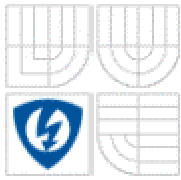

Profibus DP - VHDL BUS Model

Author: Pavel Kučera

© February 2003 by ifak e.V. Magdeburg

Institut für Automation und Kommunikation e.V. Magdeburg
Steinfeldstraße 3 (IGZ Haus 1) +49 (0) 39 203 / 8 10 - 0
D-39179 Barleben +49 (0) 39 203 / 8 11 00

Institutsleiter: Prof. Dr.-Ing. habil. Peter Neumann
www.ifak.fhg.de
info@ifak.fhg.de

1. Content

1. Content....2
2. Goal of the document....3
3. Structure of the model....3
3.1. Estelle specification....3
3.2. Specification of the Devices....4
3.3. Specification of the Equipments....4
4. Description of the model....5
4.1. Application entity....5
4.1.1. DP Master Application - DpMasterAppl....5
4.1.2. DP Slave Application - DpSlaveAppl....6
4.2. Communication entity....7
4.2.1. DP Master communication - DpMasterComm....7
4.2.2. DP Slave communication - DpSlaveComm....8
4.3. Link & Physical layer entity....9
4.3.1. DP Send/Receive Telegram - DpSRTelegram....9
4.3.2. DP UART - DpUART....11
4.4. Telegram Monitor - Monitor....13
4.5. UART Monitor - uartMonitor....14
4.6. UART Electromagnetic Interference - EMI....14
4.7. Telegram Repeater - TelRep1....16
4.8. UART Repeater - UARTRep11....17
4.9. Wireless Link Station - WlLS....18
4.10. Wireless Base Station - WlBS....20
4.11. Wireless Link Base Station - WlLBS....21
4.12. DP Wireless Station - DPwlS....22
4.13. Wireless Electromagnetic Interference - wlEMI....23
5. Configurations of the simulations....24
5.1. Telegram Bus Test - test....25
5.2. Telegram Bus Test - test8....26
5.3. UART Bus Test - test8UART....26
5.4. UART Bus Test - testUART....27
5.5. Telegram Repeater - testTelRep1....28
5.6. UART Repeater - testUARTRep1....28
5.7. Link Station - testLS....29
5.8. Base Station - testBS....29
5.9. Link Base Station - testLBS....30
5.10. Wireless Station - testwlS....30
5.11. Base Station & Wireless Station - testBSwlS....31
6. User-defined types....32
7. References....32

2. Goal of the document

The main intention of this document is to describe VHDL formal model of the Profibus DP fieldbus. Document has four primary objectives:

1. Basic description of the new model of the Profibus DP physical layer - i.e. model based on telegram or bit oriented communication bus.
2. Description of the models of communication entities based on a wire medium.
3. Description of the models of communication entities based on a wireless medium.
4. Description of the configurations for purpose of simulations.

3. Structure of the model

Principle of the model is shown in Fig. 1.

3.1. Estelle specification

Software specification of the Profibus DP communication standard (protocol) is based on Estelle tool. The generated implementation is encapsulated in an executable dynamic library DpDll2.dll. This library is a communication interface for models of the Link and Physical layers. These layers are described by VHDL as well as different kinds of mediums, communication interfaces or influence of the environment (EMI).

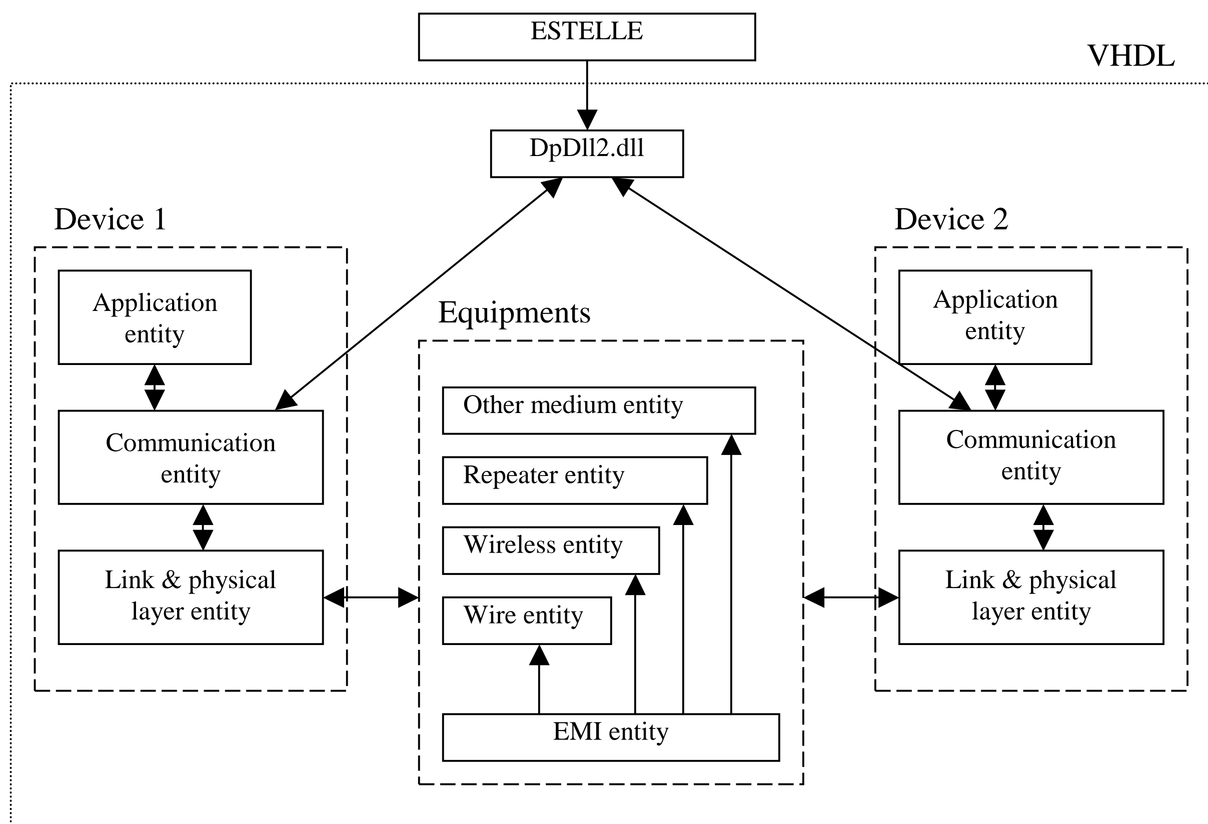

Fig. 1 - Principle of the model

3.2. Specification of the Devices

Every Profibus DP Master/Slave device, except Wireless Station, is described by three different VHDL entities:

1. Application entity - chapter 4.1
2. Communication entity - chapter 4.2
3. Link & Physical layer entity - chapter 4.3

Wireless Station is described by the same Application and Communication entities, however Link & Physical entity is different (due to wireless medium) and the entity is described in the chapter 4.12.

3.3. Specification of the Equipments

Specification of the communication equipments consists of several VHDL models:

1. Telegram Monitor - chapter 4.4
2. UART Monitor - chapter 4.5
3. UART EMI - chapter 4.6
4. Telegram Repeater - chapter 4.7
5. UART Repeater - chapter 4.8
6. Link Station - chapter 4.9
7. Base Station - chapter 4.10
8. Link Base Station - chapter 4.11
9. Wireless Station - chapter4.12
10. Wireless EMI - chapter 4.13

© ifak e.V. Magdeburg 2003

4. Description of the model

4.1. Application entity

Application entity has two different models with two different architectures:

1. DpMasterAppl - for Master Application module; 4.1.1
2. DpSlaveAppl - for Slave Application module; 4.1.2

4.1.1. DP Master Application - DpMasterAppl

file: dpmasterappl.vhd

Interface of the Master Application entity is shown in Fig. 2 and data types of its signals and generic variables are listed in Tab. 1.

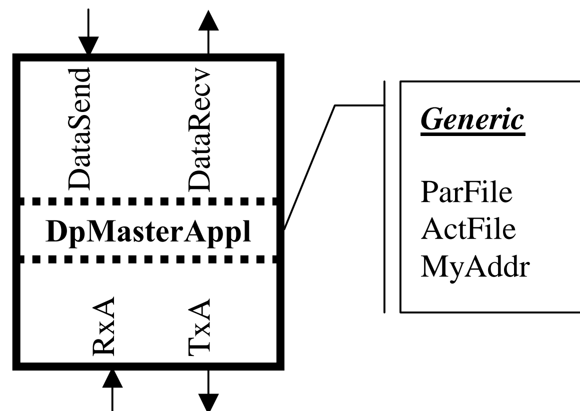

Fig. 2 - DpMasterAppl entity

Tab. 1 - DpMasterAppl; list of signals/variables

Name	***Direction***	***Type***	***Notice***
ParFile	generic	STRING	Pointer to the parametric file
ActFile	generic	STRING	Pointer to the protocol file
MyAddr	generic	INTEGER	Master's Profibus DP address
DataSend	in	byte_vector*	Data to send
DataRecv	out	byte_vector*	Data to receive
RxA	in	byte_vector*	Message to receive
TxA	out	byte_vector*	Message to transmit

* User-defined type; see Tab. 17 on page 32 for details.

Master Application entity consists of the one process called MasterState. State machine of this process is shown in Fig. 3. Master is in the state STOP at the beginning. Once the message with OK status is received, i.e. RxA(2) = 00h, the master enter the CLEAR state and it attempts to parameterize and configure its slaves. Once OK message is received again, the master enter the OPERATE state and it is in user data operation with the slave(s).

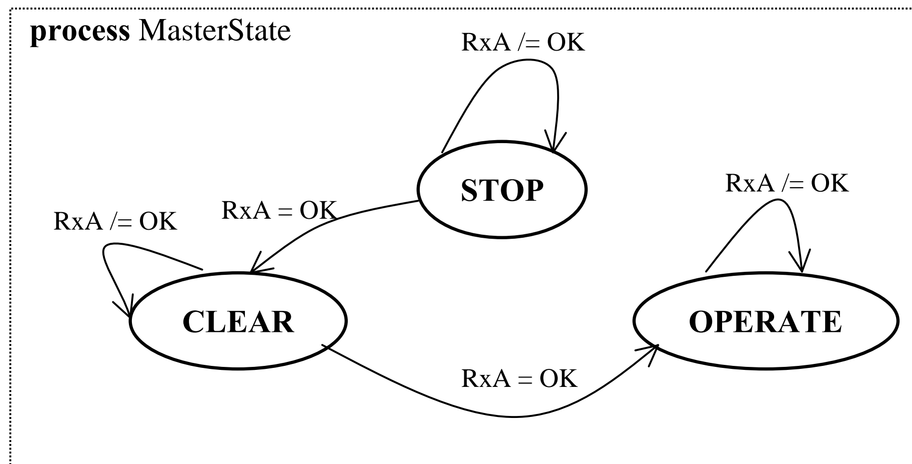

Fig. 3 - State machine of the Master Application

4.1.2. DP Slave Application - DpSlaveAppl

file: dpslaveappl.vhd

Interface of the Slave Application entity is shown in Fig. 4 and data types of its signals and generic variables are listed in Tab. 2.

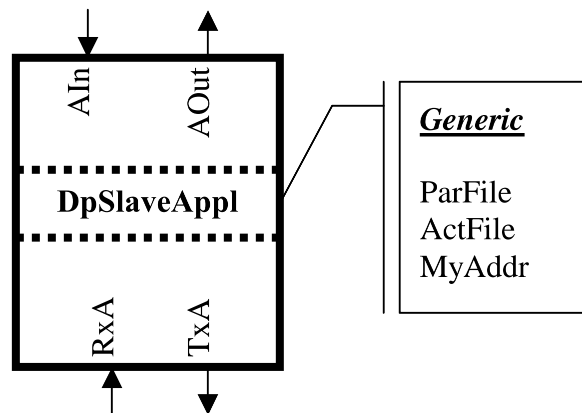

Fig. 4 - DpSlaveAppl entity

Tab. 2 - DpSlaveAppl; list of signals/variables

Name	*Direction*	*Type*	*Notice*
ParFile	generic	STRING	Pointer to the parametric file
ActFile	generic	STRING	Pointer to the protocol file
MyAddr	generic	INTEGER	Slave's Profibus DP address
AIn	in	bytes*	Data to send
AOut	out	bytes*	Data to receive
RxA	in	byte_vector*	Message to receive
TxA	out	byte_vector*	Message to transmit

* User-defined type; see Tab. 17 on page 32 for details.

Note: Implementation of the DpSlaveAppl architecture is not realized.

4.2. Communication entity

Communication entity has two different models with two different architectures:

1. DpMasterComm - for Master Communication module; 4.2.1
2. DpSlaveComm - for Slave Communication module; 4.2.2

4.2.1. DP Master communication - DpMasterComm

file: dpmastercomm.vhd

Interface of the Master Communication entity is shown in Fig. 5 and data types of its signals and generic variables are listed in Tab. 3.

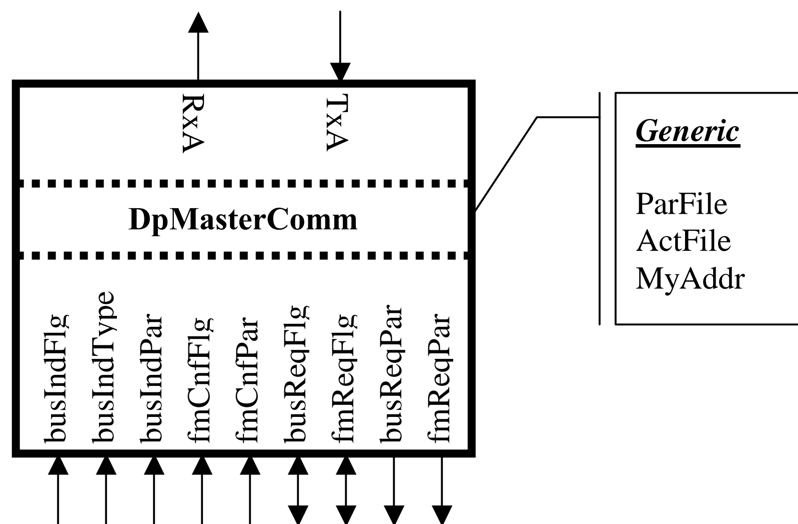

Fig. 5 - DpMasterComm entity

Tab. 3 - DpMasterComm; list of signals/variables

Name	*Direction*	*Type*	*Notice*
ParFile	generic	STRING	Pointer to the parametric file
ActFile	generic	STRING	Pointer to the protocol file
MyAddr	generic	INTEGER	Master's Profibus DP address
RxA	in	byte_vector*	Message to application
TxA	out	byte_vector*	Message from application
busIndFlg	in	BIT	Marker to DpMasterComm
busIndType	in	estBusActivityT*	Message to DpMasterComm
busIndPar	in	byte_vector*	Message to DpMasterComm
fmCnfFlg	in	BIT	Marker to DpMasterComm
fmCnfPar	in	byte_vector*	Message to DpMasterComm
BusReqFlg	buffer	BIT	Marker from DpMasterComm
fmReqFlg	buffer	BIT	Marker form DpMasterComm
busReqPar	out	byte_vector*	Message from DpMasterComm
fmReqPar	out	byte_vector*	Message from DpMasterComm

* User-defined type; see Tab. 17 on page 32 for details.

Master communication architecture is an interface for Master device between DpDll2.dll executable library and VHDL model. It consists of 4 processes:

1. AppProc - trigged by TxA event; then it calls function estDpMasterCommEnq with appropriate message (TxA) from the Application model (DpMasterAppl). If the call is successful then AppProc starts the MainProc.
2. busIndProc - trigged by busIndFlg event; then it calls function estDpMasterCommEnq with appropriate busIndType and busIndPar messages. If the call is successful then busIndProc starts the MainProc.
3. fmCnfProc - trigged by fmCnfFlg; then it calls function estDpMasterCommEnq with appropriate fmCnfPar message. If the call is successful then busIndProc starts the MainProc.
4. MainProc - trigged by any of AppProc, busIndProc or fmCnfProc processes; then it calls ddl function estDpMasterCommAct.

Note: Detailed information about communication functions is in (Hintze and Donath, 2001).

4.2.2. DP Slave communication - DpSlaveComm

file: dpslavecomm.vhd

Interface of the Slave Communication entity is shown in Fig. 6 and data types of its signals and generic variables are listed in Tab. 4.

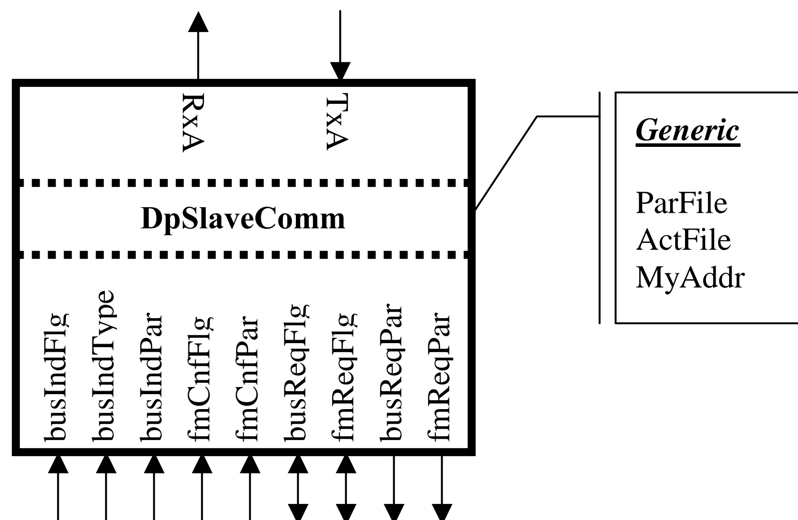

Fig. 6 - DpSlaveComm entity

Tab. 4 - DpSlaveComm; list of signals/variables

Name	***Direction***	***Type***	***Notice***
ParFile	generic	STRING	Pointer to the parametric file
ActFile	generic	STRING	Pointer to the protocol file
MyAddr	generic	INTEGER	Slave's Profibus DP address
RxA	in	byte_vector*	Message to application
TxA	out	byte_vector*	Message from application

busIndFlg	in	BIT	Marker to DpSlaveComm
busIndType	in	estBusActivityT*	Message to DpSlaveComm
busIndPar	in	byte_vector*	Message to DpSlaveComm
fmCnfFlg	in	BIT	Marker to DpSlaveComm
fmCnfPar	in	byte_vector*	Message to DpSlaveComm
BusReqFlg	buffer	BIT	Marker from DpSlaveComm
fmReqFlg	buffer	BIT	Marker form DpSlaveComm
busReqPar	out	byte_vector*	Message from DpSlaveComm
fmReqPar	out	byte_vector*	Message from DpSlaveComm

* User-defined type; see Tab. 17 on page 32 for details.

Slave communication architecture is an interface for Slave device between DpDll2.dll executable library and VHDL model. It consists of 4 processes:

1. AppProc - trigged by TxA event; then it calls dll function estDpSlaveCommEnq with appropriate message (TxA) from the Application model (DpSlaveAppl). If the call is successful then AppProc starts the MainProc.
2. busIndProc - trigged by busIndFlg event; then it calls dll function estDpSlaveCommEnq with appropriate busIndType and busIndPar messages. If the call is successful then busIndProc starts the MainProc.
3. fmCnfProc - trigged by fmCnfFlg; then it calls dll function estDpSlaveCommEnq with appropriate fmCnfPar message. If the call is successful then busIndProc starts the MainProc.
4. MainProc - trigged by any of AppProc, busIndProc or fmCnfProc. Then process calls ddl function estDpMasterCommAct.

Note: Detailed information about communication function is in (Hintze and Donath, 2001).

4.3. Link & Physical layer entity

Link & Physical layer entity is an interface between DP communication entity and different communication mediums. There are two entities for wire communication medium in the model:

1. DpSRTelegram - for telegram-oriented bus; 4.3.1
2. DpUART - for bit-oriented bus; 4.3.2

and one entity for wireless communication medium - DpwlS; 4.12.

4.3.1. DP Send/Receive Telegram - DpSRTelegram

file: dpsrtelegram.vhd

Interface of the DP Send/Receive Telegram entity is shown in Fig. 7 and data types of its signals and generic variables are listed in Tab. 5.

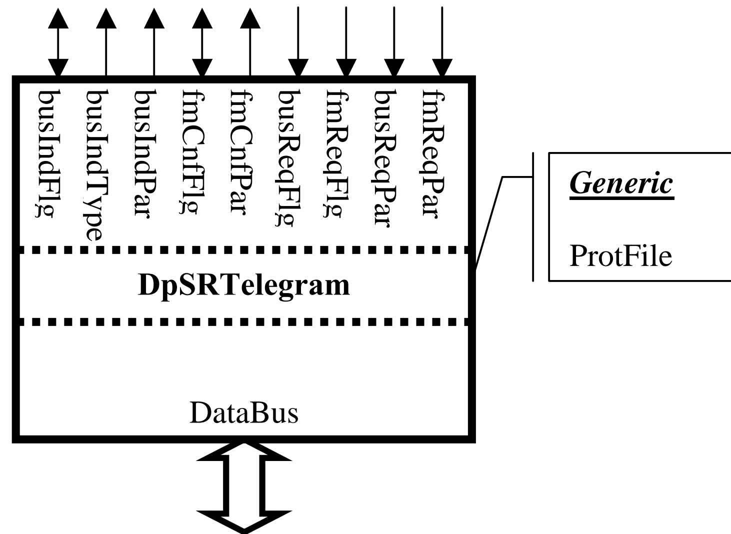

Fig. 7 - DpSRTelegram entity

Tab. 5 - DpSRTelegram; list of signals/variables

Name	*Direction*	*Type*	*Notice*
ProtFile	generic	STRING	Pointer to the protocol file
busIndFlg	buffer	BIT	Marker to Comm module
busIndType	out	estBusActivityT*	Message to Comm module
busIndPar	out	byte_vector*	Message to Comm module
fmCnfFlg	buffer	BIT	Marker to Comm module
fmCnfPar	out	byte_vector*	Message to Comm module
BusReqFlg	in	BIT	Marker from Comm module
fmReqFlg	in	BIT	Marker form Comm module
busReqPar	in	byte_vector*	Message from Comm module
fmReqPar	in	byte_vector*	Message from Comm module
DataBus	inout	TDataBus*	Telegram oriented bus medium

* User-defined type; see Tab. 17 on page 32 for details.

DataBus is a signal that models common telegram oriented data medium. All communication devices are connected together through this medium (in case of the telegram oriented configuration). Therefore *inout* direction is necessary in the interface of the DataBus signal. This signal is based on user-defined type TDataBus. The type allows modeling 'Read' and 'Write' access to the bus for each of the communication device. Since VHDL normally allows only one driver for a signal it is necessary to use resolved type for signals based on TDataBus type. A resolved type is based on resolution function that uses defined access right to determine the final signal value. In our case, the resolution function is implemented as follows:

```
type TDataBusArray is array (integer range <>) of TDataBus;
function resolve_bus (drivers:  in TDataBusArray) return TDataBus;
subtype TResolvedDataBus is resolve_bus TDataBus;
```

```
function resolve_bus (drivers : in TDataBusArray) return TDataBus is
begin
        for index in drivers'range loop
                if drivers(index).acl = 'W' then
                        return drivers(index);
                end if;
        end loop;
        return drivers(0);
end resolve_bus;
```

Function scans all possible DataBus signals and returns the first signal with 'Write' access right. Because, under normal circumstances, only one device in Profibus DP bus is allowed to write data on the bus, this function realizes simple multiplexer device controlled by a priority coder (from the write access point of view). The priority is expressed by the DataBus signal with access level 'W' with the lowest index. Definitions of the types and implementation of the resolution function is in package: testtypepack.vhd.

DpSRTelegram architecture consists of 3 processes:

1. Process Receive - triggered by 'W' event on the common telegram oriented DataBus. Process is responsible for correct transfer of the telegram from DataBus structure to the superior communication module (DpMasterComm or DpSlaveComm).
2. Process fmReqProc - is responsible for configuration of the bus parameters, e.g. baud rate, station address etc.
3. Process Send is responsible for correct transfer of the telegram from superior communication module (DpMasterComm or DpSlaveComm) to the DataBus.

4.3.2. DP UART - DpUART

file: dpuart.vhd

Interface of the DP UART entity is shown in Fig. 8 and data types of its signals and generic variables are listed in Tab. 6.

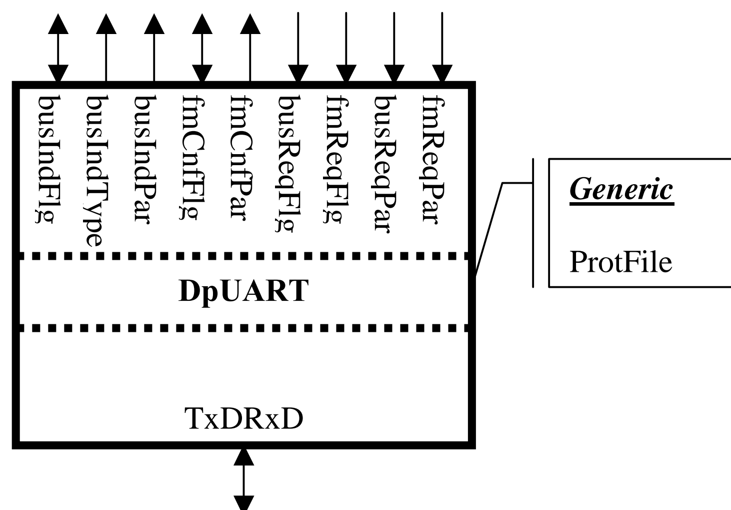

Fig. 8 - DpUART entity

Tab. 6 - DpUART; list of signals/variables

Name	***Direction***	***Type***	***Notice***
ProtFile	generic	STRING	Pointer to the protocol file
busIndFlg	buffer	BIT	Marker to Comm module
busIndType	out	estBusActivityT*	Message to Comm module
busIndPar	out	byte_vector*	Message to Comm module
fmCnfFlg	buffer	BIT	Marker to Comm module
fmCnfPar	out	byte_vector*	Message to Comm module
BusReqFlg	in	BIT	Marker from Comm module
fmReqFlg	in	BIT	Marker form Comm module
busReqPar	in	byte_vector*	Message from Comm module
fmReqPar	in	byte_vector*	Message from Comm module
TxDRxD	inout	uart_level*	Bit oriented bus medium

[*] User-defined type; see Tab. 17 on page 32 for details.

TxDRxD is a signal that models common bit oriented data medium - RS485. All communication devices are connected together through this medium (in case of the bit oriented configuration). Therefore *inout* direction is necessary in the interface of the TxDRxD signal. This signal is based on user-defined type uert_level:

```
type uart_level is (SPACE, MARK, EMI_SPACE, EMI_MARK);
```

SPACE = log. 1 - default
MARK = log. 0
EMI_SPACE - Electromagnetic Interference SPACE
EMI_MARK - Electromagnetic Interference MARK

Defined states SPACE and MARK correspond to states defined in Recommended Standard for asynchronous serial communication; i.e. SPACE is idle state and MARK active state on the bus. Beside it, two another states for electromagnetic interference are defined: EMI_SPACE and EMI_MARK. As was mentioned in the chapter 4.3.1, resolution function has to be implemented:

```
type uart_array is array (integer range <>) of uart_level;
function resolve_uart(drivers: in uart_array) return uart_level;
subtype TResolvedUARTLevel is resolve_uart uart_level;

function resolve_uart (drivers : in uart_array) return uart_level is
variable ret: uart_level;
begin
        ret := SPACE;
        for index in drivers'range loop
                if drivers(index) = EMI_MARK then
                        return MARK;
                end if;
                if drivers(index) = EMI_SPACE then
                        return SPACE;
                end if;
                if drivers(index) = MARK then
                        ret := MARK;
                end if;
        end loop;
        return ret;
end resolve_uart;
```

Function scans all possible TxDRxD signals and returns the first appearance of the state EMI_MARK or EMI_SPACE or MARK in this order. If there is no such appearance on the bus, then function returns SPACE state. It is evident, that EMI has the highest priority and in every case it overrides any of the regular states (MARK or SPACE). Detailed information about EMI model is in the chapter 4.6. Definitions of the types and implementation of the resolution function is in package: testtypepack.vhd.

DpUART architecture consists of 4 processes:

1. Process Receive - triggered by START condition on the TxDRxD bus. Process receives asynchronous serial telegram and translates it into structure required by the superior communication module (DpMasterComm or DpSlaveComm). If the reception and translation is successful then the telegram is transferred to the communication module.
2. Process fmReqProc - is responsible for configuration of the bus parameters, e.g. baud rate, station address etc.
3. Process Send translates communication structure from the superior communication module (DpMasterComm or DpSlaveComm) into the real Profibus DP telegram structure and transmits this structure over asynchronous serial line.
4. Process req1 provides resolution function (logical OR) for busIndType and busIndFlg signals.

4.4. Telegram Monitor - Monitor

file: monitor.vhd

This entity monitors the telegram-oriented bus. Its interface is shown in Fig. 9 and data types of its signal and generic variables are listed in Tab. 7.

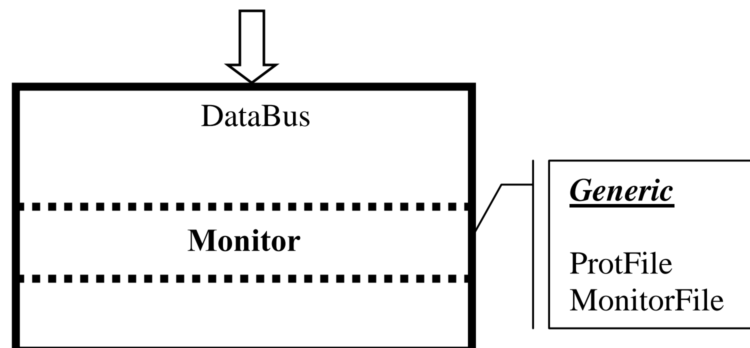

Fig. 9 - Monitor entity

Tab. 7 - Monitor; list of signals/variables

Name	*Direction*	*Type*	*Notice*
ProtFile	generic	STRING	Pointer to the protocol file
MonitorFile	generic	STRING	Pointer to the monitor file
DataBus	in	TDataBus*	Telegram oriented bus medium

* User-defined type; see Tab. 17 on page 32 for details.

Generic variable ProtFile is used as entry value for BUMO_Init function. Detailed information about DataBus signal is in the chapter 4.3.1. Architecture has only one process - Scan. This process detects DataBus telegram on the bus and calls monitor functions: BUMO_StartTransmit and BUMO_StopTransmit. Process also creates monitor file (name of this file is specified in generic variable MonitorFile), where time markers and contents of the particular telegrams are archived.

4.5. UART Monitor - uartMonitor

file: uartmonitor.vhd

This entity monitors the bit-oriented bus. Its interface is shown in Fig. 10 and data types of its signal and generic variables are listed in Tab. 8.

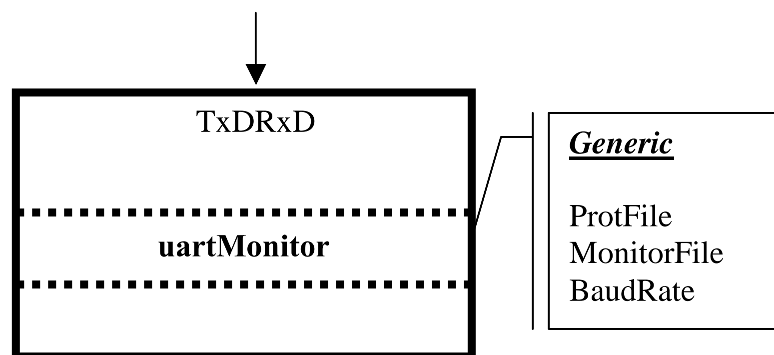

Fig. 10 - uartMonitor entity

Tab. 8 - uartMonitor; list of signals/variables

Name	*Direction*	*Type*	*Notice*
ProtFile	generic	STRING	Pointer to the protocol file
MonitorFile	generic	STRING	Pointer to the monitor file
BaudRate	generic	INTEGER	BaudRate of the medium
TxDRxD	in	uart_level*	Bit oriented bus medium

[*] User-defined type; see Tab. 17 on page 32 for details.

Generic variable ProtFile is used as entry value for BUMO_Init function. Detailed information about TxDRxD signal is in the chapter 4.3.2. Generic variable BaudRate determines communication speed of the monitor. Architecture has only one process - Scan. This process detects TxDRxD signal and receives Profibus DP link layer telegram. Then the received telegram is transformed into structure required by monitor functions BUMO_StartTransmit and BUMO_StopTransmit and they are called. Process also creates monitor file (name of this file is specified in generic variable MonitorFile), where time markers, contents of the particular telegrams and occurred errors are archived.

4.6. UART Electromagnetic Interference - EMI

file: emi.vhd

This entity stimulates bit-oriented bus by the defined EMI model. Its interface is shown in Fig. 11 and data types of its signal and generic variable are listed in Tab. 9.

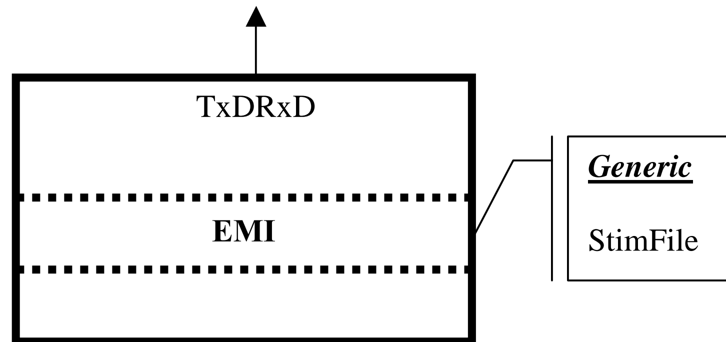

Fig. 11 - EMI entity

Tab. 9 - EMI; list of signal/variable

Name	*Direction*	*Type*	*Notice*
StimFile	generic	STRING	Pointer to the stimulation file
TxDRxD	out	uart_level*	Bit oriented bus medium

* User-defined type; see Tab. 17 on page 32 for details.

Detailed information about TxDRxD signal is in the chapter 4.3.2. Architecture has only one process - Stimul. This process reads text file defined in generic variable (StimFile) and changes bus state according to its content. Structure of the StimFile is shown in Fig. 12.

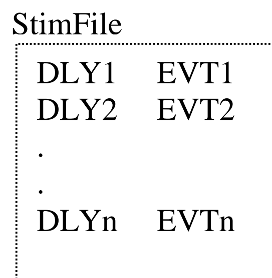

Fig. 12 - Structure of the StimFile

Structure of the StimFile is sequential list of EMI events. These events are defined by the parameters DLY and EVT. Every event = new raw in the file.

- DLY is a time delay [ms] between previous defined EMI event and this new event. Since DLY1 has no previous defined EMI, this delay is measured from the beginning of the simulation.
- EVT is an enumerate type of EMI event. Three different EVT values are possible:

0 - for EMI_MARK disturbance
1 - for EMI_SPACE disturbance
any - for no EMI disturbance

For instance, if the StimFile includes following list:

```
14.0    0
6.0     1
8.0     x
```

then process Stimul generates on the TxDRxD bus EMI as is shown in Fig. 13.

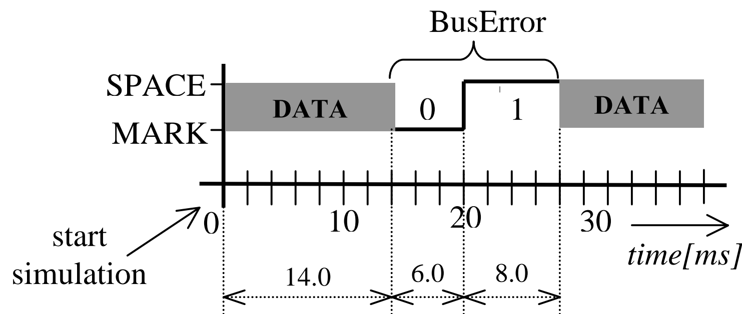

Fig. 13 - An example of the error on the bus

4.7. Telegram Repeater - TelRep1

file: telrep1.vhd

This entity models bus repeater with constant propagation delay for telegram oriented bus (TDataBus). Interface of the entity is shown in Fig. 14 and data types of its signal and generic variable are listed in Tab. 10.

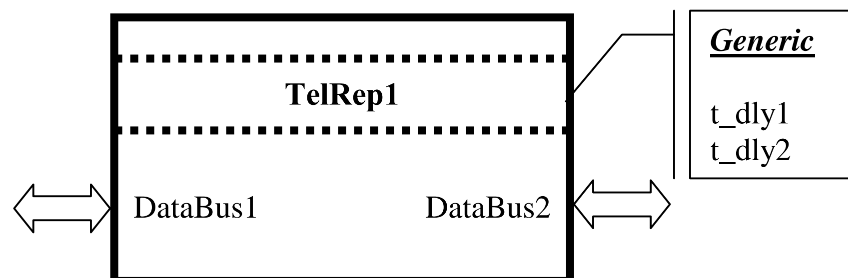

Fig. 14 - TelRep1 entity

Tab. 10 - TelRep1; list of signals/variables

Name	*Direction*	*Type*	*Notice*
t_dly1	generic	TIME	Propagation delay between segment 1 and 2
t_dly1	generic	TIME	Propagation delay between segment 2 and 1
DataBus1	inout	TDataBus*	Bus segment 1
DataBus2	inout	TDataBus*	Bus segment 2

* User-defined type; see Tab. 17 on page 32 for details.

Meaning of the variables t_dly1 and t_dly2 is evident from the Fig. 15.

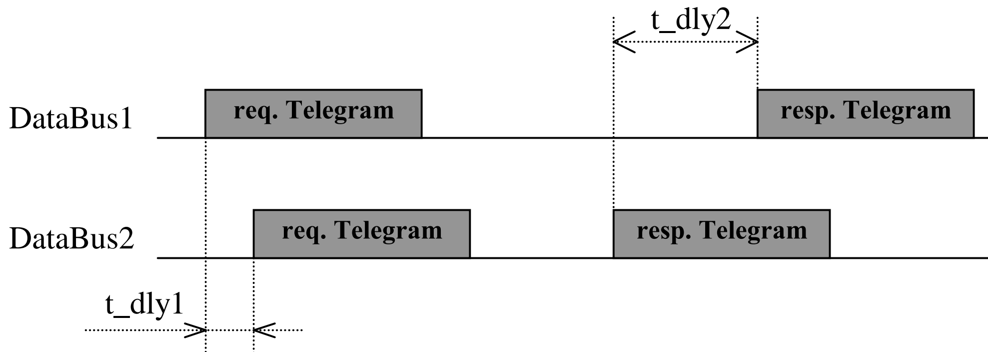

Fig. 15 - Time behaviour of the TelRep1

4.8. UART Repeater - UARTRep11

file: uartrep1.vhd

This entity models bit-oriented bus repeater. This repeater supports different communication speeds on the connected segments. Interface of the entity is shown in Fig. 16 and data types of its signal and generic variable are listed in Tab. 11.

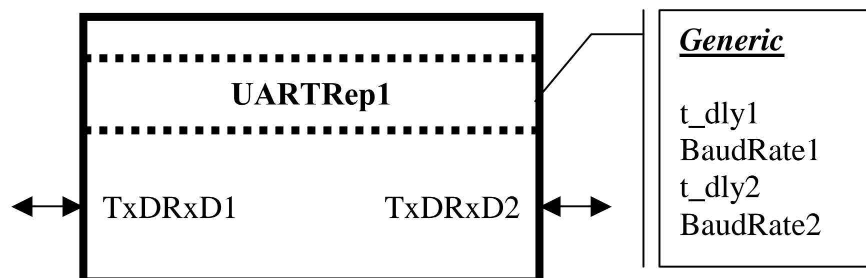

Fig. 16 - UARTRep1 entity

Tab. 11 - UARTRep1; list of signals/variables

Name	*Direction*	*Type*	*Notice*
t_dly1	generic	TIME	Propagation delay between segment 1 and 2
BaudRate1	generic	INTEGER	Baud rate of the segment 1
t_dly1	generic	TIME	Propagation delay between segment 2 and 1
BaudRate2	generic	INTEGER	Baud rate of the segment 2
TxDRxD1	inout	uart_level*	Bus segment 1
TxDRxD2	inout	uart_level*	Bus segment 2

* User-defined type; see Tab. 17 on page 32 for details.

BaudRate1 and BaudRate2 are generally different as well as t_dly1 and t_dly2. Propagation delay of the telegram between segments is at least t_dly1(2). It can be greater (t_gap) due to different communication speeds on the segments, see Fig. 17 for details.

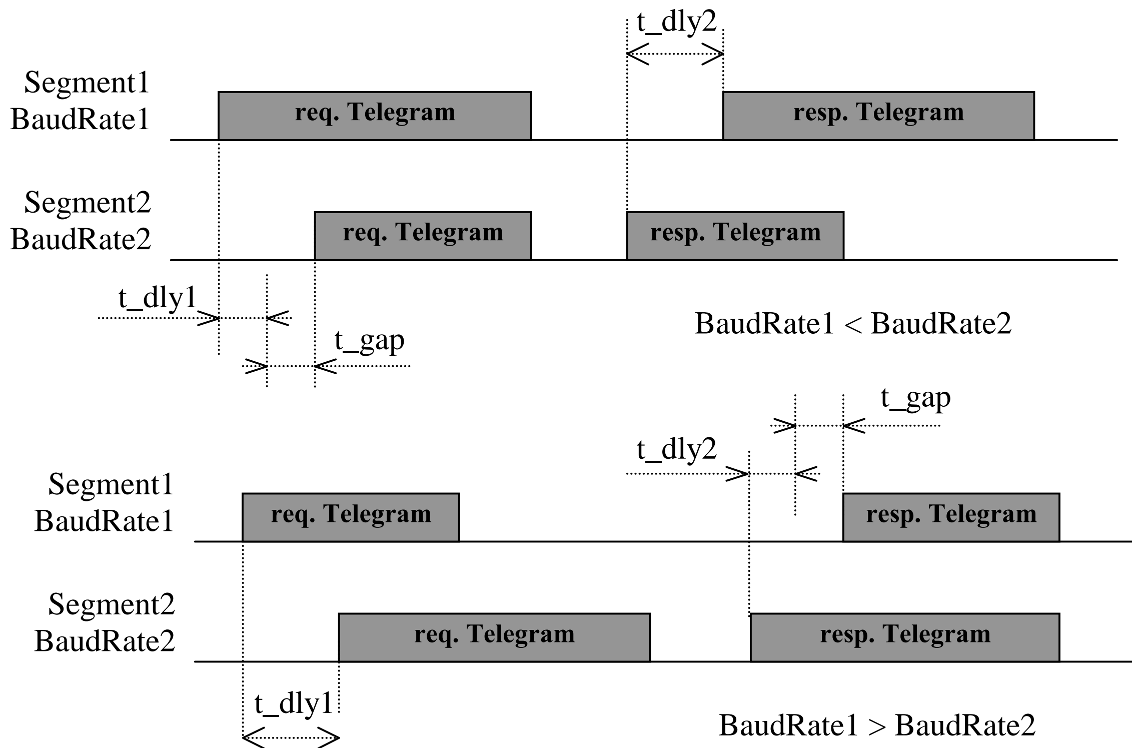

Fig. 17 - Time behaviour of the UARTRep1

4.9. Wireless Link Station - WlLS

file: wlls.vhd

This entity models behavior of the Wireless Station (WS). WS is a repeater between wire and wireless part of the Profibus DP network. Interface of the entity is shown in Fig. 18 and data types of its signal and generic variable are listed in Tab. 12.

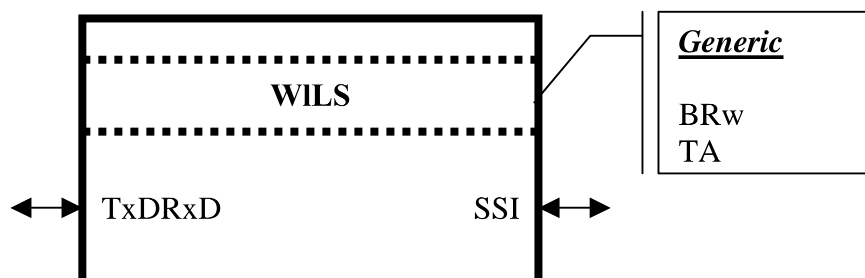

Fig. 18 - WlLS entity

Tab. 12 - WlLS; list of signals/variables

Name	*Direction*	*Type*	*Notice*
BRw	generic	INTEGER	Baud Rate of the wire segment
TA	generic	bits8*	Transmitter Address
TxDRxD	inout	uart_level*	Wire medium

SSI	inout	ssi_level*	Wireless medium

* User-defined type; see Tab. 17 on page 32 for details.

SSI (Synchronous Serial Interface) is a signal that models wireless bit-oriented communication medium. This signal is based on user-defined type ssi_level:

```
type ssi_level is (L0, L1, EMI_L0, EMI_L1);
```

L0 = log. 0
L1 = log. 1
EMI_L0 - log. 0 EMI
EMI_L1 - log. 1 EMI

Defined states L0 and L1 correspond to the logical states use for transmitting signal by RFieldbus Radio Physical Layer Protocol (Adamczyk and coll., 2001). Beside it, two another states for electromagnetic interference are defined: EMI_L0 and EMI_L1. For this new type the resolution function has to be defined:

```
type ssi_array is array (integer range <>) of ssi_level;
function resolve_ssi (drivers: in ssi_array) return ssi_level;
subtype TResolvedSSILevel is resolve_ssi ssi_level;

function resolve_ssi (drivers : in ssi_array) return ssi_level is
variable ret: ssi_level;
begin
        ret := L0;
        for index in drivers'range loop
                if drivers(index) = EMI_L1 then
                        return L1;
                end if;
                if drivers(index) = EMI_L0 then
                        return L0;
                end if;
                if drivers(index) = L1 then
                        ret := L1;
                end if;
        end loop;
        return ret;
end resolve_ssi;
```

Resolution function scans all possible SSI signals in the project and returns the first appearance of the state EMI_L1 or EMI_L0 or L1 in this order. If there is no such appearance on the bus, then function returns L0 state. It is evident, that EMI has the highest priority and in every case it overrides any of the regular states (L0 or L1). Detailed information about wireless EMI model is in the chapter 4.13. Definitions of the types and implementation of the resolution function is in package: testtypepack.vhd.

RFFieldbus PhLPDU telegram format is implemented as well as it is described in (Adamczyk and coll., 2001 - chapter 6), except RFieldbus Phy Preamble. In VHDL model, the Preamble is generated as 30 us bit pattern of L0 and L1 states on the bus - see Fig. 19.

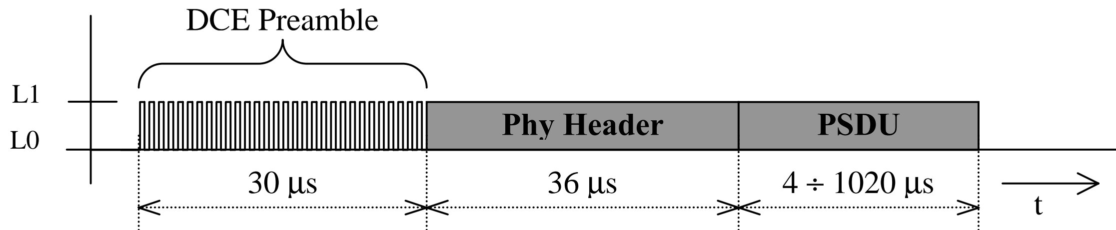

Fig. 19 - VHDL model of the PhLPDU

During DCE Preamble every SSI receiver tunes its Baud Rate generator such a way that at the end of the Preamble, the final Baud Rate (BR) value is equal to:

$$BR = \frac{59}{\sum_{n=1}^{59} DLY_n} \qquad \text{[Baud]}$$

where DLY is a time delay between the minimum state change on the bus during synchronization - see Fig. 20.

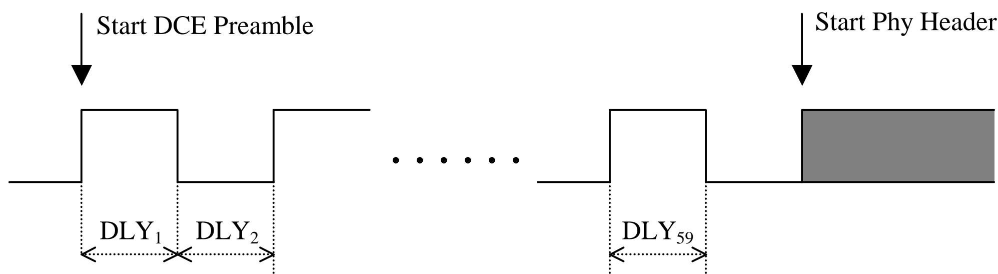

Fig. 20 - Detail of the DCE Preamble

Function behavior of the LS model corresponds to the (Koulamas and Koubias, 2002; chapter 2.4.)

4.10. Wireless Base Station - WlBS

file: wlbs.vhd

This entity models behavior of the Base Station (BS). BS is a repeater between two wireless segments. Interface of the entity is shown in Fig. 21 and data types of its signal and generic variable are listed in Tab. 13.

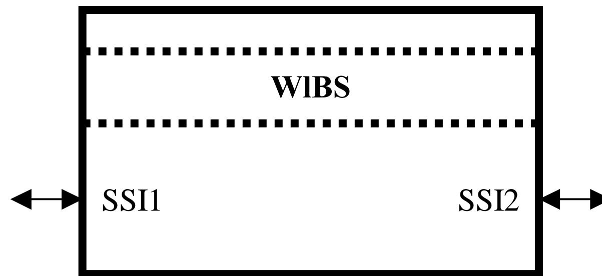

Fig. 21 - WlBS entity

Tab. 13 - WlBS; list of signals/variables

Name	*Direction*	*Type*	*Notice*
SSI1	inout	ssi_level*	Wireless segment 1
SSI2	inout	ssi_level*	Wireless segment 2

* User-defined type; see Tab. 17 on page 32 for details.

Function behavior of the BS model corresponds to the (Koulamas and Koubias, 2002; chapter 2.5), except the beacon mechanism that is not implemented and only two wireless segments (SSI1 and SSI2) are possible.

4.11. Wireless Link Base Station - WlLBS

file: wllbs.vhd

This entity models behavior of the Link Base Station (LBS). LBS is a repeater between two wireless segments and one wire segment. Interface of the entity is shown in Fig. 22 and data types of its signal and generic variable are listed in Tab. 14.

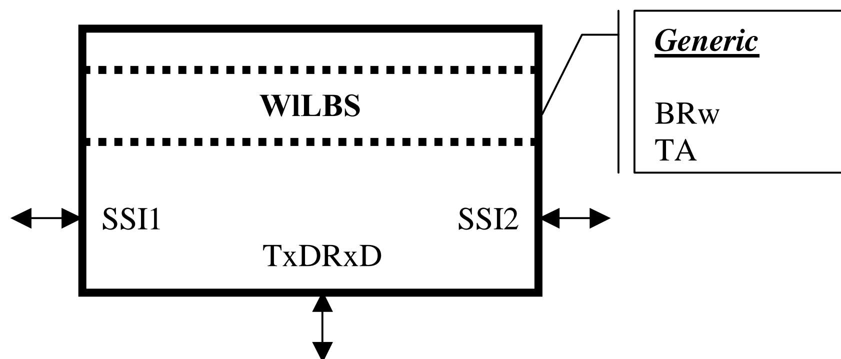

Fig. 22 - WlLBS entity

Tab. 14 - WlLBS; list of signals/variables

Name	*Direction*	*Type*	*Notice*
BRw	generic	INTEGER	Baud Rate of the wire segment
TA	generic	bits8*	Transmitter Address
SSI1	inout	ssi_level*	Wireless segment 1
TxDRxD	inout	uart_level*	Wire segment
SSI2	inout	ssi_level*	Wireless segment 2

* User-defined type; see Tab. 17 on page 32 for details.

Function behavior of the LBS model corresponds to the (Koulamas and Koubias, 2002; chapter 2.4 and 2.5), except the beacon mechanism that is not implemented and only two wireless segments (SSI1 and SSI2) are possible.

4.12. DP Wireless Station - DPwlS

file: dpwls.vhd

Wireless Station (WS) entity is an interface between DPMasterComm or DPSLaveComm module and wireless segment of the network. Interface of the WS entity is shown in Fig. 23 and data types of its signals and generic variables are listed in Tab. 15.

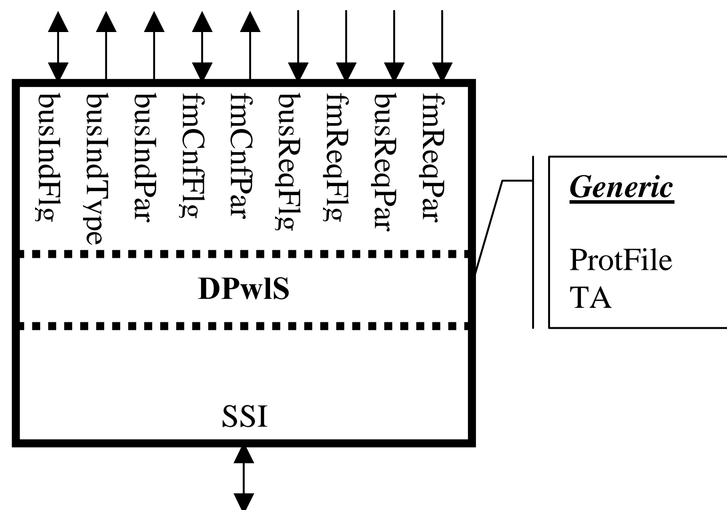

Fig. 23 - DPwlS entity

Tab. 15 - DPwlS; list of signals/variables

Name	***Direction***	***Type***	***Notice***
ProtFile	generic	STRING	Pointer to the protocol file
busIndFlg	buffer	BIT	Marker to Comm module
busIndType	out	estBusActivityT*	Message to Comm module
busIndPar	out	byte_vector*	Message to Comm module
fmCnfFlg	buffer	BIT	Marker to Comm module
fmCnfPar	out	byte_vector*	Message to Comm module
BusReqFlg	in	BIT	Marker from Comm module
fmReqFlg	in	BIT	Marker form Comm module
busReqPar	in	byte_vector*	Message from Comm module
fmReqPar	in	byte_vector*	Message from Comm module
SSI	inout	ssi_level*	Wireless medium

* User-defined type; see Tab. 17 on page 32 for details.

4.13. Wireless Electromagnetic Interference - wlEMI

file: wlemi.vhd

This entity stimulates wireless signal by the defined EMI model. Its interface is shown in Fig. 24 and data types of its signal and generic variable are listed in Tab. 16.

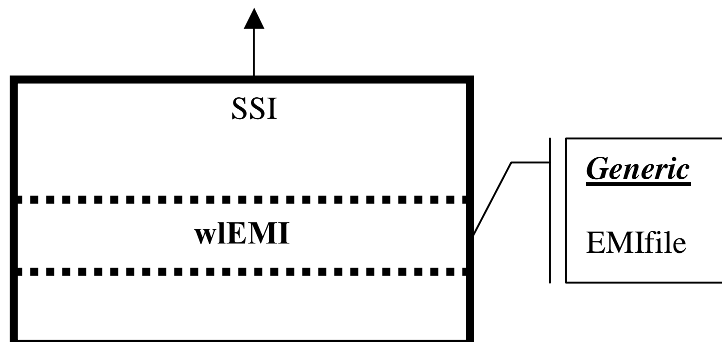

Fig. 24 - wlEMI entity

Tab. 16 - wlEMI; list of signal/variable

Name	*Direction*	*Type*	*Notice*
EMIfile	generic	STRING	Pointer to the stimulation file
SSI	out	ssi_level*	Wireless medium

* User-defined type; see Tab. 17 on page 32 for details.

Detailed information about SSI signal is in the chapter 4.9. Architecture has only one process - Stimul. This process reads text file defined in generic variable (EMIFile) and changes SSI signal according to its content. Structure of the EMIFile is shown in Fig. 25.

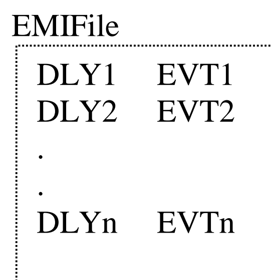

Fig. 25 - Structure of the EMIFile

Structure of the EMIFile is a sequential list of EMI events. These events are defined by the parameters DLY and EVT. Every event = new raw in the file.

- DLY is a time delay [ms] between previous defined EMI event and this new event. Since DLY1 has no previous defined EMI, this delay is measured from the beginning of the simulation.
- EVT is an enumerate type of EMI event. Three different EVT values are possible:

0 - for EMI_L0 disturbance
1 - for EMI_L1 disturbance

any - for no EMI disturbance

For instance, if the EMIFile includes following list:

```
14.0    0
6.0     1
8.0     x
```

then process Stimul generates on the SSI transfer medium EMI as is shown in Fig. 26.

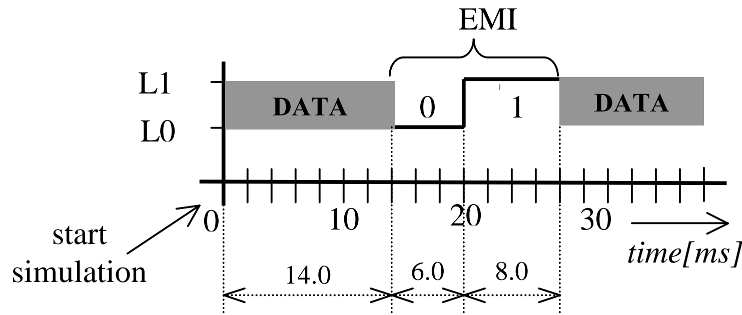

Fig. 26 - An example of the error on the SSI medium

5. Configurations of the simulations

For the purpose of the simulations, several test entities were created. These entities have prefix "test" in the name and include VHDL configuration structure. All simulations require parameterisation file (usually test.par) and all simulations except test and test8 require StimFile (usually emi.txt) for EMI model.

5.1. Telegram Bus Test - test

file: test.vhd
Configuration of the model is shown in Fig. 27.

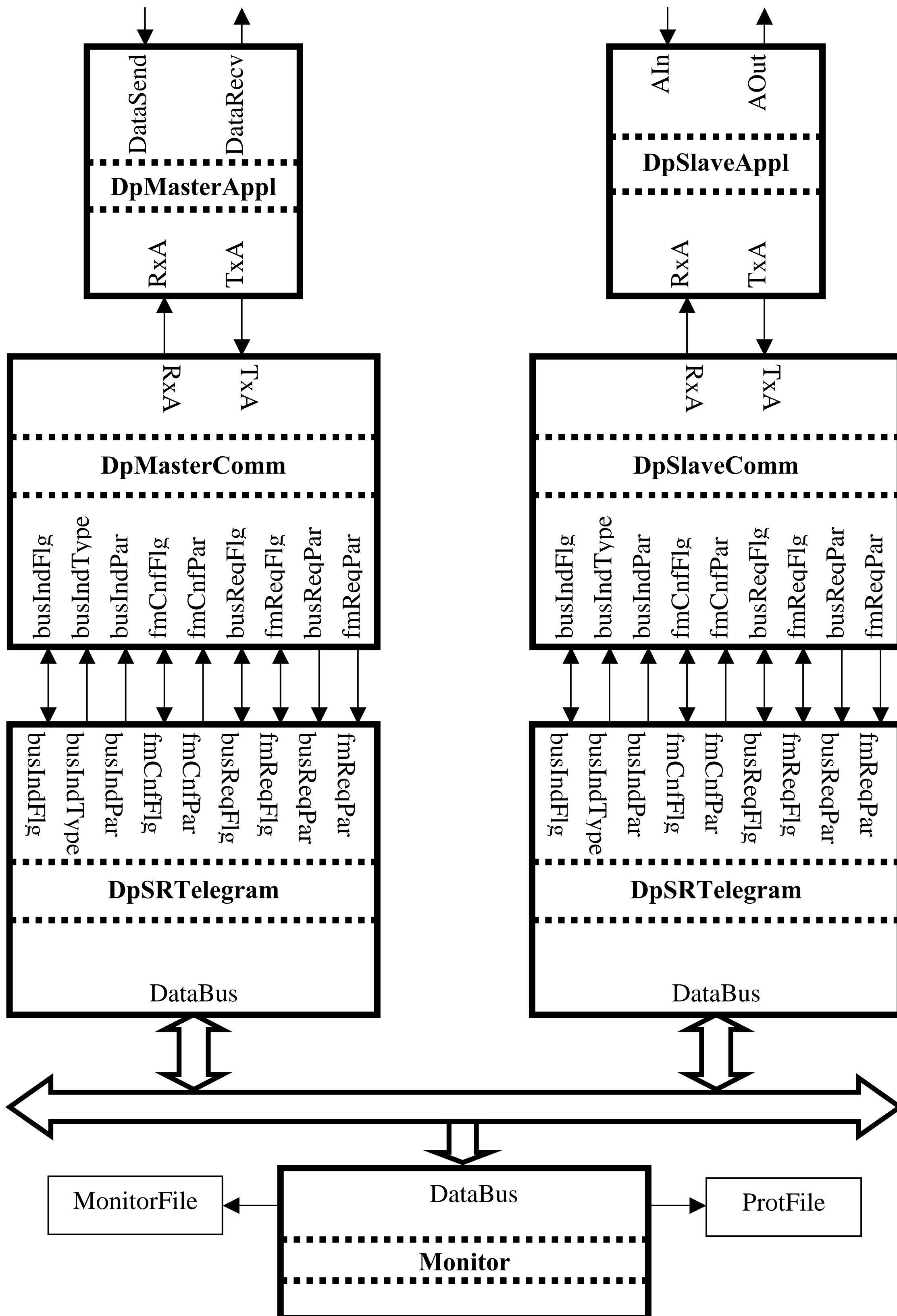

Fig. 27 - Test configuration

5.2. Telegram Bus Test - test8

file: test8.vhd

Simplified configuration of the model is shown in Fig. **28**.

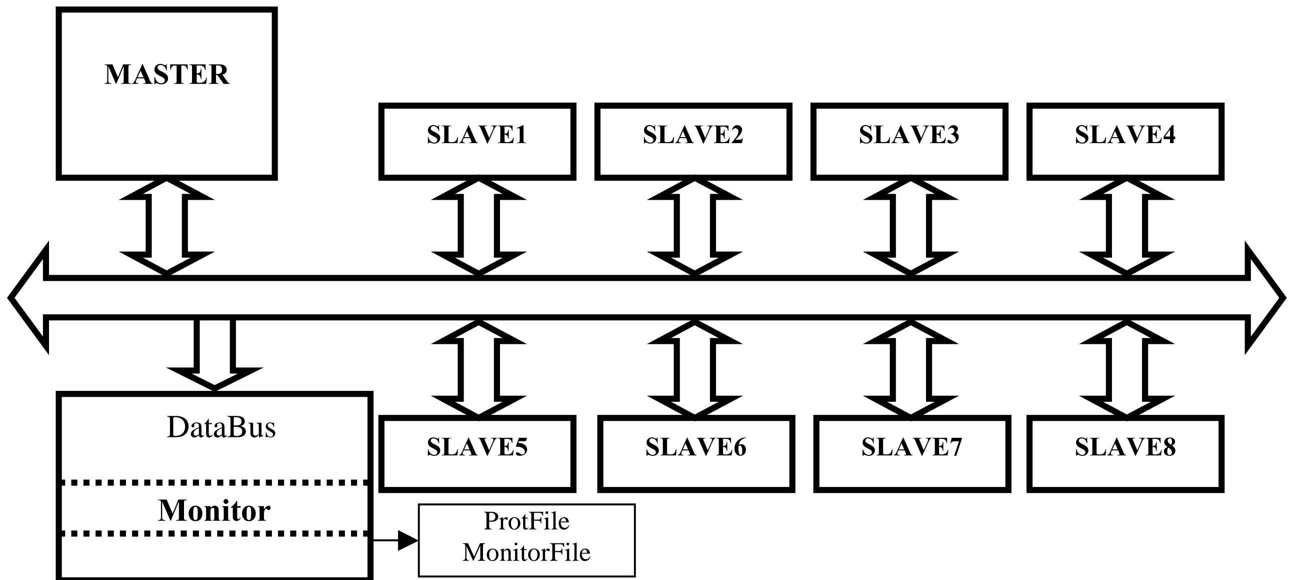

Fig. 28 - Test8 configuration

5.3. UART Bus Test - test8UART

file: test8uart.vhd

Simplified configuration of the model is shown in Fig. 29.

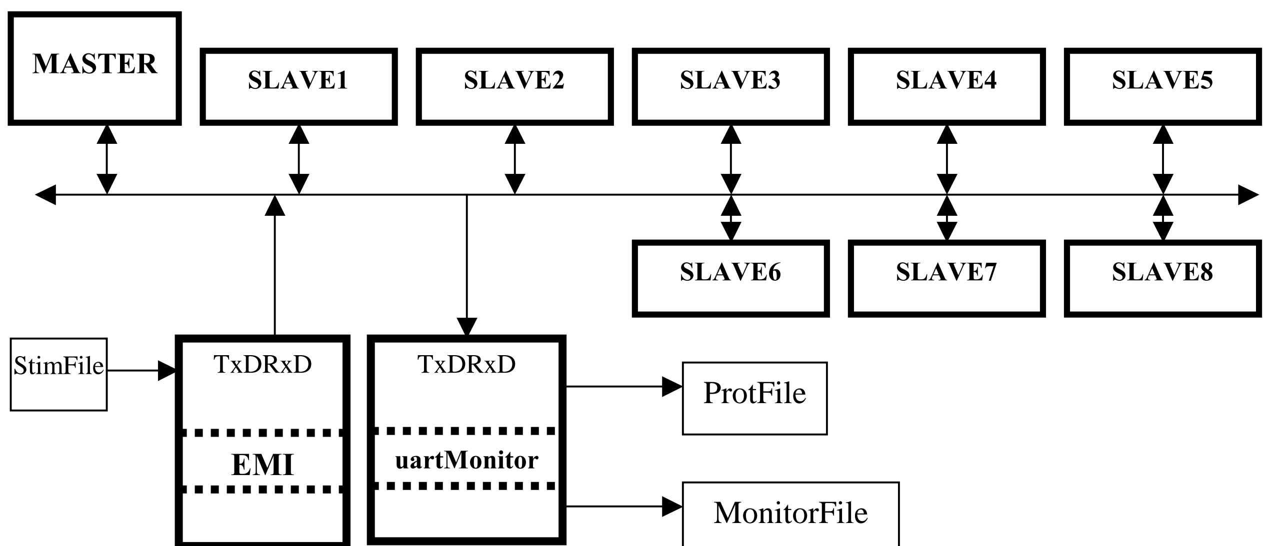

Fig. 29 - Test8UART configuration

5.4. UART Bus Test - testUART

file: testUART.vhd
Configuration of the model is shown in Fig. 30.

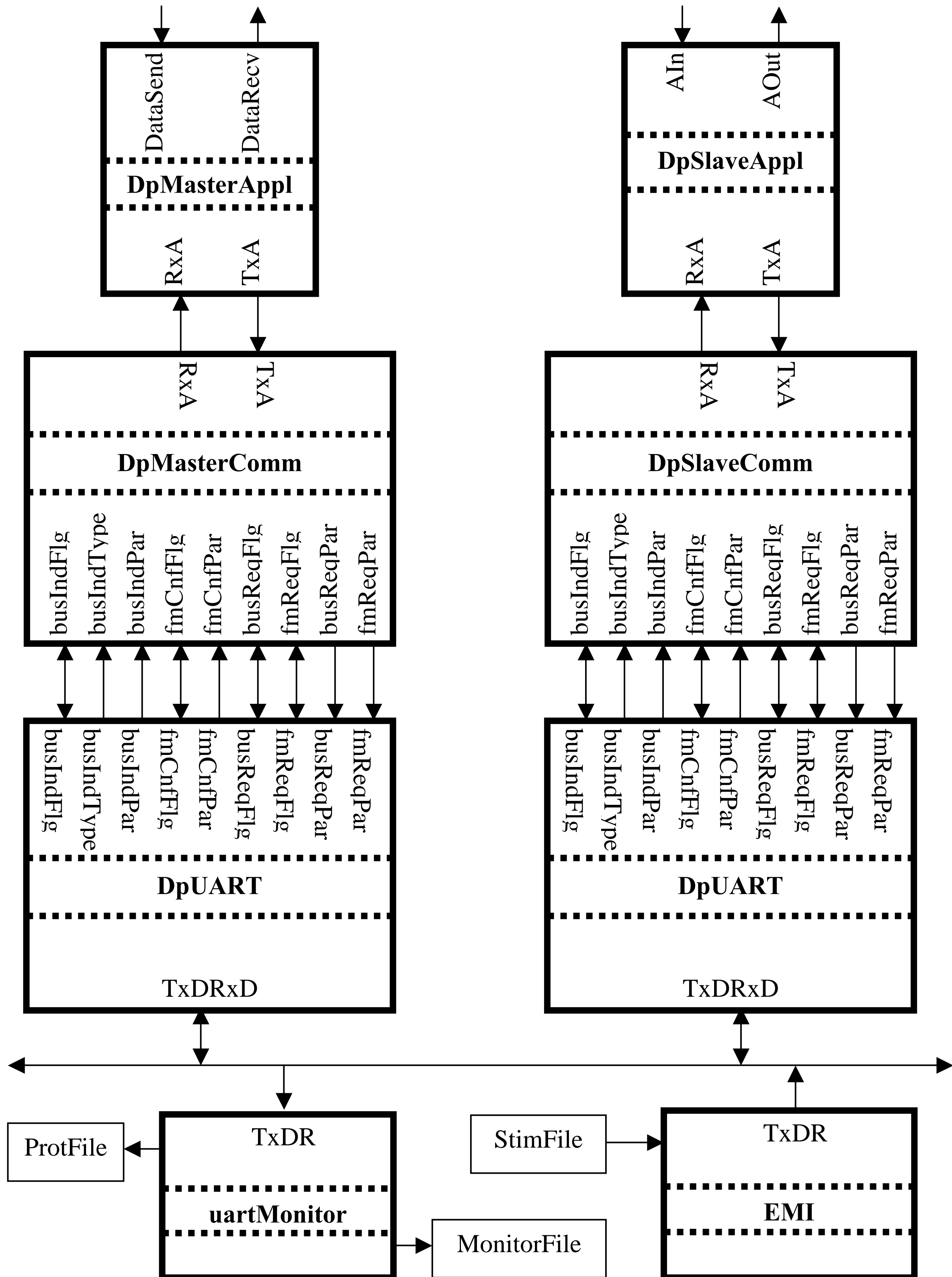

Fig. 30 - testUART configuration

5.5. Telegram Repeater - testTelRep1

file: testTelRep1.vhd
Configuration of the model is shown in Fig. 31.

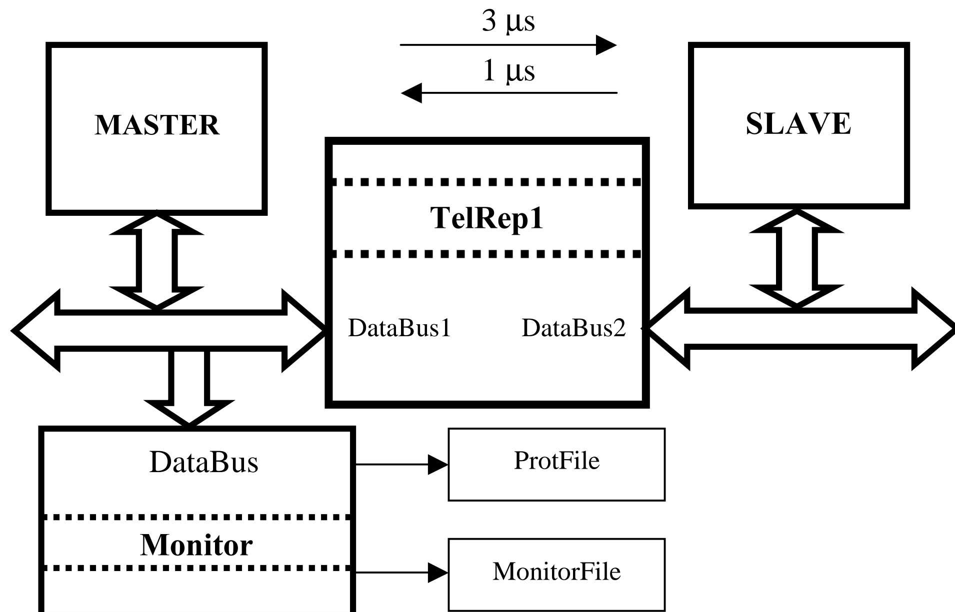

Fig. 31 - testTelRep1 configuration

5.6. UART Repeater - testUARTRep1

file: testuartrep1.vhd
Configuration of the model is shown in Fig. 32.

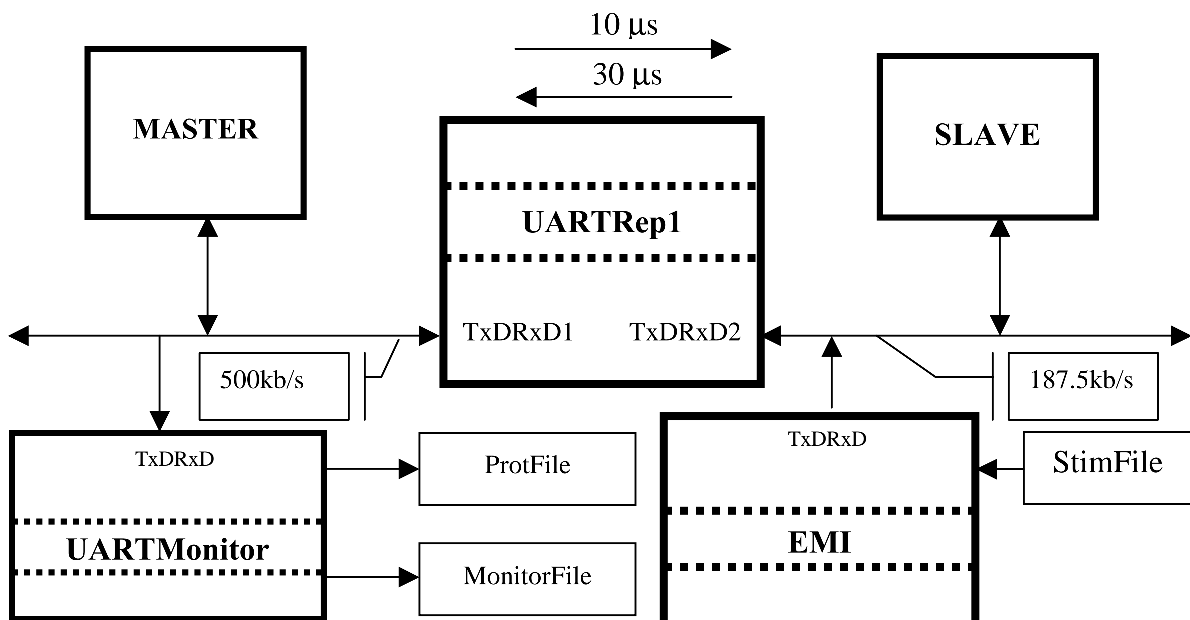

Fig. 32 - testTelRep1 configuration

5.7. Link Station - testLS

file: testls.vhd
Configuration of the model is shown in Fig. 33.

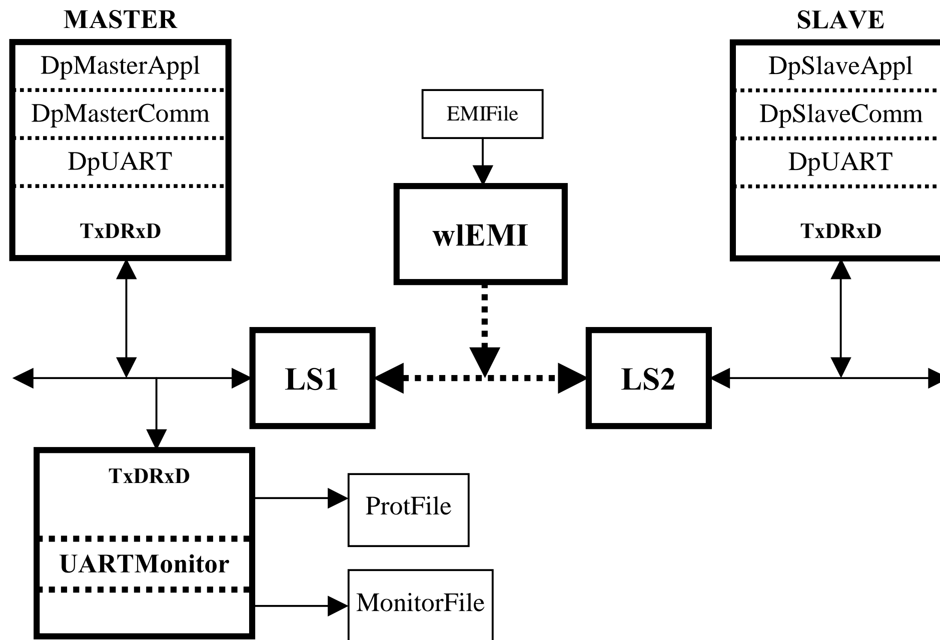

Fig. 33 - testLS configuration

5.8. Base Station - testBS

file: testbs.vhd
Configuration of the model is shown in Fig. 34.

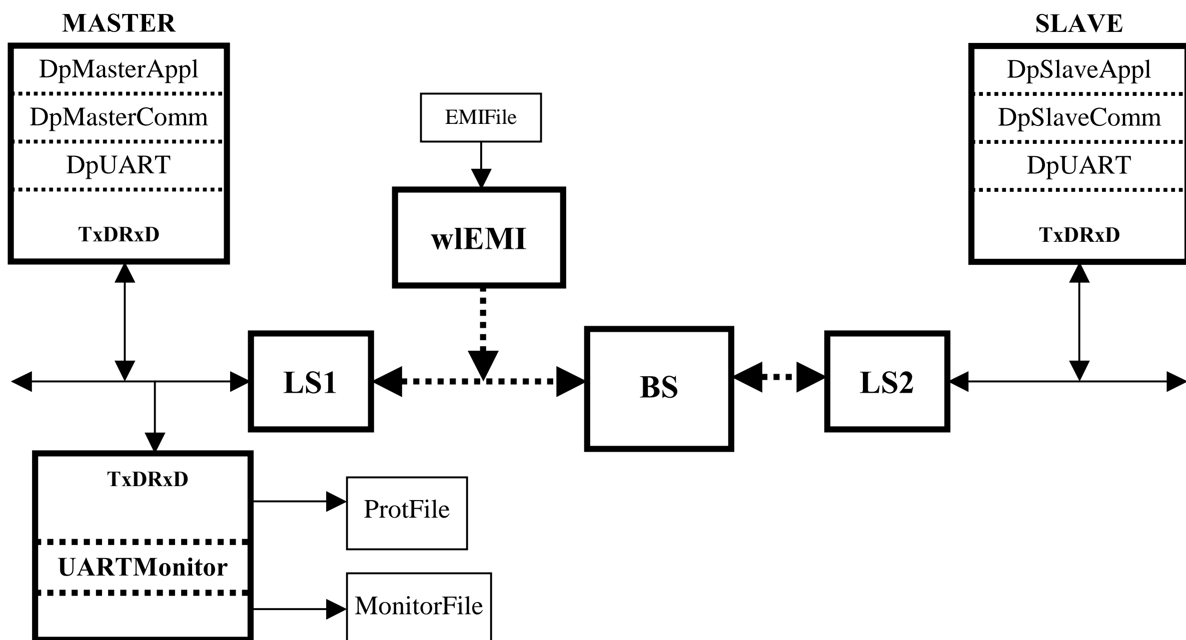

Fig. 34 - testBS configuration

5.9. Link Base Station - testLBS

file: testlbs.vhd

Configuration of the model is shown in Fig. 35.

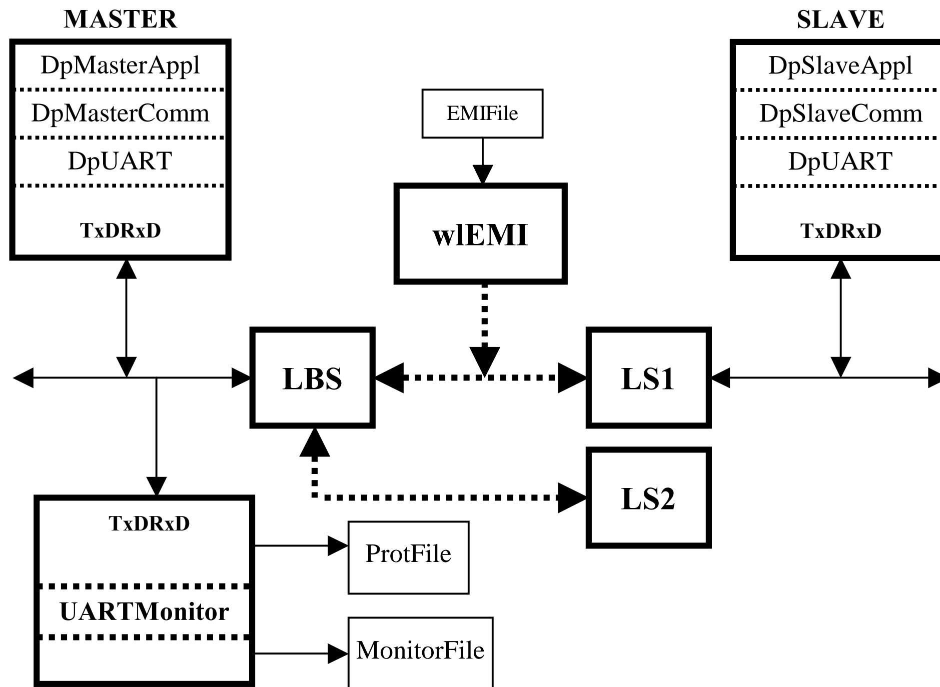

Fig. 35 - testLBS configuration

5.10. Wireless Station - testwlS

file: testwls.vhd

Configuration of the model is shown in Fig. 36.

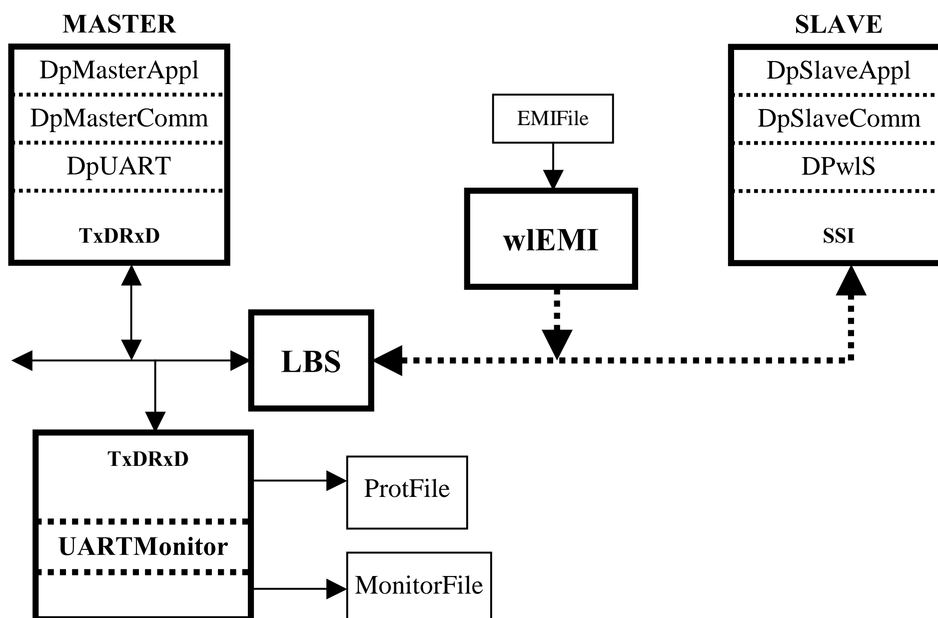

Fig. 36 - testwlS configuration

5.11. Base Station & Wireless Station - testBSwlS

file: testbswls.vhd
Configuration of the model is shown in Fig. 37.

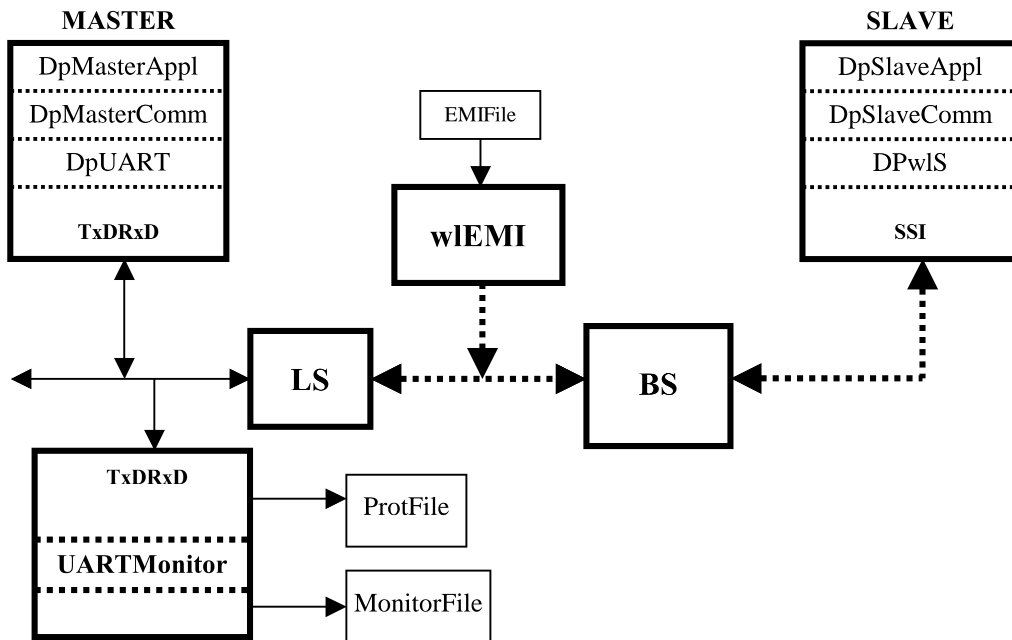

Fig. 37 - testBSwlS configuration

6. User-defined types

file: testtypepack.vhd

Tab. 17 - list of the user-defined types

Type	*VHDL definition*
bits8	bit_vector (7 downto 0)
byte	integer range 0 to 255
byte_vector	array (0 to255) of byte
bytes	array (natural range <>) of byte
estBusActivityT	(NoBusActivity, ReceiveSTART, ReceiveSTOP, FCSErrorSTOP, SDErrorSTOP, SendSTOP, FMIndCnf)
TDataBus	record tlg : byte_vector; acl : access_level;
access_level	('R', 'W')
uart_level	(SPACE, MARK, EMI_SPACE, EMI_MARK)
ssi_level	(L0, L1, EMI_L0, EMI_L1)

7. References

Hintze E. and Donath, U. (2001). In: Schnittstellenspezifikation; ifak e.V. Magdeburg

Adamczyk H., Gendron F., Hammer G., Koulamas C., Lekkas A. (2001). In.: IST-1999-11316 RFieldbus, D2.1.1 - Physical Layer Specification - Part2, Protocol & Operational Characteristics Definition

Koulamas C., Koubias S., (2002). In.: Real-Time Characteristics of the RFieldbus Link Device Bridging Operation