block need to be consistent for mimimum of 4 words (8 bytes).
Couple area: PIW / POW of each machine need to be adjustable to the same numbers
(i.e: PIW 256, POW 256)

The yellow inputs define the link andthe timeout detection. They are mandatory.
The yellow output „BUS_ERR“ is useful.

The red inputs and outputs belong to the sending unit.

The green inputs and outputs belong to the receive unit.



The Input parameters:

Definition of the I/O area:

IO_Adress_Ptr i.e.. PIW256 Start of the I/O area

IO_Length_Words Length of the I/O area in words.
Allowed: 4 .. 112

Commands to send

Reset_SND Reset send job
Normally, this input is unused. (default = FALSE). If a necessity for a reset is detected by the transfer program, it is processed automaticly.
This input may be used, if the application decides, that is necessary.

Start_SND start send command :
For sending one time, a pulse is sufficient, as long as the output SND_IDLE is in TRUE state.
The send command may be set constant to TRUE, then sending of the parametrized SND_DB will be processed periodicly.

SND_DB ANY-Pointer to the DB area in the local machine, which hast to be sended.
The length of the telegram is deduced from the lenth in this pounter.

Remote_DB_NR Number of the destination DB in the remote machine

Remote_DBB_NR Number of the destination DBB in the remote machine

Command to receive

Enable_RCV Enable receive data
this signal is transfered to the partner machine and will be shown at the output parameter "SND_REM_ENABLE".
Because the first telegram from sender only contains the telegram header (destintion parameters), this input can control the write access to data blocks from remote.

Timeout detection

timeout_val value for Timeout (100ms units)

Imp_100ms time pulse for Timeout detection (100ms Tact)
(no Blink tact, a pulse is necessary.)

The output parameter:

General

BUS_ERR communikations error, for details: see bits in the Instance-DB, DIW 34

Receive

RCV_DB_NR Number of the actually received DB

RCV_DBB Number of the DBB start position of the actually received DB
"RCV_DB_NR" and "RCV_DBB" report the values which have been written to the parameters Remote_DB_NR and Remote_DBB_NR by the sending partner.
Both values are set to 0 if no receive is in action.

RCV_RDY receive state: ready
This signal is a pulse which is TRUE if the receive jobe was regulary done.

RCV_ERR receive state: error on receive job. Destination data block may contain crippled data.

Send

SND_RDY Send State : ready.
This bit is set, if the remote system has signalized a correctly finished transfer. (See partner: RCV_RDY)
This bit is reset, if an new sending job was started or if an error occured.

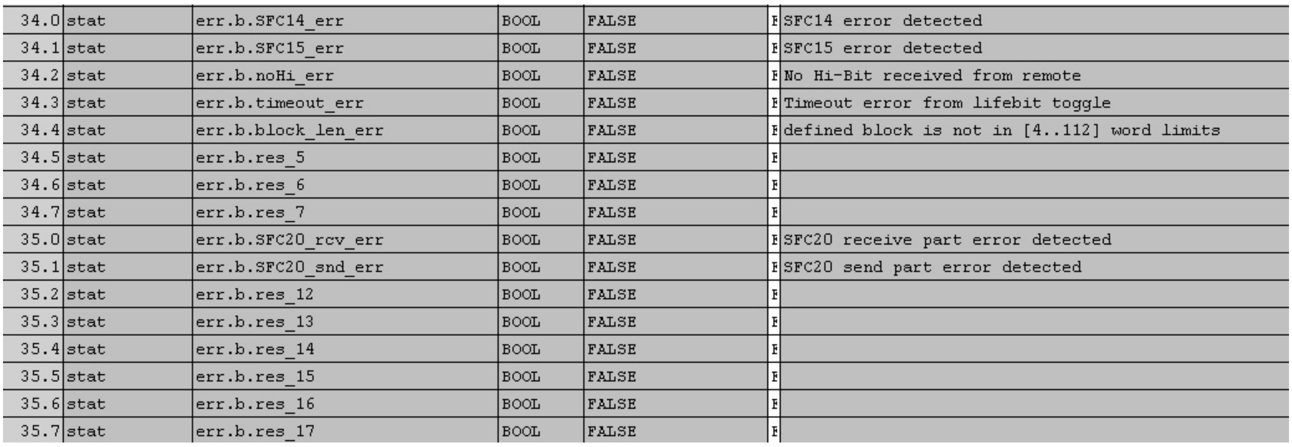
SND_SRC_ERR Sending error: The source DB of the local machine cannot be read.
Does the DB exist? How about the length ?

SND_REM_ENABLE Send State : The partner has enabled the receiving

SND_REM_DEST_ERR Send State : error message from the partner:
The destination DB may not exist or may be too short.

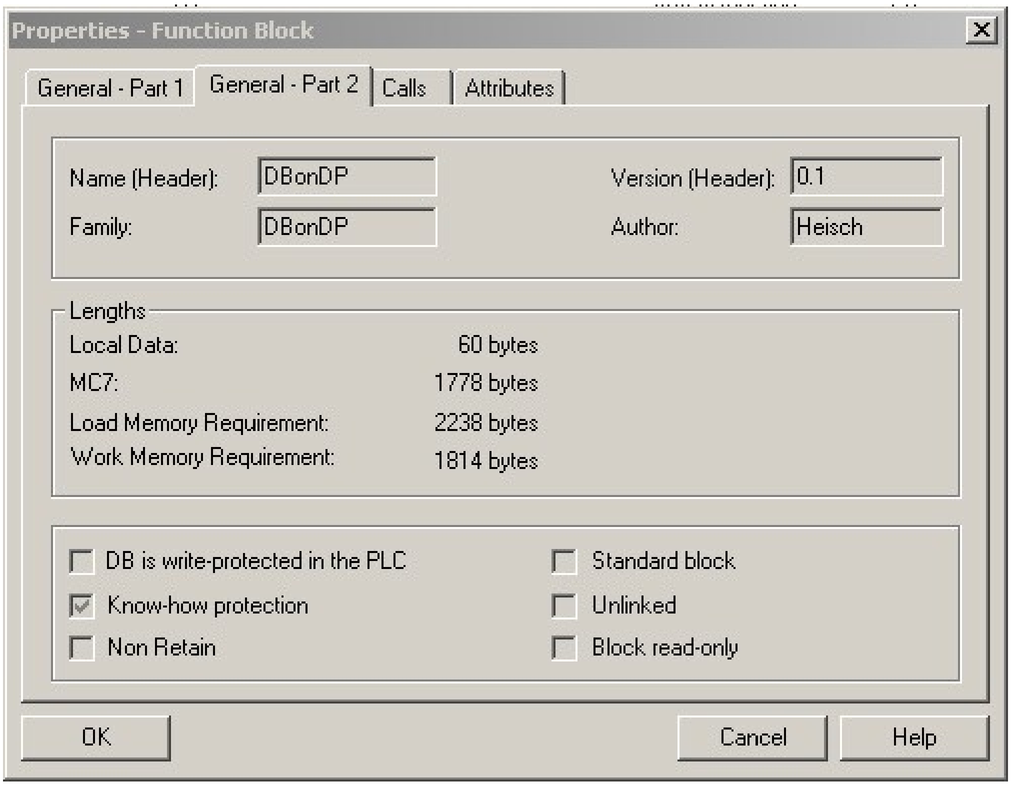
SND_IDLE Send State : sending part is idle, ready for next job.

Further informations:
error bits in DIW 34 of the Instance DB.



34.0	stat	err.b.SFC14_err	BOOL	FALSE	F	SFC14 error detected
34.1	stat	err.b.SFC15_err	BOOL	FALSE	F	SFC15 error detected
34.2	stat	err.b.noHi_err	BOOL	FALSE	F	No Hi-Bit received from remote
34.3	stat	err.b.timeout_err	BOOL	FALSE	F	Timeout error from lifebit toggle
34.4	stat	err.b.block_len_err	BOOL	FALSE	F	defined block is not in [4..112] word limits
34.5	stat	err.b.res_5	BOOL	FALSE	F	
34.6	stat	err.b.res_6	BOOL	FALSE	F	
34.7	stat	err.b.res_7	BOOL	FALSE	F	
35.0	stat	err.b.SFC20_rcv_err	BOOL	FALSE	F	SFC20 receive part error detected
35.1	stat	err.b.SFC20_snd_err	BOOL	FALSE	F	SFC20 send part error detected
35.2	stat	err.b.res_12	BOOL	FALSE	F	
35.3	stat	err.b.res_13	BOOL	FALSE	F	
35.4	stat	err.b.res_14	BOOL	FALSE	F	
35.5	stat	err.b.res_15	BOOL	FALSE	F	
35.6	stat	err.b.res_16	BOOL	FALSE	F	
35.7	stat	err.b.res_17	BOOL	FALSE	F	

Ressources:



Properties - Function Block

General - Part 1 | General - Part 2 | Calls | Attributes

Name (Header): DBonDP Version (Header): 0.1

Family: DBonDP Author: Heisch

Lengths

Local Data:	60 bytes
MC7:	1778 bytes
Load Memory Requirement:	2238 bytes
Work Memory Requirement:	1814 bytes

□ DB is write-protected in the PLC □ Standard block

☑ Know-how protection □ Unlinked

□ Non Retain □ Block read-only

Transfer speed:
The FB needs at last 3 cycles for one transfer.
The first cycle transports the header (destination address) to the partner plc, the next cycles transport the data and the last cycle is the acknowledge cycle to return „SND_RDY“.

The count of cycles depends from the telegram length and the size of the I/O area (see parameter IO_Length_Words).
The first word in the I/O area allways contains a control word, the rest is used by the raw data.
Therefore the transfered raw data are (IO_Length_Words – 1) * 2 bytes per cycle.

For example: Communication between tho CPU315-DP
The block size is adjusted to 32 word, this is the biggest consistent I/O block for such a CPU.

1kByte shall be transferred.

The count of cycles for the data will be calculated:

$$\text{Cdat} = \frac{\text{telegram length [Byte]}}{(\text{IO\_Length\_Word} - 1) * 2} = \frac{1024}{(32 - 1) * 2} = \frac{1024}{62} = 16{,}52 \rightarrow 17$$

The count of all cycles : 2 + Cdat = 19

The dominating element for the transfer is the longest OB1 cycle of the two communication partners plus the bus cycle. (get not confused about something like: and half of the cycle of the partner plc. We do not use the process immage, we are writing to periphery !)

Guess with known values:
20ms is shurely a common value for a OB1 cycle of such a CPU, also a common bus cycle for a profibus DP may be 2 to 3 ms. We assume 25 ms together.

Time for one telegram: 19 cycles * 25 ms = 475ms.
This is about 2,1 kByte/s or about 17k bps.

Because the FB 120 contains a send unit and a receive unit which are working independently, sending and receiving at the same time has no negative influence to the transfer speed.

Creating data integrity:

While designing the FB120, it was not intended to implement full data integrity, because therfore, a input buffer and a output buffer of the maximum size of a data block would be needed.
This would be a handicap for the use in small CPUs.
But:
If the input data are not changed while sending, at the time while the „RCV_REDY“ output pulse is TRUE, we have data integrity.

Therefore, full data integrity may be implemented very easy:

Send side:
Use a DB as sende buffer. Write the destination address and telegram length (raw data) into this send buffer as a header. Copy the source data (raw data) behind this header.
The length of the actually sent telegram can be limited to header + raw data.
Send this complete telegram to the partner.

Receive side:
Implement also a receive buffer.
If „RCV_RDY“ is TRUE, receive was processed complete.
This singal can trigger the further execution.

Evaluate the header and copy the raw data to the destination found in to header. (i.e. Using SFC20)

If necessary, even a couple flag can be generated from this header:

```
L        „receive_buffer“.“header.DB-Nr
L        234   // If DB 234
 ==I
U        „RCV_RDY“
=        „couple flag“  // This flag is TRUE only for one cycle because of „RCV_RDY“.
```

4. Installation and usage of the library

- Copy the packed library to your programming device.
- Unpack the liblrary into yout library direction (Use STEP 7 to unpack!)
  (i.e: C:\Programme\Siemens\Step7\S7libs\ ).
- Generate a new project for your test machine.
- Copy the contents of the „DBonDP“ library to your project.

!!! Hint:

The also shipped FC 9 is part of all of our projects.
It expects, that the MB 1 is the tact flag byte of the CPU.
(HardwareConfig->CPU->Properties->Cycle/Clock memory there: enable clock memory, set it to memory byte 1)

5. Copyrights

DBonDP and all containing programs are copyright of Heisch Automation.
The manual, the FB „DBonDP“ and the example programs are copyright protected.
All rights reserved, including copiing, translation, mikroverfilming and processing by electronic systems.

Heisch Automation grants the rights to the buyers of DBonDP:
copiing „DBonDP“ and the containing demo programs and to use into own projects,
as long as the copy right label stays unchanged.

SIMATIC, S 7, Step7 are trademarks of SIEMENS AG.

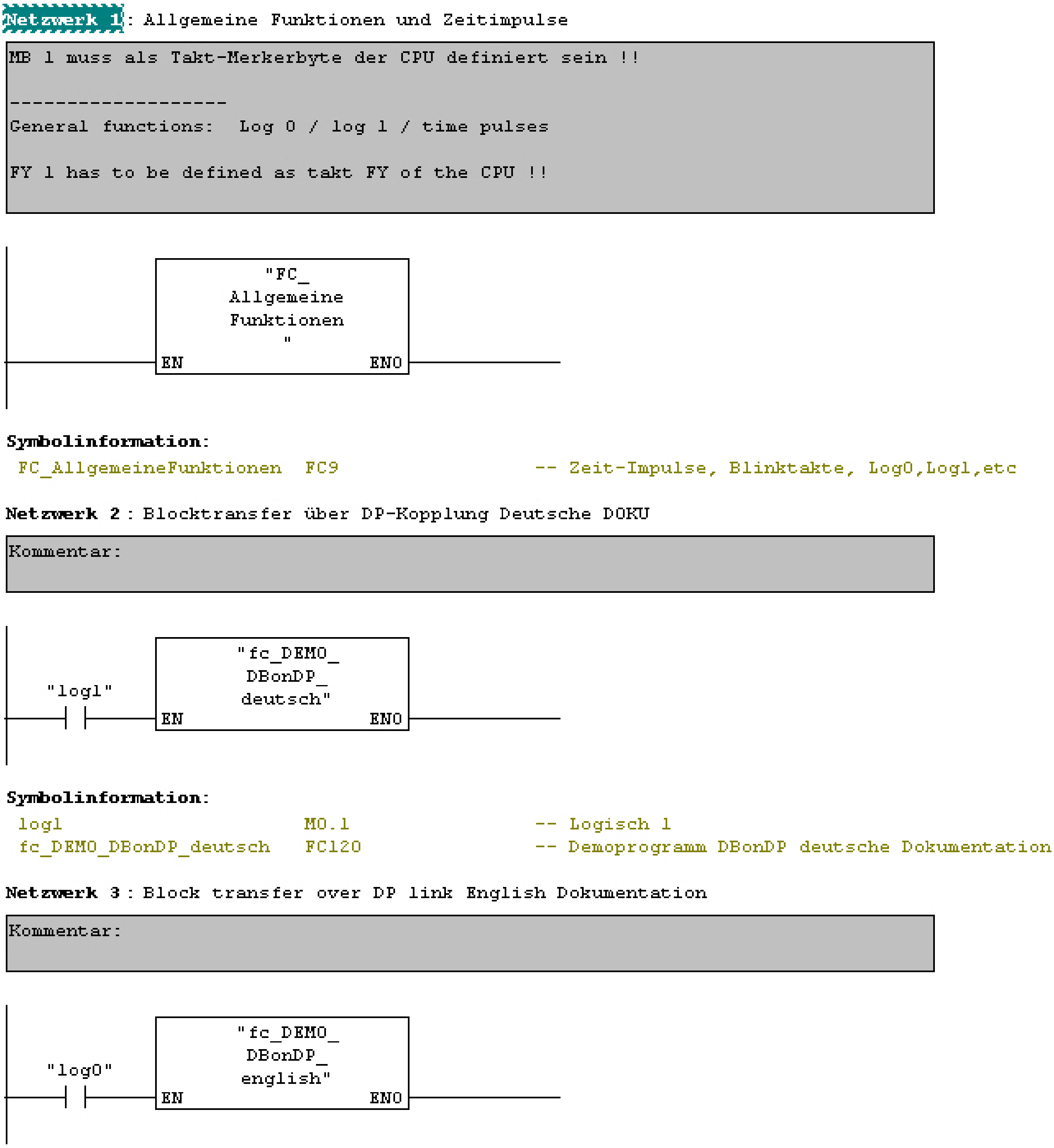
SFC 14, SFC15 and SFC20 are copyright of SIEMENS AG.
Using into this library, we see no violation of copyrights, because:

1. the shipped SFCs are not executable,they are only headers and only executable in CPUs.
2. The SFCs are parts of the CPU, everybody, who writes programs for a S7-CPU, is allowed to use it.
3. everybody, who ownes STEP 7, owns these headers in his library.
4. Only buyers who correspond to 2. or 3. , can use the „DBonDP“ library.

5. Description of the example program

In the example program FB121 is used instead of FB120.
Id does not communicate with periphery, it communicates with a DB which simulated the periphery.This enables the test of both communications partner in only one machine.

OB 1



Netzwerk 1: Allgemeine Funktionen und Zeitimpulse

```
MB 1 muss als Takt-Merkerbyte der CPU definiert sein !!

--------------------
General functions:  Log 0 / log 1 / time pulses

FY 1 has to be defined as takt FY of the CPU !!
```

Symbolinformation:

```
FC_AllgemeineFunktionen  FC9                -- Zeit-Impulse, Blinktakte, Log0,Log1,etc
```

Netzwerk 2 : Blocktransfer über DP-Kopplung Deutsche DOKU

```
Kommentar:
```

Symbolinformation:

```
log1                   M0.1                 -- Logisch 1
fc_DEMO_DBonDP_deutsch  FC120                -- Demoprogramm DBonDP deutsche Dokumentation
```

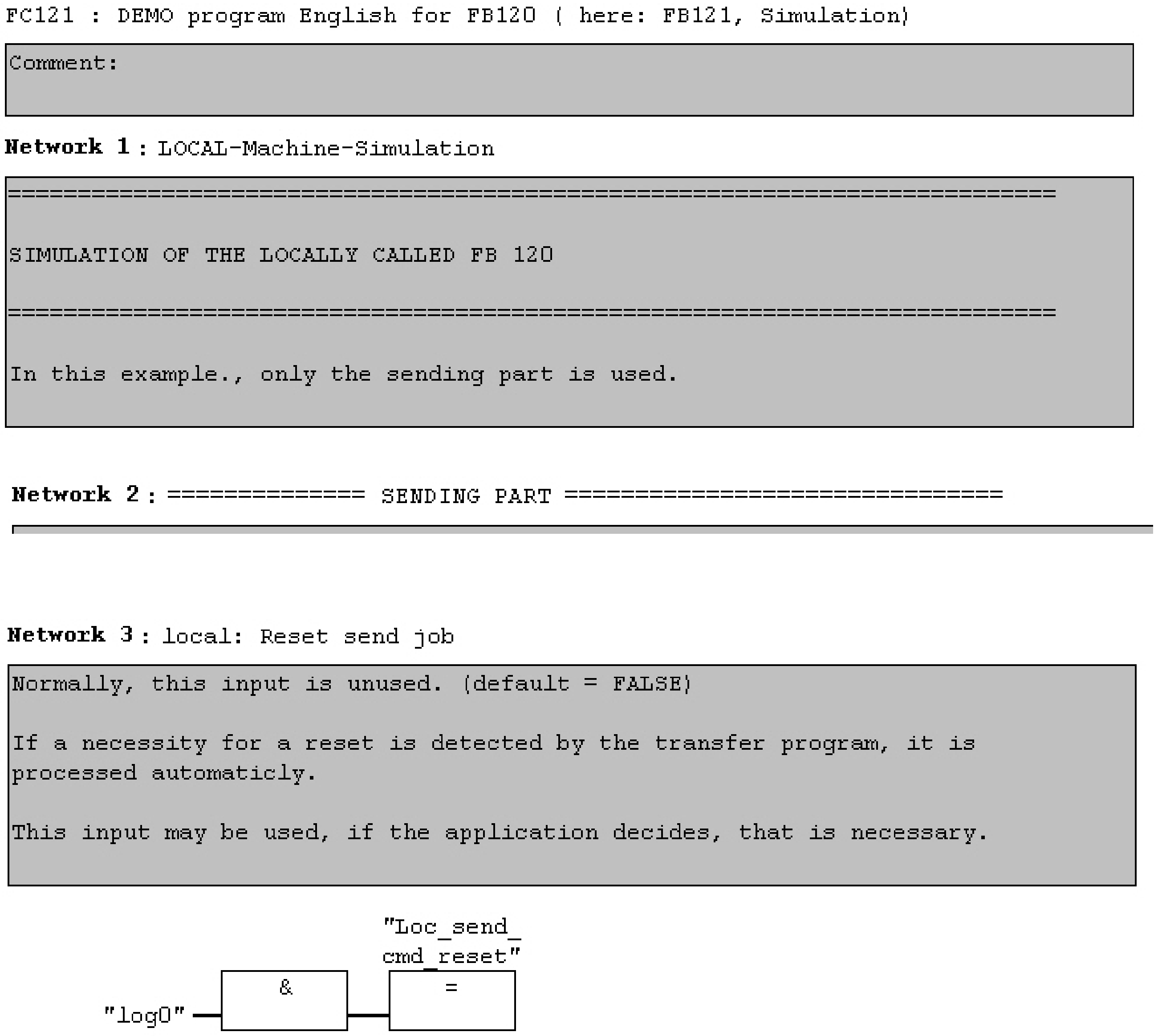
Netzwerk 3 : Block transfer over DP link English Dokumentation

```
Kommentar:
```

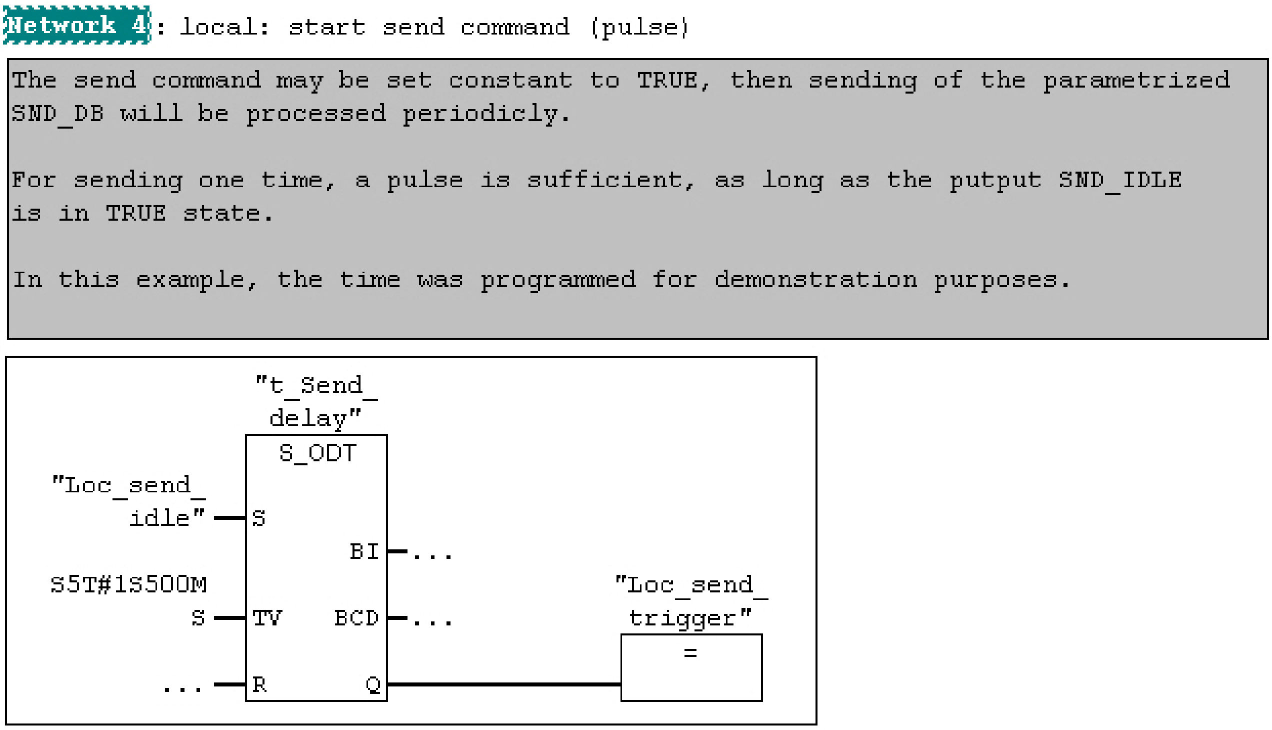
OB1 contains FC9 which contains the generation of Log0, Log1 and pulse generation.
The pulse generation is used by the timeout detection of FB 120 / FB 121.

FC 120 Simulation of the local CPU and the remote CPU

To simplificate the example, only a one way transfer is implemented.
Of course, FB120 (here: FB121) also works in both directions.



```
FC121 : DEMO program English for FB120 ( here: FB121, Simulation)

Comment:

Network 1 : LOCAL-Machine-Simulation

============================================================================

SIMULATION OF THE LOCALLY CALLED FB 120

============================================================================

In this example., only the sending part is used.


Network 2 : ============== SENDING PART ================================


Network 3 : local: Reset send job

Normally, this input is unused. (default = FALSE)

If a necessity for a reset is detected by the transfer program, it is
processed automaticly.

This input may be used, if the application decides, that is necessary.
```

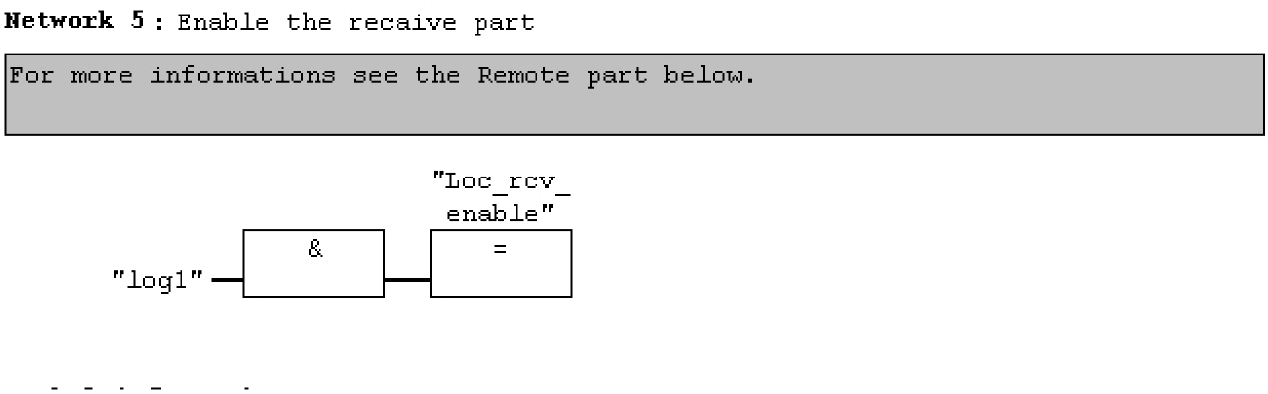


Network 4: local: start send command (pulse)

The send command may be set constant to TRUE, then sending of the parametrized SND_DB will be processed periodicly.

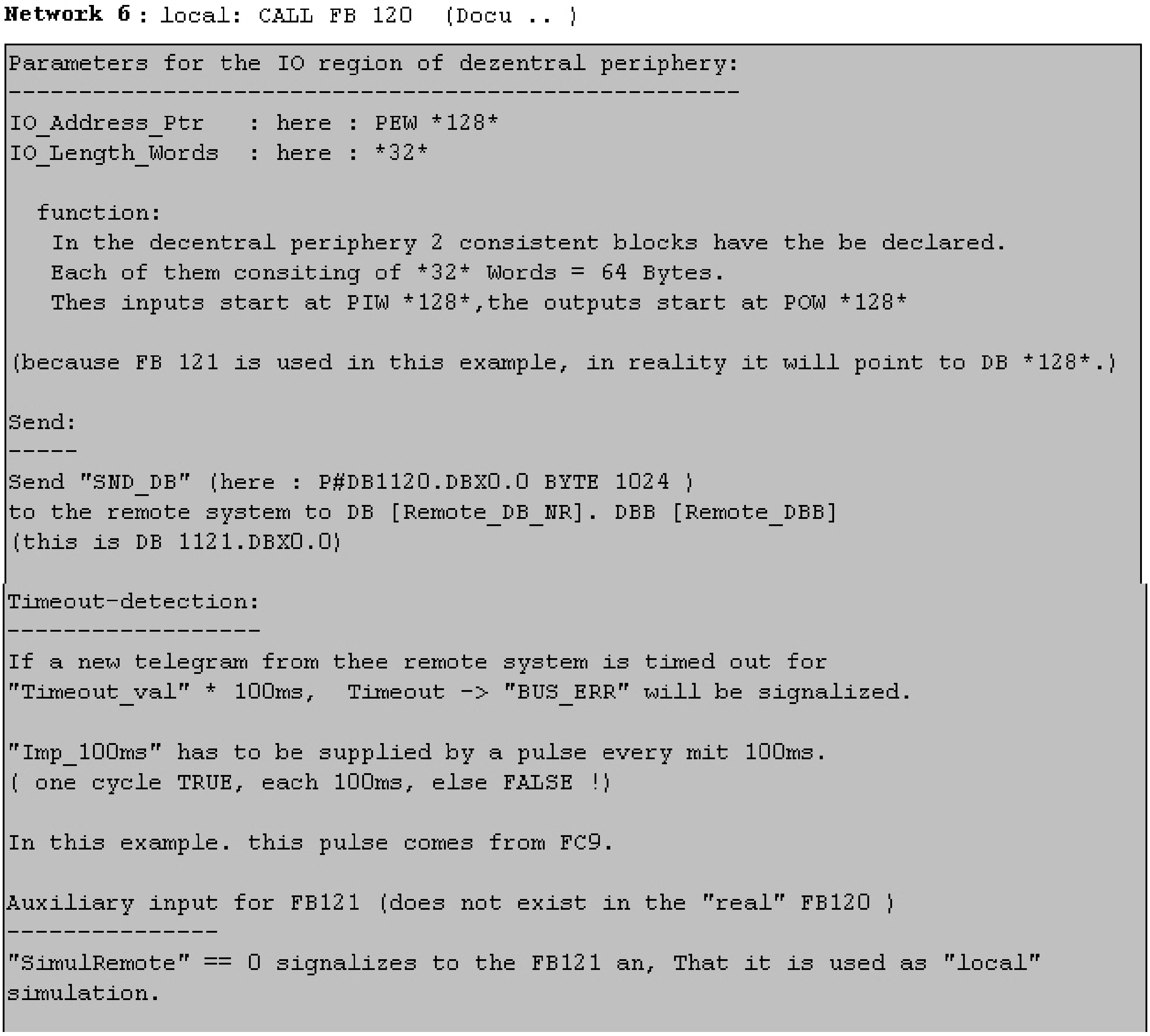
For sending one time, a pulse is sufficient, as long as the putput SND_IDLE is in TRUE state.

In this example, the time was programmed for demonstration purposes.



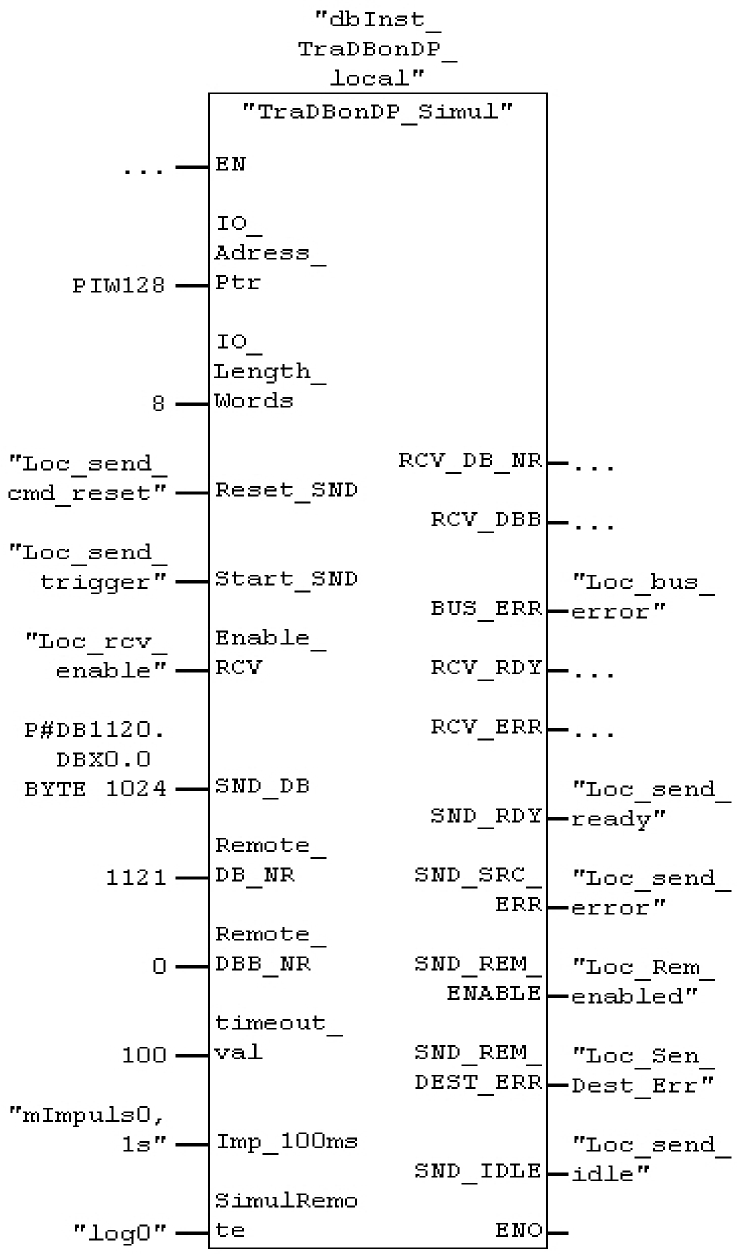
Network 5 : Enable the recaive part

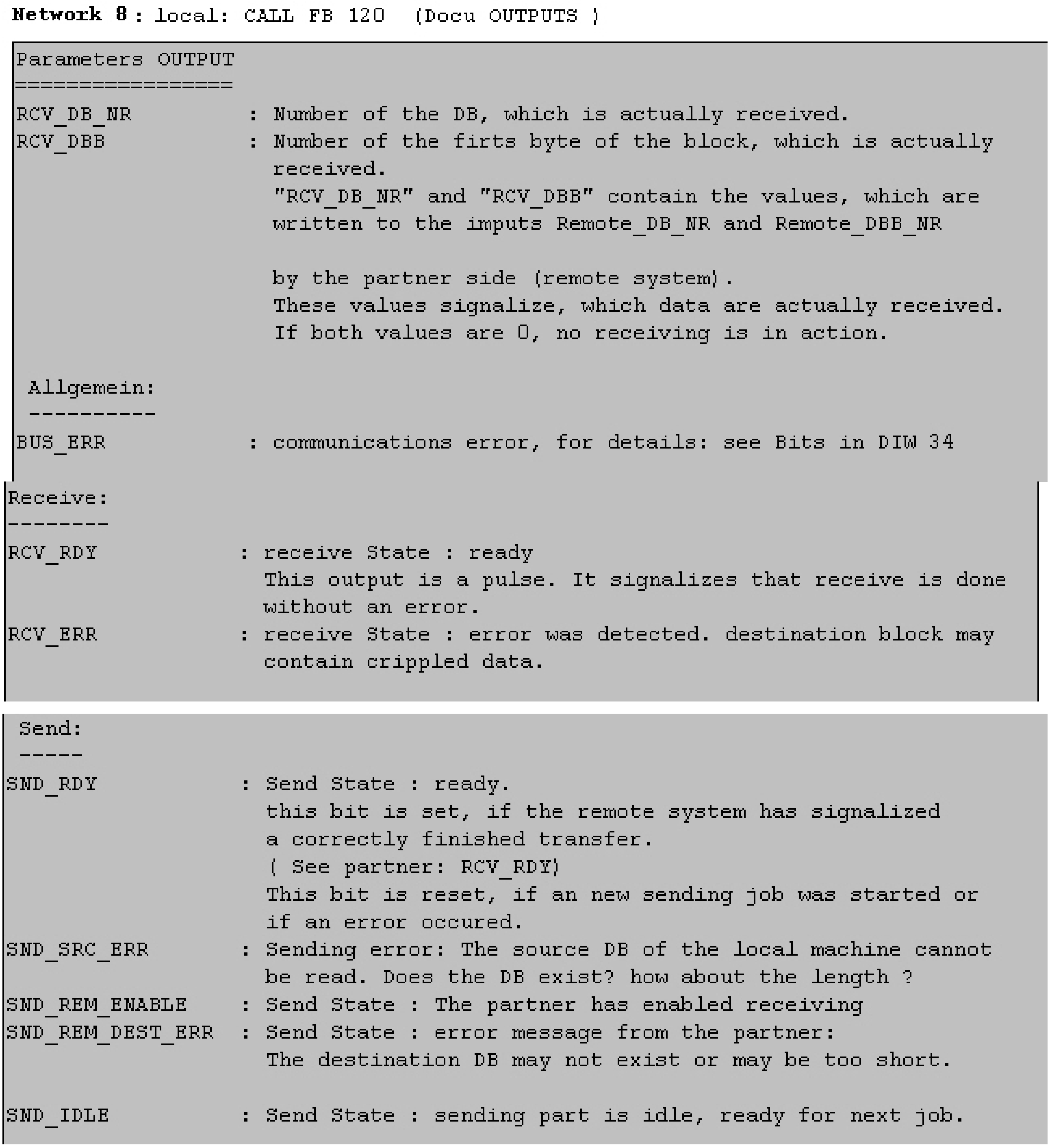
For more informations see the Remote part below.



Network 6 : local: CALL FB 120 (Docu ..)

```
Parameters for the IO region of dezentral periphery:
----------------------------------------------------
IO_Address_Ptr   : here : PEW *128*
IO_Length_Words  : here : *32*

  function:
   In the decentral periphery 2 consistent blocks have the be declared.
   Each of them consiting of *32* Words = 64 Bytes.
   Thes inputs start at PIW *128*,the outputs start at POW *128*

(because FB 121 is used in this example, in reality it will point to DB *128*.)

Send:
-----
Send "SND_DB" (here : P#DB1120.DBX0.0 BYTE 1024 )
to the remote system to DB [Remote_DB_NR]. DBB [Remote_DBB]
(this is DB 1121.DBX0.0)

Timeout-detection:
------------------
If a new telegram from thee remote system is timed out for
"Timeout_val" * 100ms,  Timeout -> "BUS_ERR" will be signalized.

"Imp_100ms" has to be supplied by a pulse every mit 100ms.
( one cycle TRUE, each 100ms, else FALSE !)

In this example. this pulse comes from FC9.

Auxiliary input for FB121 (does not exist in the "real" FB120 )
---------------
"SimulRemote" == 0 signalizes to the FB121 an, That it is used as "local"
simulation.
```

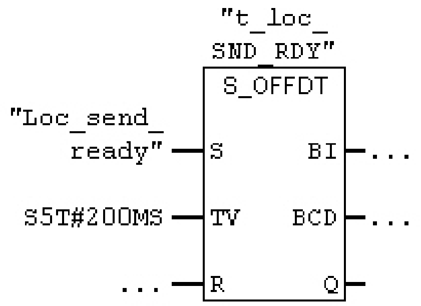

"dbInst_
TraDBonDP_
local"
"TraDBonDP_Simul"
...
EN
IO_
Adress_
Ptr
PIW128
IO_
Length_
Words
8
"Loc_send_
cmd_reset"
Reset_SND
"Loc_send_
trigger"
Start_SND
"Loc_rcv_
enable"
Enable_
RCV
P#DB1120.
DBX0.0
BYTE 1024
SND_DB
Remote_
DB_NR
1121
Remote_
DBB_NR
0
timeout_
val
100
"mImpuls0,
1s"
Imp_100ms
SimulRemo
te
"log0"
RCV_DB_NR
...
RCV_DBB
...
BUS_ERR
"Loc_bus_
error"
RCV_RDY
...
RCV_ERR
...
SND_RDY
"Loc_send_
ready"
SND_SRC_
ERR
"Loc_send_
error"
SND_REM_
ENABLE
"Loc_Rem_
enabled"
SND_REM_
DEST_ERR
"Loc_Sen_
Dest_Err"
SND_IDLE
"Loc_send_
idle"
ENO



```
Network 8 : local: CALL FB 120  (Docu OUTPUTS )

Parameters OUTPUT
=================
RCV_DB_NR          : Number of the DB, which is actually received.
RCV_DBB            : Number of the firts byte of the block, which is actually
                     received.
                     "RCV_DB_NR" and "RCV_DBB" contain the values, which are
                     written to the imputs Remote_DB_NR and Remote_DBB_NR

                     by the partner side (remote system).
                     These values signalize, which data are actually received.
                     If both values are 0, no receiving is in action.

 Allgemein:
 ----------
BUS_ERR            : communications error, for details: see Bits in DIW 34

Receive:
--------
RCV_RDY            : receive State : ready
                     This output is a pulse. It signalizes that receive is done
                     without an error.
RCV_ERR            : receive State : error was detected. destination block may
                     contain crippled data.

 Send:
 -----
SND_RDY            : Send State : ready.
                     this bit is set, if the remote system has signalized
                     a correctly finished transfer.
                     ( See partner: RCV_RDY)
                     This bit is reset, if an new sending job was started or
                     if an error occured.
SND_SRC_ERR        : Sending error: The source DB of the local machine cannot
                     be read. Does the DB exist? how about the length ?
SND_REM_ENABLE     : Send State : The partner has enabled receiving
SND_REM_DEST_ERR   : Send State : error message from the partner:
                     The destination DB may not exist or may be too short.

SND_IDLE           : Send State : sending part is idle, ready for next job.
```

Network 11 : here: Time only for demonstation , SND_RDY may be very short.

Comment:



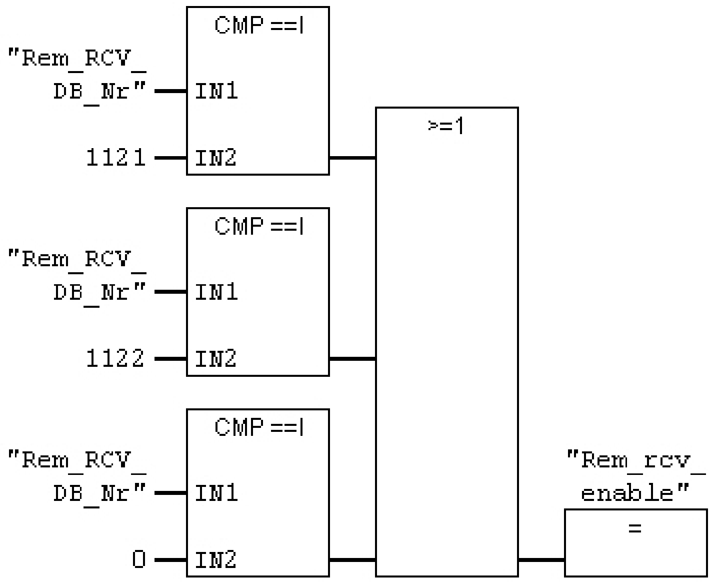
Symbol information:

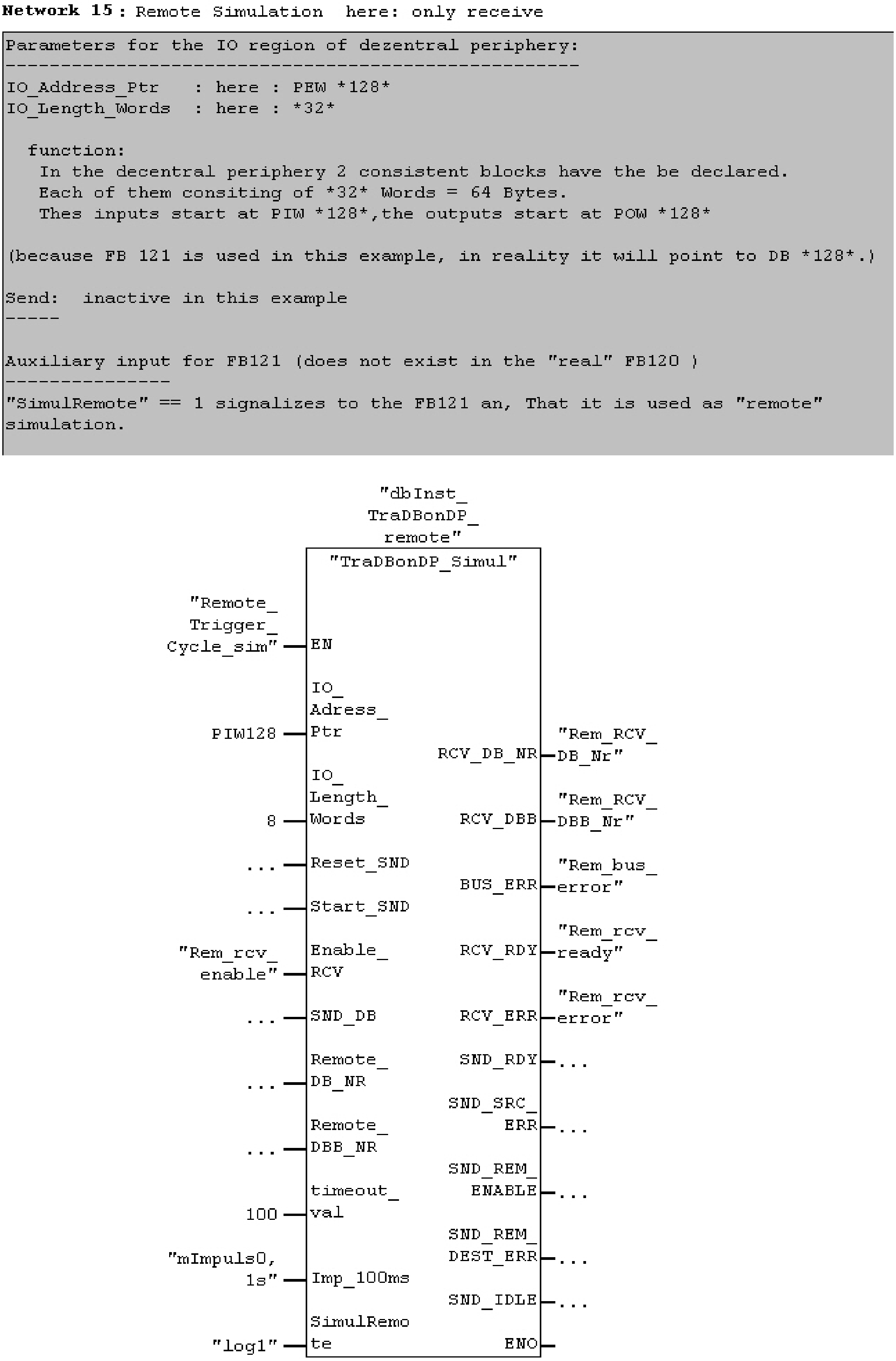
Network 14 : Enable receive for DB 1121 and DB 1122 only

If no receive is enabled: set to FALSE.
If receive is enabled generally : set to TRUE.

This circuit enab les a selective receive.
here: Only DB 1121 and DB1122 are allowed.

The 3. path (compare with 0)is necessary,to enable tghe first telegram which only contains the header (destination informations).

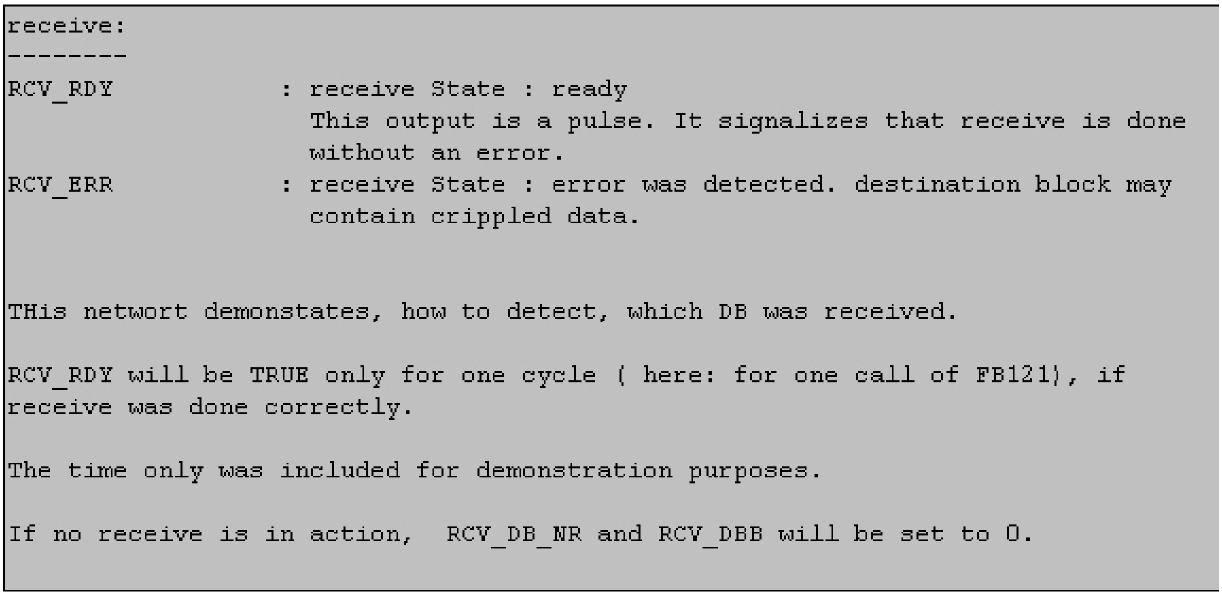




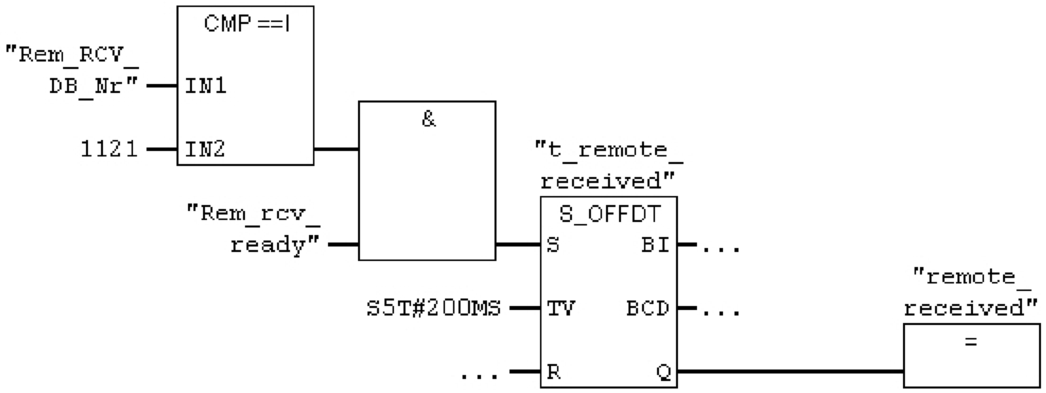
Network 15 : Remote Simulation here: only receive

```
Parameters for the IO region of dezentral periphery:
----------------------------------------------------
IO_Address_Ptr    : here : PEW *128*
IO_Length_Words   : here : *32*

  function:
   In the decentral periphery 2 consistent blocks have the be declared.
   Each of them consiting of *32* Words = 64 Bytes.
   Thes inputs start at PIW *128*,the outputs start at POW *128*

(because FB 121 is used in this example, in reality it will point to DB *128*.)

Send:  inactive in this example
-----

Auxiliary input for FB121 (does not exist in the "real" FB120 )
---------------
"SimulRemote" == 1 signalizes to the FB121 an, That it is used as "remote"
simulation.
```

Network 16 : OUTPUTS RECEIVE



```
receive:
---------
RCV_RDY           : receive State : ready
                    This output is a pulse. It signalizes that receive is done
                    without an error.
RCV_ERR           : receive State : error was detected. destination block may
                    contain crippled data.


THis networt demonstates, how to detect, which DB was received.

RCV_RDY will be TRUE only for one cycle ( here: for one call of FB121), if
receive was done correctly.

The time only was included for demonstration purposes.

If no receive is in action,  RCV_DB_NR and RCV_DBB will be set to 0.
```


CMP ==I
"Rem_RCV_
DB_Nr"
IN1
1121
IN2
&
"Rem_rcv_
ready"
"t_remote_
received"
S_OFFDT
S
BI
...
S5T#200MS
TV
BCD
...
...
R
Q
"remote_
received"
=