BRNO UNIVERSITY OF TECHNOLOGY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DISSERTATION WORK

for graduation of the academic degree “doctor”, abbreviated as “Ph.D.”

Field of Study

CYBERNETICS, CONTROL AND MEASUREMENT

Pavel Kučera

FORMAL METHODS IN INDUSTRIAL COMMUNICATION

Supervisor: Doc. Ing. František Zezulka, CSc.

Date of state exam: 28th August 2002

Date of delivery of the work: 17th December 2003

The work is accessible at the BUT, FEEC, Research and Foreign Department, Udolni 53, Brno, 602 00, Czech Republic.

1. Keywords

Formal methods, industrial communication, FT control, FT system, VHDL, UPPALL and PLC.

2. Abstract

This dissertation thesis deals with Formal methods in industrial communication. Industrial communication plays an important role in industrial automation due to trend of decentralizing control systems. Industrial automation generally includes many areas of engineering (HW, SW, mechanical, chemical ...) therefore close cooperation between them is necessary. But in generally, all of these engineers use different descriptions and analysing tools. Even the same areas of engineering use different description tools. This work should show one of the possible ways to solve this problem, i.e. using Formal Methods.

Contents

1. Keywords **0**

2. Abstract **0**

3. Introduction **4**

3.1. State of the art *4*
3.1.1. What is* formal methods 4
3.1.2. Software engineering 4
3.1.3. Hardware engineering 7
3.1.4. Industrial communication 8

4. M-Redundancy HW system **11**

4.1. Quantitative evaluation methods - Failure Rate and Reliability Function *12*
4.2. Software reliability model *14*
4.3. Reliability model for industrial buses *20*
4.4. Reliability model of the M-redundancy structure *23*

5. Formal verification **25**

5.1. Temporal Logic *25*
5.2. Temporal operators *26*
5.2.1. Logical operators 26
5.2.2. Temporal operators 27
5.3. UPPAAL *28*
5.3.1. Mutual Exclusion Algorithm check example 29

6. Formal specification of the real-time communication system **34**

6.1. Project's management *34*
6.2. FT control of the railway *36*
6.2.1. Inputs 36
6.2.2. Outputs 37
6.2.3. Control Structure 38
6.2.4. Industrial communication network 40
6.2.5. Control algorithm 41
6.2.6. Error detection 43
6.2.7. I/O error 44
6.2.8. Calculating error 44
6.2.9. Formal description of the control algorithm 45

7. Formal model of the heterogeneous industrial communication bus **58**

7.1. Profibus DP *58*
7.2. Structure of the model *59*
7.3. Application entity *60*
7.3.1. DP Master Application - DpMasterAppl 60
7.3.2. DP Slave Application - DpSlaveAppl 61
7.4. Communication entity *62*

7.4.1. DP Master communication - DpMasterComm 62
7.4.2. DP Slave communication - DpSlaveComm 63
7.5. Link & Physical layer entity 65
7.5.1. DP Send/Receive Telegram - DpSRTelegram 65
7.5.2. DP UART - DpUART 67
7.6. Telegram Monitor - Monitor 69
7.7. UART Monitor - uartMonitor 70
7.8. UART Electromagnetic Interference - EMI 71
7.8.1. Static model with step-by-step defined disturbance 71
7.8.2. Static model with stochastic disturbance. 73
7.8.3. Dynamic model with stochastic disturbance 75
7.9. Telegram Repeater - TelRep1 77
7.10. UART Repeater - UARTRep11 78
7.11. Wireless Link Station - WlLS 79
7.12. Wireless Base Station - WlBS 81
7.13. Wireless Link Base Station - WlLBS 82
7.14. DP Wireless Station - DPwlS 83
7.15. Wireless Electromagnetic Interference - wlEMI 83
7.15.1. Static model with step-by-step defined disturbance 84
7.16. Configurations of the simulations 85
7.16.1. Telegram Bus Test - test 86
7.16.2. Telegram Bus Test - test8 87
7.16.3. UART Bus Test - testUART 87
7.16.4. UART Bus Test - test8UART 88
7.16.5. Telegram Repeater - testTelRep1 89
7.16.6. UART Repeater - testUARTRep1 89
7.16.7. Link Station - testLS 90
7.16.8. Base Station - testBS 90
7.16.9. Wireless Station - testwlS 91
7.16.10. Base Station & Wireless Station - testBSwlS 91
7.17. User-defined types 92

8. Conclusions 93

9. Appendix 96

9.1. SFC to STL transfer 96
9.2. TLAKAN 97

10. Glossary 104

References 109

11. Curriculum Vitae 114

Copyright & Trademarks:

- Intel is registered trademark of Intel Corporation
- Motorola is registered trademarks of Motorola, Inc
- Microsoft is registered trademark of the Microsoft Corporation
- MicroSim is registered trademark of the Microsim Corporation
- Siemens is registered trademark of Siemens AG
- Simatic is registered trademark of Siemens AG
- Unified Modelling Language and UML are trademarks of Object Management Group Inc.

Utere quaesitis modice, cum sumptus abundat,
Labitur exiguo, quod partum est tempore longo.

Disticha Catonis, Liber II

3. Introduction

This chapter briefly introduces description tools used for formulation of the designers' visions in the SW and HW part of the industrial communication systems.

3.1. State of the art

3.1.1. What is* formal methods

Every human activity is based on a certain kind of abstraction. The painter, drawing her/his picture, uses this abstraction directly - the result of her/his activity is a state of her/his imagination during drawing. The composer uses the same technique using another expression tools. In both cases the result of their activity is not a vision in their brain but some kind of expression of it. In fact, people are not able to tell each other their vision directly. They must use a tool. But it can degrade the original human vision. Nice example is Exupery's picture of a hat or a boa constrictor digesting an elephant [Exup00]. Wrong interpretation of a child vision implicates misunderstanding - however beautiful misunderstanding.

Technical engineers also have visions. But in contradistinction to artist, the correct interpretation is decisive. It is a basic assumption for system development and communication and information interchange between engineers.

3.1.2. Software engineering

Let us note the situation in software engineering. Following example shows an Intel's 8086 machine code [Trie92] that adds two 8-bit integral numbers and result leaves in accumulator ACC = (25_h + 19_h):

```
Address:               Operation code:
00000000 00000000      10100000 00100101
00000000 00000010      00000100 00011001
```

* The term *formal methods* to refer to a particular collection of knowledge.

It is a simple example and really simple expression tool. Only two symbols (0,1) are used for description this „vision”. But it is not good idea to program a word processor such a way. Also this language (if we can talk about language) is insufficient because it does not include all necessary information. For instance type of the processor or starting address are not defined. It means that sending this machine code to another software engineer (who can prefer Motorola’s processor) can cause serious misunderstanding. That is reason why higher layer of description must be used - assembly language. The same example written in this language [Vrat97] follows:

```
;*******************************************************************
;*  Adder (c) P.K. 1998 v1.0
;*
;*  MASM, 8086
;*******************************************************************

.MODEL TINY
.8086
.CODE

Main:
  MOV  AL,25h
  ADD  AL,19h
END  Main
```

This language includes all necessary information - type of processor (8086), starting address (Main) and used assembly language (MASM) in addition. This source code is sufficient and any software engineer is able to use the author’s vision (2 bytes adder) in the right way. But this description tools has another serious negative - platform dependence. Non-Intel software engineer cannot use it - of course it is still better than this engineer, with good faith, uses previous machine code written for Intel. Thus another abstraction layer (an abstraction from the processor’s point of view) must be used. Such abstraction level satisfies high-level program languages like Pascal C or C++. The same example written in C [Hero95] follows:

```
int main (void) {
  return 0x25+0x19
}
```

Now we have an original idea (2 bytes adder) described such a way that everybody who is familiarised in C programming language is able to understand this idea.

Presented example is very simple (and useless) and cannot reflect more complicated human imagination. Now let us complicate a designer vision. For instance let's write a program that prints out all permutation of the set [1,2,3,4,5,6,7]. There are 7! = 5040 possible combinations how to permutate this set and our first idea is based on simply algorithm that step-by-step checks all numbers between <1234567, 7654321> and for each of them is calculated its inner product between digits - for instance inner product of the number 5321467 is 5*3*2*1*4*6*7 = 5040. If the inner product of the current number is equal to 7! then all digits inside this number belong to the set of permutations and this number is printed out. C program of this algorithm follows:

```
#include<stdio.h>
unsigned long i;
char a[8];

void main(void) {
      clrscr();
      for (i = 1234567L; i <= 7654321L; i++) {
            sprintf(a,"%lu",i);
            if ((a[0] - 48) * (a[1] - 48) * (a[2] - 48) * (a[3] - 48)
            *(a[4] - 48) * (a[5] - 48) * (a[6] - 48) == 5040)
                  printf("%lu\n",i);
      }
}
```

The idea is simple, algorithm and C program too. But the result is wrong. Our consumption about inner product is necessary condition but not sufficient. Our algorithm operates well up to number 1235377. Following number 1235378 satisfies our necessary condition (1*2*3*5*3*7*8 = 5040) but evidently does not belong to the set of permutations. Of course that wrong algorithm cannot produce good result. But the reason why is algorithm wrong is not only because poor mathematical knowledge of the designer. Another reason is not so evident and it is hidden in the one important thing - designer adapts the algorithm to the possibilities of the programming language. This problem is a nice example of using unsatisfactory describing tool. It is better to describe this problem using a language of predicate logic - Prolog. The main reason is that language of predicate logic implicitly forces programmer to describe the process of permutation and not to explicitly generate the set of all permutations. Such a way it is simpler the process of verifications because it is highly probably that program that recognizes one permutation of the given set is able to generate (using resolution rule during recursion) all possible permutation. Program written in Prolog follows:

```
delete(X,[X|T],T).
delete(X,[Y|T1],[Y|T2]):-
   delete(X,T1,T2).

insert(X,S1,S2):-
 delete(X,S2,S1).

permutation([],[]).
permutation([H1|T1],P):-
   permutation(T1,T2),
 insert(H1,T2,P).
```

The question in Prolog:

```
?- permutation([1,2,3,4,5,6,7],OUT).,
```

gives all 5040 right answers. The understanding this code requires good knowledge about Prolog - reader can find all relevant information in [Brat95].

3.1.3. Hardware engineering

Electronic engineers have much more difficult situation. Many of sufficient methods are commonly used for description electronic systems and at least the same number of analysing tools [Lope64]. Let us show, on the simple example of low-pass filter, important relations between them.

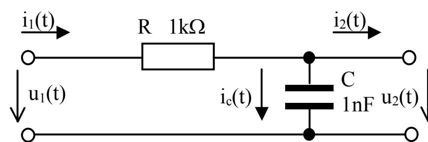

Figure 3-1: Low-pass filter

Schematic diagram of the low-pass filter is shown in Figure 3-1. Schematic diagrams are most frequently used because of standardization. Also many simulations tools use Netlist [Brei95] format:

```
* Schematics Netlist *

C_C          0 $N_0001  C
R_R          $N_0002 $N_0001  R
```

Another form how to describe low-pass filter is to use mathematical description. Following equation has the same meaning like schematic diagram in Figure 3-1:

$$R \cdot C \cdot \frac{du_2(t)}{dt} + u_2(t) = u_1(t) \qquad (3\text{-}1)$$

But unfortunately this description is too formal because describes not only the system in Figure 3-1, but also thousands other systems (from electronic to economy). Single equation above does not tell us what is the purpose of the system. What is a main idea of the designer is hidden while main idea of the designer in Figure 3-1 is evident. But even in Figure 3-1 a misunderstanding could occur. For instance if we consider values C = 1nF and R = 1kΩ and the circuit is used like input filter with bandwidth $< 0,1000 >$ Hz then the serious problem arises; a frequency analysis of this circuit discovers the problem - Figure 3-2.

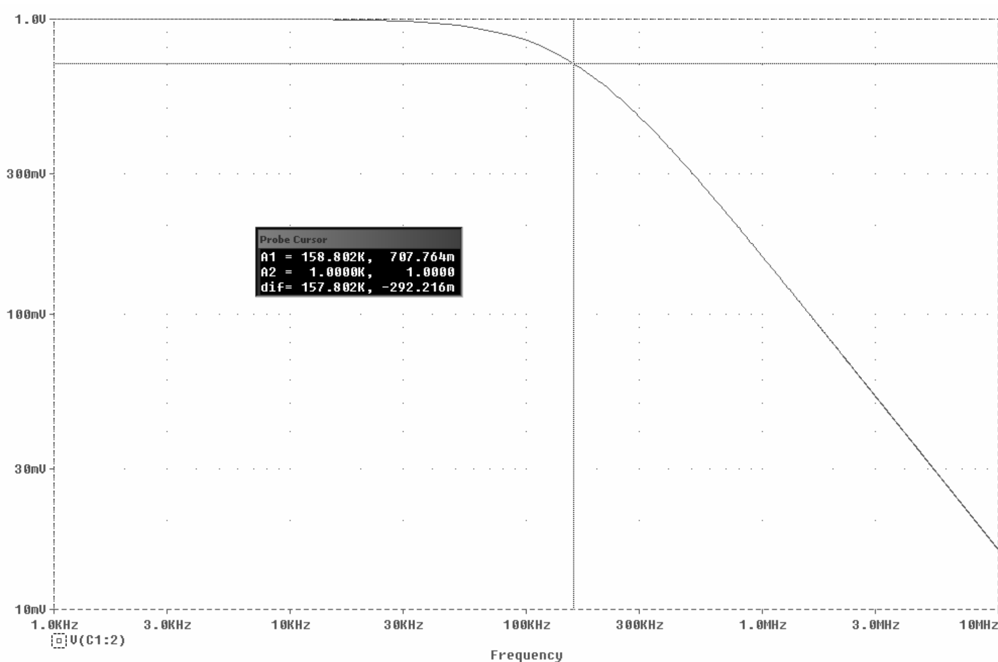

Figure 3-2: Frequency analysis of the low-pass filter

Even more than hundred times higher frequency (159 kHz) then is required highest frequency (1kHz) is not affected by the filter - filter does not fulfil its function. This problem is not obvious neither Figure 3-1 nor differential equation 3-1. Now it is evident that problem with description of such simple circuit must be infinite higher in complex device with thousand components.

3.1.4. Industrial communication

One of the important part of industrial automation deals with communication. An industrial communication is a specific area because [Zezu01], [ZezK01], [Brad02]:

- it is generally based on two-wire transmission link like RS485, CAN, Current loop, LonTalk, AS-i, DeviceNet, etc. that are well described by standard description tools,
- it does not use whole ISO/OSI model and many parts of its communication structure are not simply included in ISO/OSI layers,
- three limitations are critical: High operability, high safety and real-time operation.

Tools for description of the part I (Transmission link) were described in chapter about electronic engineering. Transmission link is always based on some kind of electronic equipment and in the case of communication on the classical theory of signal processing. This classical theory is based on models that are stationary, linear and in many cases also assume that signals have Gaussian amplitude distributions. Among this are non-linear autoregressive and state-space models, models with time-varying or state-dependent coefficients as representations of non-stationary and non-linear series, adaptive methods of forecasting, interpolation and smoothing, linear non-Gaussian methods, and methods derived from the theory of dynamical systems. All these methods are suitable for this part of industrial communication but the area of their activity necessarily ends at the gate of I/O bus of the microprocessor.

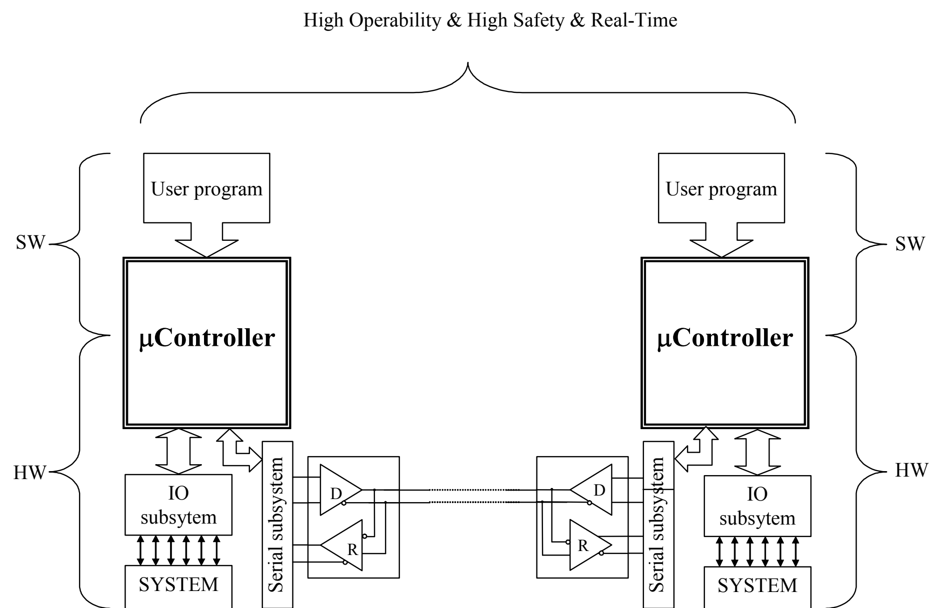

Figure 3-3: An example of industrial communication chain

Microprocessor is presently an interface between hardware engineering and software engineering - Figure 3-3 [Kuce03]. While both these branches have already developed necessary description tools in details the interface between them is from this point of view a serious gap. In other words, it is not a problem to describe, design or analyse electronic or software part of the system but we have not analysing tools for the part of the system between them. We can implement, validate, verify and test electronic and software part of the project separately but then we have no guarantee about implementation, validation verification and testing of the whole project. How to solve this problem? One of the ways is to use the formal description tool for the software and electronic part of the design.

The potential advantages of employing formal methods of system modelling, specification and development are well know, and have led to their utilization in system development being advocated as a means of improving confidence in the dependability of the finished products as is described in [Mose90]. An example of the formal description of the real-time communication system has been presented in author's early work, based on simulation of the communication protocol by VHLD [KucP03] and [KuZe03]. Close cooperation between HW and SW models as well as common simulation is presented in chapter 7 - Formal model of the heterogeneous industrial communication bus. This work demonstrate the way how to guarantee real-time, security and safety industrial communication system behaviour by standard mathematical and non-standard formal description and analytical tools.

4. M-Redundancy HW system

This chapter introduces mathematical background of the developed M-Redundancy concept. This concept is based on idea to combine a conventional wiring CW (point-to-point wiring between control elements CE, and actuated system) and serial industrial communication buses SBs. The main idea is evident from Figure 4-1.

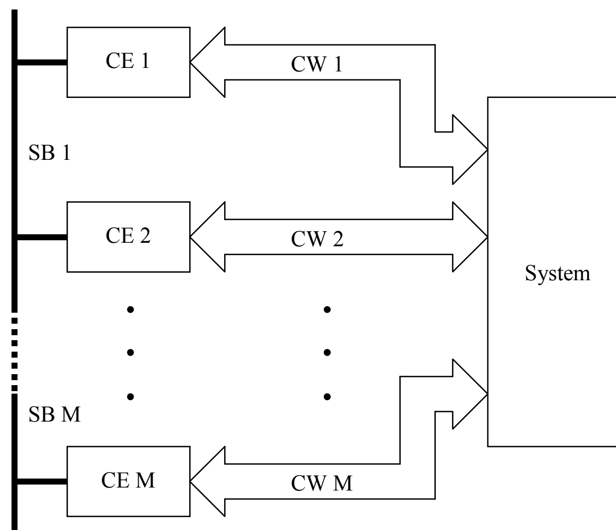

Figure 4-1: M-Redundancy HW system

Control Elements CE 1 to CE M (PLCs) are connected partly with the system by a conventional wiring CW 1 to CW2, and partly together by the serial communications bus segments SB 1 to SB M. These segments generally represent different communication mediums (wire, wireless, optical ...) and standards (Profibus DP/PA, DeviceNet, AS-i ...) and so form heterogeneous communication architecture. Following chapters introduces various formal description techniques with the aim to describe, analyze and easy implement this industrial communication system.

4.1. Quantitative evaluation methods - Failure Rate and Reliability Function

The intention of the quantitative methods is to assign a number to some attribute of a system with the aim to objectively comparison among systems. In our case the failure rate, typically denoted as λ, and reliability function, typically denoted as $R(t)$, will be used to compare existing design of the industrial communication systems and newly developed methodology.

Failure rate is the expected number of failures of a type of system per a given time period [Shoo68]. The mathematical concept of a failure rate is easy to understand by the help of the reliability function. $R(t)$ of a system is the conditional probability that the system operates correctly throughout the time interval $< t_0, t >$ given that it was operating correctly at time t_0. The mathematical definition of $R(t)$ is based on probability that the system has survived the interval $< t_0, t >$.

Let M is number of identical systems placed in operation at time t_0. Let $M_1(t)$ be the number of systems that are operating correctly at time t and let $M_0(t)$ be the number of systems that that have failed at time t. It is assumed that:

1. $M_1(t_0) = M \quad thus \quad M_0(t_0) = 0$,
2. $\forall t_x \in < t_0, t >: \quad M_0(t_x) \le M_0(t) \wedge M_0(t_x) + M_1(t_x) = M$.

Then the reliability of the systems at time t is given by:

$$R(t) = \frac{M_1(t)}{M_1(t) + M_0(t)} = \frac{M_1(t)}{M} = 1 - \frac{M_0(t)}{M}. \qquad (4\text{-}1)$$

Time differentiate $R(t)$, we obtain

$$\frac{dR(t)}{dt} = -\frac{1}{M}\frac{dM_0(t)}{dt} \Rightarrow \frac{dM_0(t)}{dt} = -M\frac{dR(t)}{dt}. \qquad (4\text{-}2)$$

The derivate $\frac{dM_0(t)}{dt}$ is simply the instantaneous rate at which components are failing. The time curve of this rate is expressed by *failure rate function*:

$$\gamma(t) = \frac{1}{M_1(t)}\frac{dM_0(t)}{dt} \Rightarrow \gamma(t) = -\frac{1}{R(t)}\frac{dR(t)}{dt}. \qquad (4\text{-}3)$$

A common modelling techniques used to represent $\gamma(t)$ is Weibull distribution [Siew82]. The failure rate function associated with the Weibull distribution is given by:

$$\gamma(t) = \alpha\lambda(\lambda t)^{\alpha-1}, \qquad (4\text{-}4)$$

where α and λ are constants controlling the variation of the failure rate function with time. The reliability function that results from the Weibull distribution (4-4) is the solution to the differential equation (4-3):

$$\frac{dR(t)}{dt} = -\alpha\lambda(\lambda t)^{\alpha-1} R(t), \qquad (4\text{-}5)$$

and it is well known to be an exponential function of the above mentioned constants α and λ given by:

$$R(t) = e^{-(\lambda t)^{\alpha}}. \qquad (4\text{-}6)$$

The commonly accepted relationship between the failure rate function and time for electronic components is called the bathtub curve and is shown in Figure 4-2.

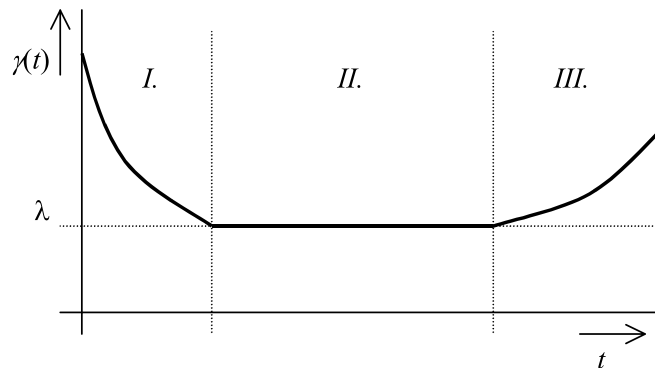

Figure 4-2: Bathtub curve

IT is possible to recognize three varied parts (I., II. and III.) in bathtub curve. The decreasing part of the bathtub curve (I.) is called *early-life* and during this period systems failures occur frequently due to substandard, imperfect or weak components. If we consider equation (4-4) then the coefficient α is less than 1, $\gamma(t)$ decreases as time increases.

The increasing part of the bathtub curve (III.) is called *wear-out* region where systems have been functional for a long period of time and are beginning to experience failures due to the physical wearing of components. If we consider equation (4-4) then the coefficient α is greater than 1, $\gamma(t)$ increases as time increases.

During the intermediate region (II.) the failure rate function is assumed to be a constant; this region of bathtub is called the *useful life* phase of the system. The failure rate function is assumed to have a value of λ during that period; λ is referred to as the failure rate and is normally expressed in units of failures per hour h^{-1}. This period of a constant failure rate is the most useful portion of the system's life. The coefficient in Weibull distribution function α is equal to 1 and the failure function is simply constant:

$$\gamma(t) = \lambda \,, \tag{4-7}$$

and the solution of the differential equation (4-3) is well known to be an exponential function of the parameter λ given by

$$R(t) = e^{-\lambda t} \,, \tag{4-8}$$

where λ is the constant failure rate. The exponential relationship between the reliability and time is known as the exponential failure law.

4.2. Software reliability model

The functionality of PLC operations becomes more essential and complicated and critical software applications increase in size and complexity. Pham writes that the reliability of the software not only becomes more important, but faults in software design become more subtle [Pham00]. Generally, software reliability is different form hardware reliability in the sense that software does not wear out; in fact, the software itself does not fail. The human interpretation and implementation of design goals can possibly cause a failure in system.

Software reliability models are used to describe the evolution of the software debugging process and to measure the quality of the software. There exist two types of software reliability: dynamic models and static models. Dynamic models follow the changes of the software throughout the entire testing period. Static models usually measure the quality of the software by taking one snapshot. This snapshot is taken either at the beginning of the debugging process to assist managers in planning or at the validation phase. Software reliability is defined to be the probability of failure free operation of a computer program in a specific environment for a specified period of time. The major expenses in a system are the penalty costs of software failures. A research studies [Musa80], [Lipo82], [Tria95] have shown that professional programmers average six software defects for every 1000 lines of code (LOC) written.

At this rate, typical automation software for medium automation task (a hundred I/O) can easily contain over 10 programming errors, see below.

Yu supposes a following form of the defect model of the software product [YuSh98]:

$$D = f(M, ED, T, other), \tag{4-9}$$

where

D	is the number of defects to be found a certain time period,
M	is static program metrics - such as lines of code, volume, etc.,
ED	is the defects found in the earlier stages of testing,
T	is the testing time in CPU time, calendar time, etc.,
other	stands for additional parameters such as programmer's skills, programming language, complex of the task, etc.

Usually, a static model assumes that some variable(s) influences the number of error placed in the software and that the influence may be described via regression analysis between the errors found in a software project and the variables that supposedly influence the errors. Such models include the equations proposed by Akiyama [Akiy71], which assume a linear relationship between errors and a number of software metrics, and the Compton model [Comp90], which assumes a quadratic relationship between errors and program length. The later defines defect density E_d:

$$E_d = \min(E_{d1}, E_{d2}), \tag{4-10}$$

where E_{d1} and E_{d2} are:

$$E_{d1} = 1.56 + \frac{69}{\hat{N}} + 4.7 \cdot 10^{-4}\,\hat{N}, \tag{4-11}$$

$$E_{d2} = 2.2351 + \frac{998}{\hat{N}} + 9.778 \cdot 10^{-5}\,\hat{N}, \tag{4-12}$$

and $\hat{N}$ is Halstead's estimated program length [Hals77], [Otte79]:

$$\hat{N} = n_1 \log_2(n_1) + n_2 \log_2(n_2), \tag{4-13}$$

where n_1 is a number of unique operators and n_2 is a number of unique operands.

The number of defects D (5-9) is then calculated as:

$$D = \frac{E_d \hat{N}}{1000} \quad . \tag{4-14}$$

Equation (4-10) is easy to modify for practical application into form:

$$E_d = \begin{cases} 1.56 + \dfrac{69}{\hat{N}} + 4.7 \cdot 10^{-4} \hat{N} & \hat{N} \in <1, N_0> \\ 2.2351 + \dfrac{998}{\hat{N}} + 9.778 \cdot 10^{-5} N & \hat{N} \in <N_0 + 1, \infty> \end{cases} , \tag{4-15}$$

where N_0 is a positive solution of the quadratic equation:

$$E_{d2} - E_{d1} = 0 \Rightarrow N_0 = 2728 , \tag{4-16}$$

because in the interval $\hat{N} \in <1, N_0>$ is $E_{d2} > E_{d1}$ and in the interval $\hat{N} \in <N_0 + 1, \infty>$ is $E_{d2} < E_{d1}$.

Lipow [Lipo82] used the Halstead theory to compute a series of equations of the form:

$$\frac{D}{\hat{N}} = A_0 + A_1 \ln \hat{N} + A_2 \ln^2 \hat{N} \quad , \tag{4-17}$$

where each of the A_i is dependent on the average number of usages of operators and operands per LOC for a particular language. For example, for Instruction List (IL), as EC61131-3 standard PLC controller programming language, $A_0 = 1.2 \cdot 10^{-3}; A_1 = 10^{-4}; A_2 = 2 \cdot 10^{-6}$.

Complexity of the automation task is possible to formulate as a function of number of inputs and outputs (NIO) in the system. The relationship between NIO and the deliverable lines of code in the PLC (DLOC) is an exponential function in the form:

$$DLOC = \alpha e^{\frac{NIO \cdot 10}{\alpha}} \quad , \tag{4-18}$$

where α is a relation coefficient between programming language and number of physical I/O. For IL (assembler) $\alpha = 310$. The dependence between NIO and DLOC is evident from Figure 4-3.

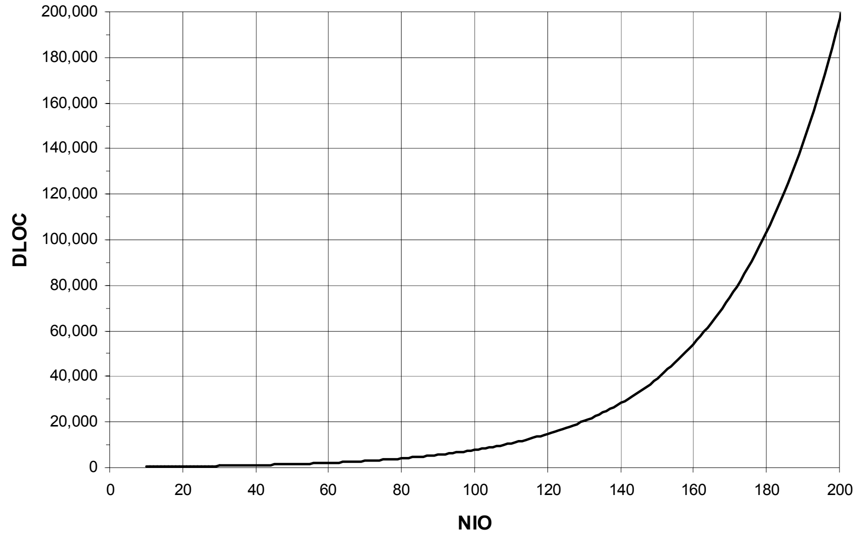

Figure 4-3: NIO vs. DLOC

For practical application, DLOC is recalculated into LOC. Function between DLOC and LOC is a power function in the form:

$$LOC = \gamma DLOC^{\gamma} \ , \qquad (4\text{-}19)$$

where parameter γ depends on implementation language; for IL (assembler) $\gamma = 0.96$. The dependence between DLOC and LOC is shown in Figure 4-4.

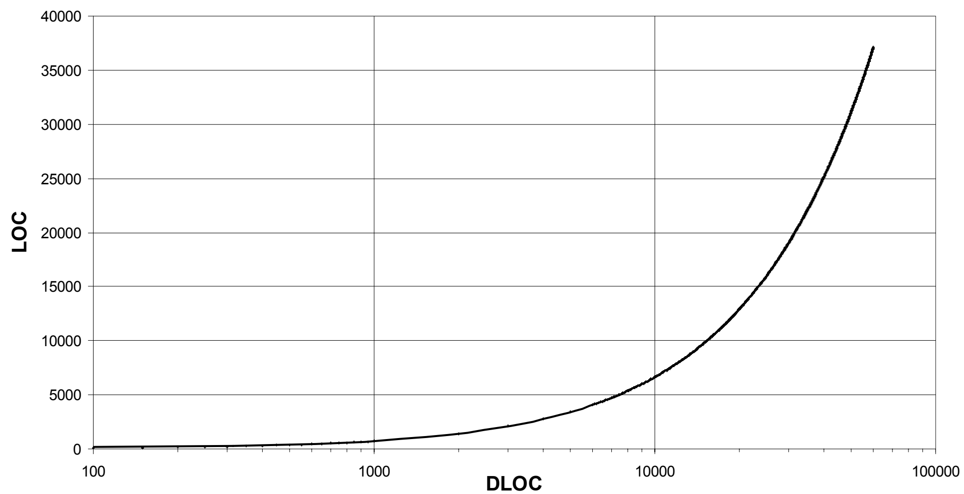

Figure 4-4: DLOC vs. LOC

If we consider equation (4-13) for Halstead's estimated program length, where coefficient n_1 (number of unique operators) is equal to size of instruction set of the PLC $n_1 = 100$ and n_2 (number of unique operands) is directly proportional to the NIO

$n_2 = NIO \cdot 4$, then it is possible to compare Lipow's model with Halstead's model; the result is shown in Figure 4-5.

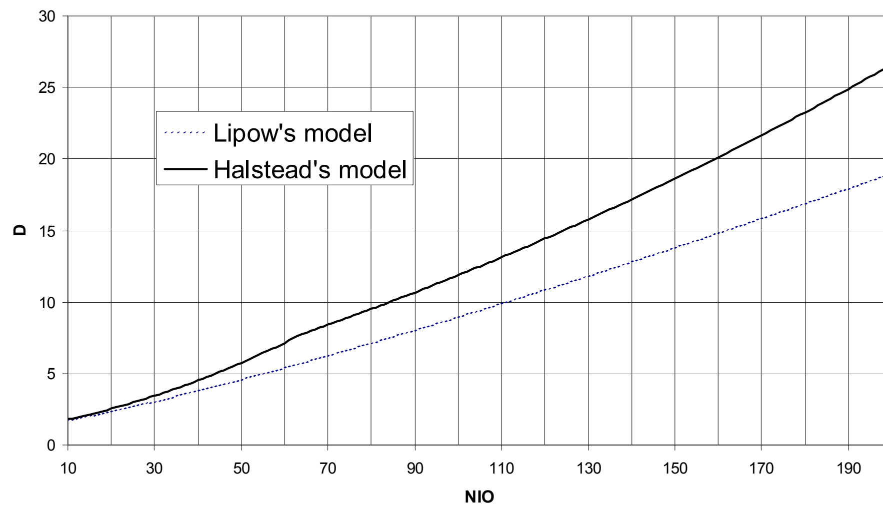

Figure 4-5: Comparison Lipow's model with Halstead's model

Halstead's model gives greater values of D; therefore it is used in the rest of the work.

Another important parameter of the software development is time of design T. The Software Science [Hals77] developed by M. H. Halstead principally attempts to estimate the programming effort. The countable properties are above-mentioned parameters n_1 and n_2 and estimated program length $\hat{N}$ (4-13). Halstead also defines Program Volume V as a metric for the size of any implementation of any algorithm:

$$V = 2 \cdot \hat{N} \cdot \log_2(n_1 + n_2) \ , \qquad (4\text{-}20)$$

and Program Level L:

$$L = \frac{2n_2}{n_1 \hat{N}} \ . \qquad (4\text{-}21)$$

Programming Effort E is restricted to the mental activity required to convert an existing algorithm to an actual implementation in a programming language:

$$E = \frac{V}{L} \ . \qquad (4\text{-}22)$$

Time of design is based on a concept concerning the processing rate of the human brain, developed by the psychologist John Stroud. Stroud defined a moment as the time required by the human brain to perform the most elementary discrimination. The Stroud

number S is then Stroud's moments per second, where $S \in <5,20>$. Thus we can derive the time equation for design time:

$$T = \frac{E}{S} \ [s]. \tag{4-23}$$

We have calculated number of defect D in the PLC's software equipment (4-14) and the Time of design T (4-23). However software development includes also debugging phase where number of defects is gradually decreasing. Debugging model is based on Software Reliability Growth Model (SRGM) using non-homogeneous Poisson process (NHPP) [Musa87]. Let $N(t), t \geq 0$ be a counting process representing the cumulative number of software failures by time t. The $N(t)$ process is shown to be a NHPP with a mean value function $m(t)$ representing the s-expected number of software failures by time t. Goel and Okumoto [Goel79] assume that the number of software failures during non-overlapped time intervals is s-independent and the software failure rate is proportional the residual fault content; thus $m(t)$ can be obtained by solving the following differential equation:

$$\lambda(t) = \frac{dm(t)}{dt} = b(a - m(t)), \tag{4-24}$$

where a denotes the initial number of faults contained in a program and b represents the fault detection rate. The result is:

$$m(t) = a(1 - e^{-bt}). \tag{4-25}$$

If we consider, that the debugging time T_d is one third of the design time T, the Stroud number $S = 18$ and fault detection rate $b = 0.03 \ h^{-1}$, then the trend of the failure rate function $\lambda(t)$ at the end of the debugging interval is shown in Figure 4-6.

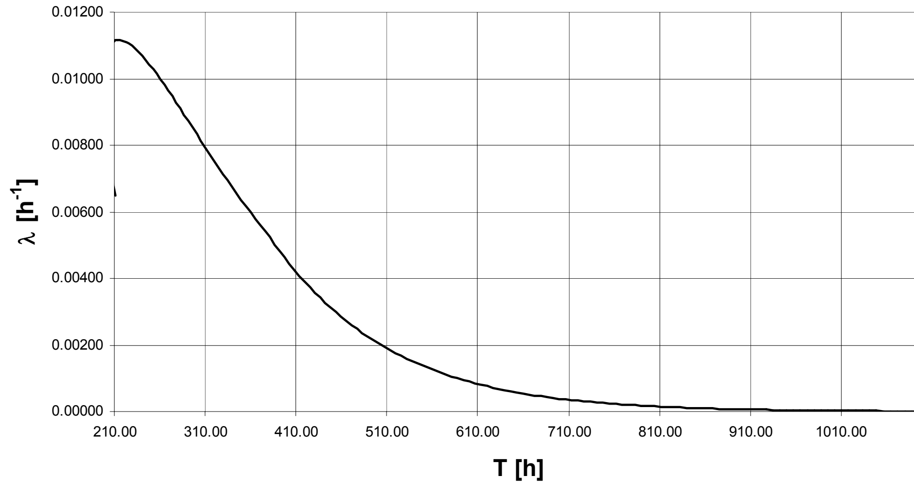

Figure 4-6: Failure rate function vs. design time

4.3. Reliability model for industrial buses

Number of necessary wires W for connection of the M control elements with the system (see Figure 4-1) is equal to

$$W = M(nI + 2nO), \tag{4-26}$$

where

M is number of control elements, $M \in \mathrm{N} - \{ 1 \}$,

nI is number of inputs signals, $nI \in \mathrm{N}_0$,

nO is number of outputs signals, $nO \in \mathrm{N}$.

Failure rate of such I/O system is equal to

$$\lambda_w = \lambda_0 \left[M\pi_m + W \int \pi_e(l)dl \right] \; [h^{-1}], \tag{4-27}$$

where

λ_0 is basic failure rate of the single connection, $\lambda_0 = 10^{-6}\, h^{-1}$,

M is above mentioned number of control elements, $M \in \mathrm{N} - \{ 1 \}$,

π_m is mechanical coefficient of the quality of the connection,

$$\pi_m = \pi_j \pi_v , \tag{4-28}$$

where

π_j is junction coefficient:

$\pi_j = 1.0$ for soldered connection,

$\pi_j = 1.5$ for plug-in connection,

$\pi_j = 2.0$ for screw connection,

π_v is coefficient of vibration:

$\pi_v = 0.0$ for laboratory environment with no vibrations,

$\pi_v = 1.0$ for stationary industrial mounts (switch-boards, static facilities ...),

$\pi_v = 2.0$ for non-stationary industrial mounts,

W is above defined number of wires.

π_e is a coefficient of electrical coupling [m^{-1}]; this coefficient determines electromagnetic influences of the environment.

Typical waveforms of this coefficient for two control elements (M=2) with mutual distance *L* are shown in Figure 4-7.

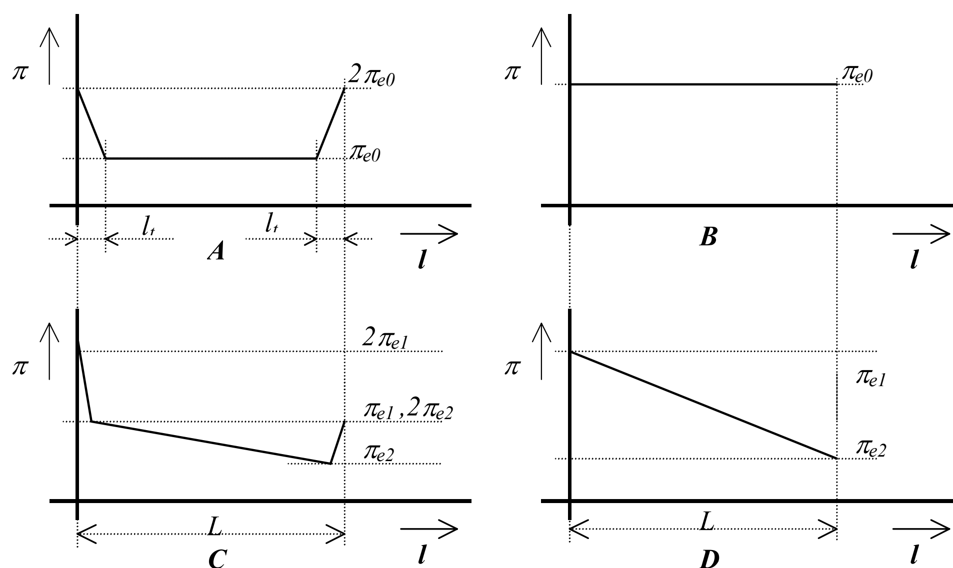

Figure 4-7:Typical waveforms of the electrical coefficient

Waveforms *A* and *B* represent situations, where both control elements are in the same area - from the EMC point of view. Waveform *A* determines situation where wire connection has a shielding. In this case the electrical coefficient π_e has constant value π_{e0} in the interval $l \in (l_t, L-2l_t)$ due to the fact, that EMC has the same effect on the shielded part of the wire connection. Non-shielded part of the wire connection in the interval $l \in <0, l_t> \cup <L-l_t, L>$ is called terminal length (l_t) and the electrical coefficient typically starts at the value $2\pi_{e0}$ at the zero distance from the control element and falls down to the value π_{e0} after terminal length l_t and vice versa for the other control element. Terminal boards are usually the weakness part of the I/O bus systems because they are usually not shielded in industrial environment and the waveform *A* reflects these weak-points. On the other hand, waveform *B* is typical for non-shielded I/O bus subsystem; therefore electrical coefficient has constant value π_{e0} in the whole interval $l \in (0, L)$. Typical values of this coefficient follow:

$\pi_{e0} = 0{,}1\, m^{-1}$ for laboratory environment or for industrial environment with none or minimal EMI,

$\pi_{e0} = 0{,}3\, m^{-1}$ for industrial environment with low EMI (low-power drives and switches),

$\pi_{e0} = 0{,}5\, m^{-1}$ for industrial environment with high EMI (high-power drives and switches),

Waveforms *C* and *D* represents situations, where both control elements are in the different area - from the EMI point of view. In this case different EMI effect on the I/O subsystem and the waveforms must respect it. In general, if we consider system with two different EMI shown in Figure 4-8.

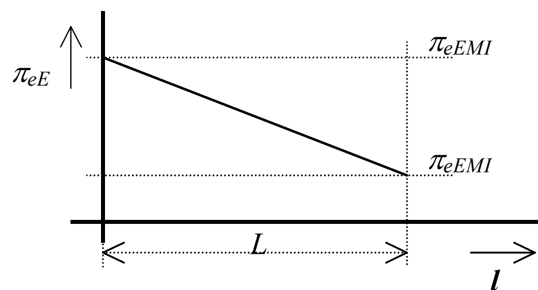

Figure 4-8: General electrical coefficient

Formally, waveform C is the Cartesian product between waveform A and the general electrical coefficient π_{eEMI}, thus

$$\Pi_e(C) \cong \Pi_e(A) \times \Pi_{eEMI} . \tag{4-29}$$

Reliability of the serial communication bus in one segment is easy to calculate from (4-27) if we set $M = W = 2$.

4.4. Reliability model of the M-redundancy structure

Reliability model of the control element CE (PLC) represents serial connection of the CPU module and SW module - Figure 4-9.

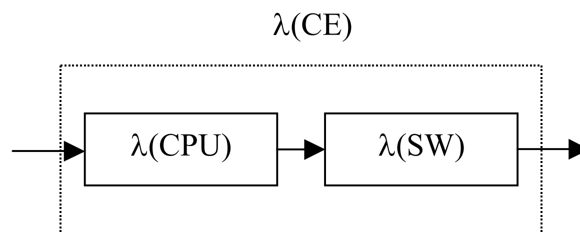

Figure 4-9: Reliability model of the control element

λ(CE) is failure rate of the control element. λ(CPU) is failure rate of the processor; producer of the PLC publishes this value. λ(SW) is failure rate of the software equipment - see chapter 4.2 for details.

Reliability model of the entire M-redundancy structure is shown in Figure 4-10.

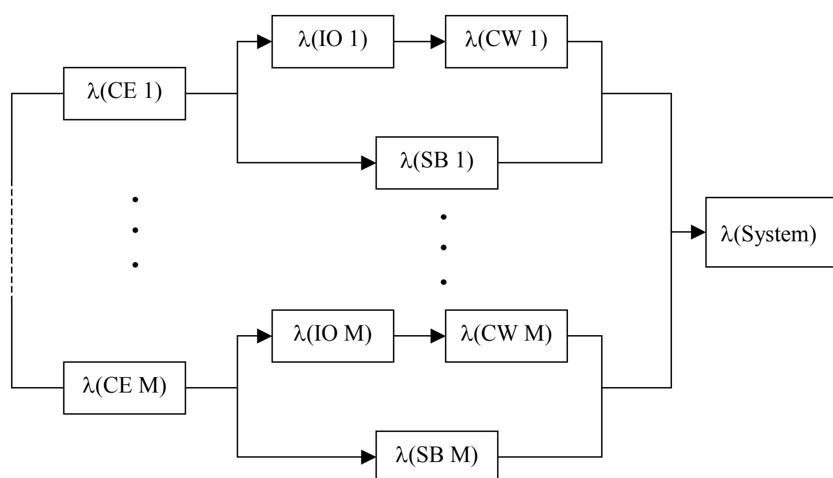

Figure 4-10: Reliability model of the M-redundancy structure

λ(IO) is failure rate of the IO subsystem of the appropriate control element; producer of the PLC publishes this value. λ(CW) and λ(SB) are failure rates of the buses - see chapter 4.3 for details. λ(System) represents failure rate of the physical system; this element should be implemented into model when its failure rate is known.

5. Formal verification

This chapter introduces the terminology used throughout the chapter 6: “Formal specification of the real-time communication system”. The chapter describes the notations used for temporal operators to make the formula more pregnant and legible. The safety and real-time conditions are described on a simple example. In addition UPPAAL implementation of temporal formulas is mentioned.

5.1. Temporal Logic

G. E. Hughes and M. J. Cresswell describe temporal logic as an evolved branch of modal logic [Hugh68]. Thorough description of temporal logic is in [Resc71], for example. Temporal logic has been proposed as applying both to the specification and verification of program behaviour, and to the specification of system behaviour. This work deals with verification of program behaviour and also system behaviour.

The first significant effort for using temporal logic to describe program behaviour came from Pnueli and his later paper [Pnue85] is a good and unique survey of the field. Other good general descriptions of the applicability of temporal logic are [Lamp86] and [Lamp83]. There have been numerous papers from Clarke and others demonstrating this applicability; [Clar86] and [Clar87] describe the development of tools to automate the process, what is the inestimable benefit for industrial communication systems. The verification of a system in temporal logic is usually divided into safety conditions, liveness conditions and fairness conditions.

The safety conditions are those conditions that have to be established at the different levels of the system hierarchy to confirm that safety (i.e. absence of hazards) is maintained down the hierarchy, so-called vertical checks. At the topmost level it must be confirmed that the hazards identified for each accident capture all of the system conditions that could lead to that accident. The safety constraints must be validated against the hazards. At subsequent levels, the safety specifications are verified against those that immediately precede it in the hierarchy. For example, each safety strategy

must be verified against the relevant safety constraint. In other words, the safety conditions are those, which must not occur in operation of the system. For example, in N-Redundancy system the situation when two control elements control task at the same time must not occur.

The liveness conditions specify what the have to do. For example, in N-Redundancy system when a control element is failed, the spare control element has to take a control.

The fairness conditions specify how nondeterministic conditions are to be resolved. For example, in N-Redundancy system where $N = 3$, the fairness condition could express what is the safe state in the system.

Temporal logic allows the system operation to be defined nondeterministically, and to include fairness conditions such that one course of action does not necessarily dominate. In general, it is best to specify a system nondeterministically, since this is the most general specification and postpones the operational details until later in the development process. Often the advantage in thus postponing such a decision is that there may be a distinct benefit of one way over another - i.e. deterministically over nondetereministically, which does not become obvious until later in the life cycle.

5.2. Temporal operators

5.2.1. Logical operators

Table 5-1 shows basic logical operators and their symbols used.

Table 5-1: Logical operators

Logical function	*Logical operator*
and	$\wedge$
or	$\vee$
not	$\neg$
equivalence	$=$
implies	$\rightarrow$

5.2.2. Temporal operators

Table 5-2 shows the temporal operators to be used.

Table 5-2: Temporal operators

Operator	***Function***
$<> a$	eventually *a* will be true
$[]a$	henceforth *a* is true
$\circ a$	the next state *a* (instant in time) will be true
$\bullet a$	the previous state *a* (instant in time) will be true
$a \bigcup b$	*a* is true until the state (instant in time) when *b* occurs; the strong until implies that *b* will eventually occur. In addition, *a* must always be true in the present.
$a = b$	*a* is strictly equivalent to *b*

Temporal logic is a modal logic; each temporal formula (query) is interpreted either as true or false against a model of the real system. System is modelled as a state machine and to check the validity of a formula against a state machine is a matter similar to that of [Clar87].

The state machine *SM* is defined over a set of atomic propositions AP (logical variables), and has the States (*S*), Relationships (*R*) and Truth (*T*). Thus:

$$SM = (S, R, T) \tag{5-1}$$

where

S is a finite set of states, $S == \{s_0, s_1, s_2, ...\}$. For example, the set of possible states for IOBusState is (see chapter 6.2.9 for details):

$$IOBusStates == \{Ready, Read, Stop, Write\} \tag{5-2}$$

R is a binary relationship on *S*, defining the possible transition(s) between states. If there is a direct path between two different states s_i and s_j then:

$$(s_i, s_j) \in R \tag{5-3}$$

If there is no such relationship in *R*, then there is no arc connecting the two states. Every state must be connected to at least one other state. For example, in Figure 6-7: Timed-automaton of the IO bus, the following relationship is a direct path:

$$(Ready, Read) \in R_1 \tag{5-4}$$

T assigns to each state the set of atomic propositions true in that state.

A path $\pi(s_i)$ is infinite sequence of states starting at the state s_i. $\pi(s_0) = (s_0, s_1, s_2, ...)$ such that each adjacent pair of states (s_i, s_{i+1}) is in the relationship **R**:

$$\forall i, i \geq 0, (s_i, s_{i+1}) \in R \tag{5-5}$$

5.3. UPPAAL

UPPAAL is an integrated tool environment for modelling, simulation and verification of real-time systems, developed jointly by BRICS at Aalborg University and the Department of Computer systems at Uppsala University [Lars98]. It is the appropriate tool for system that can be modelled as a collection of non-deterministic processes with finite control structure and real-valued clocks, communicating through channels or shared variables. Typical application areas include real-time controllers and communication protocols in particular, those where timing aspects are critical.

UPPAAL consists of three main parts: a description language, a simulator and a model-checker. The description language is a nondeterministic guarded command language with simple data types (unbounded integers, arrays, etc.). It serves as a modelling or design language to describe system behaviour as networks of automata extended with clock and data variables. The simulator is validation tool that enables examination of possible dynamic executions of a system during early design (or modelling) stages ant thus provides an inexpensive mean of fault detection prior to verification by the model-checker, which covers the exhaustive dynamic behaviour of the system. The model-checker is to check invariant and bounded-liveness properties by exploring the symbolic state-space of the system, i.e., reachability analysis in terms of symbolic states represented by constraints.

The theory of timed automaton is well described in [Alur94]. A number of verification tools have been developed for timed systems in the past year. UPPAAL is one of them as is described in [Beng95], [Hune00], [Behr01] or [Lars98]. The other tools are well described in [Yovi97].

The goal of UPPAAL has always been to serve as a platform for the tool to provide a flexible architecture that allows experimentation. it should allow orthogonal features to be integrated in an orthogonal manner to evaluate various techniques within a single framework and investigate how influence each other [David03].

5.3.1. Mutual Exclusion Algorithm check example

A mutual exclusion algorithm that ingeniously exploits real time has proposed by M Fischer. Mutual exclusion algorithm is good example where is easy to see how we can derive an automaton model from a higher programmable language and check properties related to it. The algorithm (i.e. informal description) for N-processes is as follows in C++:

```
Class Process {
  int ID;
public:
 Process(int sID);       //Constructor
    ~Process();        //Destructor
  bool request, turn;
    void Job(void);
  void Run(void);
};

Process::Process(int sID) {
 ID=sID;
}

void Process::Run(void) {
    request = true;
  turn = true;
  job();
  req = false;
}

Process::~Process() {
 cout << "Process " << ID << " destroyed!" << endl;
 printsize();
}

const int N=10;      //Number of processes
Process *P[N];

int main() {
  for (int i=0; i < N; i++) {
```

```
      P[i]=new Process(i);
      P[i].Run();
    }
  }
```

There are N processes, numbered 1 through N, and a global variable n that assumes an integer value between 0 and N. Figure 5-1 shows the state transitions of process $i, 1 \leq i \leq N$. A process transits from E to A to wait for entry to its critical section. The mutual exclusion (*ME*) state diagram is defined as:

$$ME == \{\mathrm{A}, \mathrm{B}, \Gamma, \Delta, \mathrm{E}\} \tag{5-6}$$

and its graphical formulation is shown in above-mentioned Figure 5-1.

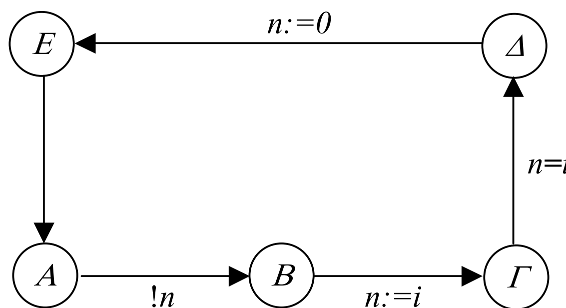

Figure 5-1: State diagram of the ME

Process state is Δ when it is in the critical section. Assume that all tests and assignments are atomic. Initially, all processes are in states E and $n = 0$. It is easy to imagine a scenario where two arbitrary processes are in their critical sections simultaneously. Following timing constraints (TC1 and TC2), ensures that this possibility is avoided.

TC1: Transition from B to Γ is completed within a unit of time. This transition only requires assigning a value to *n*, and, therefore, the transition is entirely within the control of a process.

TC2: Transition from Γ to Δ takes more than one unit of time. This requirement is implemented by process i waiting for more than a unit of time before testing *n=i*.

Let ME_i is the state of the i_{th} process. From the equation 5-6 is evident that:

$$ME_i == \{\mathrm{A}, \mathrm{B}, \Gamma, \Delta, \mathrm{E}\} \tag{5-7}$$

The initial state of any of the systems is:

$$\left(\forall i \ :: \ ME_i = \mathrm{E}\right) \ \wedge \ n = 0 \tag{5-8}$$

The state transitions for each i_{th} process are:

$$\begin{aligned}
&\{\alpha_i\}\ ME_i := \mathrm{A} \ \ if \ \ ME_i = \mathrm{E} \\
&\{\beta_i\}\ ME_i := \mathrm{B} \ \ if \ \ ME_i = \mathrm{A} \wedge n = 0 \\
&\{\gamma_i\}\ ME_i := \Gamma, n := i \ \ if \ \ ME_i = \mathrm{B} \\
&\{\delta_i\}\ ME_i := \Delta \ \ if \ \ ME_i = \Gamma \wedge n = i \\
&\{\varepsilon_i\}\ ME_i := \mathrm{E}, n := 0 \ \ if \ \ ME_i = \Delta
\end{aligned} \tag{5-9}$$

The predicates are mutually exclusive, thus:

$$\begin{aligned}
&\mathrm{A}_i \wedge \mathrm{B}_i = false \\
&\mathrm{B}_i \wedge \Gamma_i = false \\
&etc.
\end{aligned} \tag{5-10}$$

And logically:

$$\mathrm{A}_i \wedge \mathrm{B}_i \wedge \Gamma_i \wedge \Delta_i \wedge \mathrm{E}_i = true \tag{5-11}$$

A variable t_n holds the current time at any point during the computation. Predicate $p(t)$ holds after this assignment if $p(t_n)$ holds before. Two presumptions are required for variable t_n:

1. $t_n \in \Re^+$

2. t_n is monotone nondecreasing

then three new variables $\overline{\mathrm{B}_i}, \overline{\Gamma_i}, \overline{\Delta_i}$ are defined as:

$$\begin{aligned}
&\{\beta_i\}\ ME_i := \mathrm{B}, \overline{\mathrm{B}_i} := t_n \ \ if \ \ ME_i = \mathrm{A} \wedge n = 0 \\
&\{\gamma_i\}\ ME_i := \Gamma, n := i, \overline{\Gamma_i} := t_n \ \ if \ \ ME_i = \mathrm{B} \\
&\{\delta_i\}\ ME_i := \Delta, \overline{\Delta_i} := t_n \ \ if \ \ ME_i = \Gamma \wedge n = i
\end{aligned} \tag{5-12}$$

Above mentioned timing constraints TC1 and TC2 are formally now:

$$\begin{array}{llll} \text{TC1}: & (\Gamma_i \vee \Delta_i) & \Rightarrow & \overline{\Gamma_i} \leq 1 + \overline{\mathrm{B}_i} \\ \text{TC2}: & \Delta_i & \Rightarrow & 1 + \overline{\Gamma_i} < \overline{\Delta_i} \end{array} \qquad (5\text{-}13)$$

Let following variables l, m satisfy condition $1 \leq l, m \leq N$.

$$\begin{array}{llllll} \text{S1}: & (\forall l, m & :: & n = m & \Rightarrow & \overline{\mathrm{B}_l} \leq \overline{\Gamma_m}) \\ \text{S2}: & (\forall m & :: & \Delta_m & \Rightarrow & n = m) \end{array} \qquad (5\text{-}14)$$

Mutual exclusion is immediate from S2:

$$\Delta_i \wedge \Delta_l \;\Rightarrow\; n = i \wedge n = l \;\Rightarrow\; i = l \qquad (5\text{-}15)$$

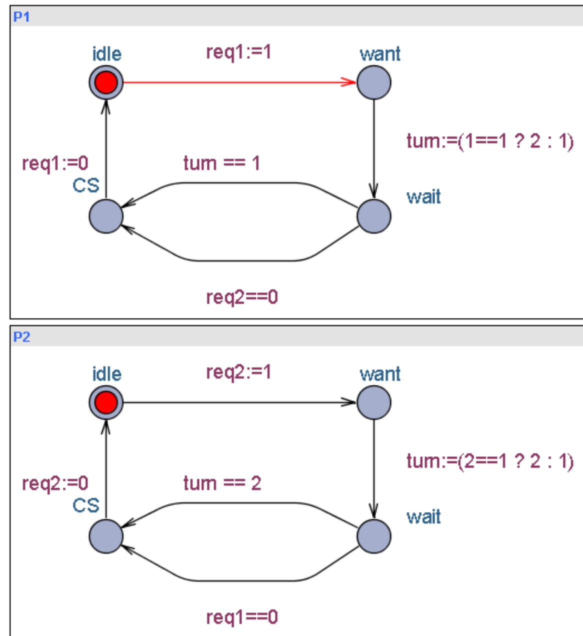

Figure 5-2: Mutex processes

If we consider two processes P1 and P2 modelled in UPPAAL and showed in Figure 5-2 (CS means critical section), then the mutual exclusion property is easy to verify in the UPPAAL verifier by query:

A[] not (P1.CS and P2.CS)

what is the UPPAAL notation for the temporal logic formula:

$$\forall[]\neg(\text{P1.CS} \wedge \text{P2.CS})$$

The answer of the verifier is, of course, *true*.

6. Formal specification of the real-time communication system

This chapter describes basic developed methodology of the formal description, analysis and implementation of the real-time industrial communication system. All used symbols and equations are explained in the previous chapter 4 and 5.

6.1. Project's management

The question of the real-time communication is crucial for the industrial communication systems; this fact was discussed in the previous chapter 3.1.4. When we are dealing with real-time industrial communication system then restricted software and hardware sources are available, as was also discussed before. Heterogeneous structure of such system requires new management of the project. This management should include control mechanisms based on formal methods used during development of the project as well as suitable final implementation tools. The flowchart of such project's management is shown in Figure 6-1.

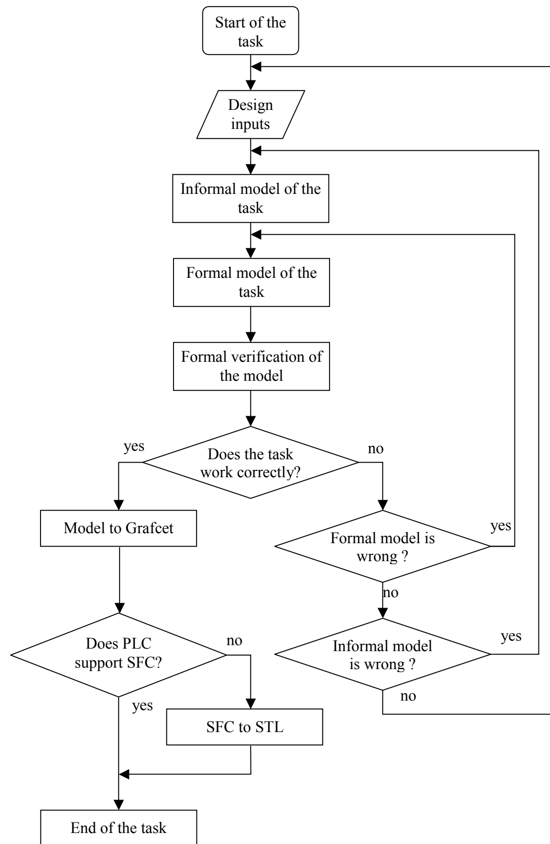

Figure 6-1: Flowchart of the project's management

The flowchart consists of several important steps. *State of the task* represents formal beginning of the project design and it is followed by step called *Design inputs*. This step is easy to understand by the inquiry: "What we must to do ?" The answer to this question is also first specification of the task; inputs from several engineering branches are collected and transformed into uniform utterance. Step *Design inputs* is followed by step *Informal model of the task;* in this step an informal model of the task is created. The unexceptional design of the task is finished in this step because an informal model is usually machine interpretation of the design inputs; computer program (indifferently to the platform) or flowchart are examples of such informal model of the task. In case of the formal design, the formal model must be created and verified. Project's management solves this issues in steps *Formal model of the task* and *Formal verification of the model*. After the verification of the model first decision replies the question: „Does the task work correctly?" In case that all conditions, specified during collection design inputs, are satisfied, the model of the task is transferred into SFC as an international standard for programming languages for PLCs. In case, that controller(s), used in the application, does (do) not support this standard, the transfer of the program

from SFC to the LAD or STL is performed - see chapter 9 (Appendix A) for details. In case that all conditions are not satisfied, it is necessary to observe what part of the project is wrong. In this case, there exist three possibilities. First, the formal model is wrong - for instance, deadlock or dead states are founded during simulation/verification; we have to return into step *Formal model of the task*. Second, the informal model is wrong - for instance, the behaviour of the task does not correspond with the claims; we have to return into step *Informal model of the task*. Third, the both above mentioned problems do not occur - then we have to return into step *Design inputs*.

In an simplified way, the entire sequence is possible to define in the following three steps:

1. To model a critical part of the communication system using timed-automaton (UPPAAL) - *Formal model of the task.*
2. To simulate it and then verify properties on it (i.e. functionality and operability) - *Formal verification of the task.*
3. To transfer timed-automaton model into appropriate language of the PLC – SFC.

All these steps are explained on the example of the real-time communication automation task.

6.2. FT control of the railway

A physical system (railway) is represented by a railway track. This railway track is situated in Laboratory of Programmable Logic Controllers and process Control System [TLPK02], [LVPK02] or [Zezu02]. Simplified schematic diagram is shown in Figure 6-2. I/O subsystem, control algorithm, timed-automaton model, Grafcet presentation and reliability model are described below.

6.2.1. Inputs

Inputs are separated into two groups.

First group integrates 10 independent switches. They enable to change the way of movement of the train. Switches are symbolized by the rectangles in the schematic diagram.

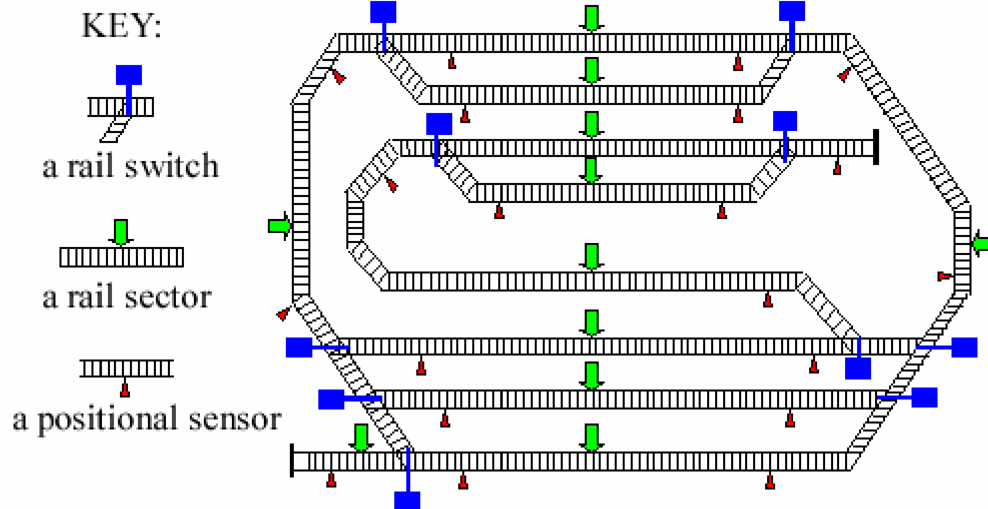

Figure 6-2: Schematic diagram of the railway track

Second group integrates 11 sectors. There is possible to switch off the power supply (i.e. stop the train) and to change the train direction inside each sector. While switching off the sectors is independent for each sector, changing direction depends upon state of the rest sectors. In fact sectors are separated into two groups and inside each group all sectors have the same direction at the same time. The arrows in the schematic diagram symbolize sectors.

6.2.2. Outputs

There is only one kind of output in model, the fast-action sensor. These fast-action sensors detect the present of the train at the location of the sensor. If, for example, the train drives through the sensor, the output of the sensor is a pulse or series of pulses. Model integrates 20 of them and they are symbolized by the small triangles in diagram.

All I/O elements are centralized in an interface module. The interface module ensures basic signal operation like galvanic protection, voltage unification and amplification. Also it protects model against unsuitable actions as a change switch

while the train is driving through or stop on it. It results from this description that interface module is the basic element that protects railway model against faults rising on the signal way from control elements to the model.

The railway model together with the interface module, are excellent examples of the failing systems. For example, the failure rate switch of the rail is $\lambda = 0,8$ (failures/hour). The reliability of the entire model (including the interface) is $R(1h) = 0,68$ (value is taken from the experiment). This parameter is an essential reliability of the model not the control system. The model is only marginal part of our problem in fact. It serves just like a computer screen, where we can see results of our control. Model is only described in purpose to introduce reader into the problematic; detailed information reader can find in [Kuce01].

6.2.3. Control Structure

As was discussed in chapter 0, control elements are absolutely based on PLCs. In our case two are used: S7-300 and S7-200, both are Siemens's products. The MTBF for these PLC is approximately $30x10^3$ hours, IO subsystem has the MTBF $30x10^4$ hours. How you can see, the reliability of these control elements is high and it is more than sufficient for many industrial applications. It raises question why use two of them, if one of them already has sufficient reliability? It is because of the external influence. The reliability of PLC itself is high but if it is placed into the plant, thousand of external influences take effect. And these influences could not be take into account when MTBF was calculated or measured because we don not know them or we can't measure them. To use more than one control element is the basic concept of the redundancy when creating FT system and this concept is called hardware redundancy. In our control design are used two PLC and each has unique algorithm of control and additional software to detect faults. This concept is called software redundancy and it is one of the key elements in our design [Aviz84].

Basic problem of the hardware redundancy concept is how to connect two (or more) control elements into one system (model). This is not only question of the physical connection of I/O of the control elements but also the question how these elements will react when the fault is occurred. At the same moment only one element is allowed to control the system; the second element is back-up element. In case of failing

of control, the back-up control element must recognise it, immediately react and override faulty control element and safely continue to control the task or safely terminate control task. This form of hardware redundancy is called active and the problem is, how to detect the existence of a fault. If 2 control elements are used and theirs I/Os are connected parallel to the system, both have the same input information (input vector). Then active element puts its calculated output values (output vector) into I/O bus and passive element reads this output vector using a common I/O bus and compares it with its own calculated output vector as is shown in Figure 6-3.

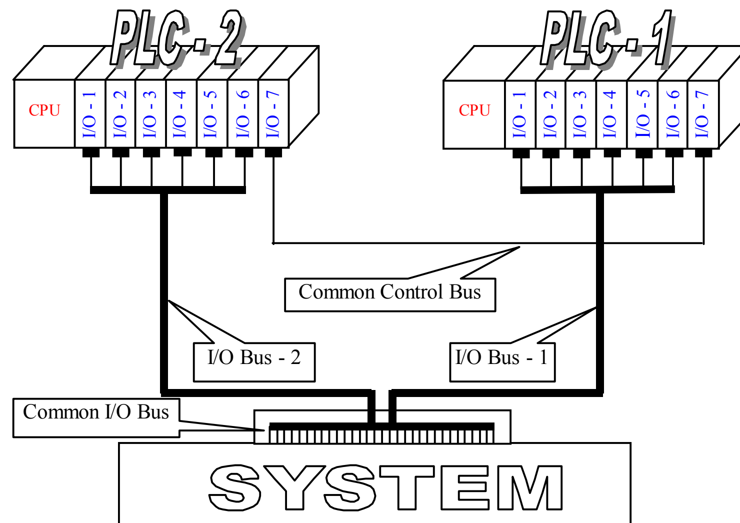

Figure 6-3: Hardware connection of PLCs

If a passive control element detects variance between its calculated output vector and real output vector on bus, then variance signal is put in common control bus and control algorithm of active element must react. Such form of control elements is able to cover following faults:

- CPU error
- Algorithm error
- I/O subsystem error
- I/O bus error

Advantage of this FT control design is simplicity of the software. Disadvantage is number of wires. If we consider equation (4-26) where

M=2, nI=20, nO=21

then number of wires is:

$$W = 2(20 + 2 \cdot 21) = 124\,, \tag{6-1}$$

because PLCs must read output vector from the common bus. Increasing number of wires in system decreases its reliability; our aim is opposed. If we consider equation (4-27), mechanical coefficient for laboratory environment $\pi_m = 0$ (no vibration), number of control elements $M = 2$, distance between them is $L = 4\,m$, number of wires $W = 124$, and constant electrical coefficient $\pi_{e0} = 0{,}1\,m^{-1}$ (bus is not shielded), then failure rate of the I/O bus system is equal to:

$$\lambda_w = 10^{-6} \cdot 124 \cdot 0{,}1 \cdot 4 = 4{,}96 \cdot 10^{-4}\,h^{-1}\,.$$

DLOC for software equipment in each PLC is calculated from equation (5-18), where $NIO = nI + nO = 41$ and $\alpha = 310$; $DLOC = 1163$ lines and LOC from equation (4-19), where $\gamma = 0.96$; $LOC = 842$ lines of IL code. Number of defects D is calculated step by step from equations (4-13), (4-15) and (4-14); $D = 5$. Design time T is calculated from equations (4-20), (4-21), (4-22) and (4-23), where the Stroud's number $S = 18$; $T = 11$ days. And finally, the failure rate of the software λ_{SW} is calculated form equations (4-24) and (4-25), where fault detection rate $b = 0.03\,h^{-1}$ and debugging time $T_d = T/2$;

$$\lambda_{SW} = 0.00261\,h^{-1}\,. \tag{6-2}$$

Failure rate of the CPU is $\lambda_{CPU} = 3.33 \cdot 10^{-5}\,h^{-1}$ and failure rate of the IO subsystem is $\lambda_{IO} = 3.33 \cdot 10^{-6}\,h^{-1}$. Therefore, reliability of two parallel connection of the PLC (with failure rate $\lambda_{CE} = \lambda_{CPU} + \lambda_{SW} + \lambda_{IO} = 0.002647$) and I/O bus system per 100 hours of operation is:

$$R(100h) = 0.9\,. \tag{6-3}$$

6.2.4. Industrial communication network

Using industrial network in our problem is a great asset. It allows use more complex control algorithm and to connect new elements into system increasing the reliability. Profibus DP network was selected due to the fact both PLCs are easy connected to Profibus (both PLCs have Profibus DP communication processor). The

advantage of industrial network is in possibility of distributing various data between communication partners. Also industrial network simplifies the connection of another control elements that increasing reliability of control. In our case, it is decentralised I/O system (WAGO). All these elements were connected into structure in Figure 6-4. Dashed lines represent Profibus DP connection and solid lines are conventional I/O signals.

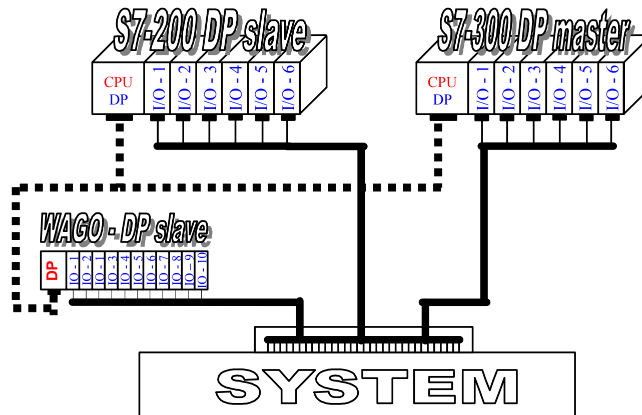

Figure 6-4: Architecture of the system with Profibus DP

In our control system Profibus DP network ensures following functions:

- It observes all connected nodes and provides useful information about the state of the system to the control algorithm. The useful information is for example what nodes are working correct or what nodes have wrong input information.
- It allows control the system using decentralised I/O terminal WAGO.
- It allows dynamically change the active control element (S7-200, S7-300, PC-5412).

Data transfer is possible only between master and its slave or between two masters. There are two masters (S7-300 and 5412 card in PC) and two slaves (S7-200 and Wago) in Figure 6-4. Only S7-300 and PC-5412 masters can control task, using I/O subsystem of their slaves.

6.2.5. Control algorithm

All possibilities of the control system structure are described in Table 6-1. For example row number 6 describes situation when PLC S7-300 controls task using the input values taken from S7-300 input card and using output card of Wago. Such

configuration accepts total error of S7-200 PLC (total error means something like PLC is not present in system), accepts error in output subsystem of S7-300 and error in input subsystem of Wago. Profibus error is not allowed here because of data exchange between S7-300 and Wago.

Table 6-1: List of possible reconfigurations

No.	Control element	Input vector from			Output vector into			Error coverage											Comment
								S7 – 300				S7-200				Wago			
		S7-300	S7-200	Wago	S7-300	S7-200	Wago	CPU	DP	OSu	ISu	CPU	DP	OSu	ISu	OSu	ISu	Profibus	
1	S7-300	X			X				X			X	X	X	X	X	X	X	Profibus e.
2	S7-300		X		X						X			X		X	X		Wago error
3	S7-300	X				X				X					X	X	X		Wago error
4	S7-300		X			X				X	X					X	X		Wago error
5	S7-300			X	X						X	X	X	X	X	X			S7-200 error
6	S7-300	X					X			X		X	X	X	X		X		S7-200 error
7	S7-300			X			X			X	X	X	X	X	X				S7-200 error
8	S7-300		X				X			X	X			X			X		
9	S7-300			X		X				X	X				X	X			
10	S7-200		X			X		X	X	X	X		X			X	X	X	Profibus e.
11	S7-200		X		X						X			X		X	X		Wago error
12	S7-200	X				X				X					X	X	X		Wago error
13	S7-200	X			X									X	X	X	X		Wago error
14	S7-200		X				X			X	X			X			X		
15	S7-200			X		X				X	X				X	X			
16	S7-200			X			X			X	X			X	X				
17	S7-200			X	X						X			X	X	X			
18	S7-200	X					X			X				X	X		X		

Comments**:**

Control element is the place where the control algorithm is running.

Input vector from and *Output vector into*

Inform what input values are used by the control algorithm and where the control algorithm writes output values.

Error coverage is an acceptable combination of errors in the system.

CPU is error in CPU of PLC; i.e. this PLC can't execute the control algorithm.

DP is error in the communication processor in the appropriate PLC; i.e. this PLC can't communicate via Profibus, but the PLC is still able execute the control algorithm.

OSu means Output Subsystem - i.e. any error occurred on output card of PLC or Wago, or error occurred on wire of I/O bus (see Figure 6-3)

ISu means Input Subsystem - i.e. any error occurred on input card of PLC or Wago, or error occurred on wire of I/O bus (see Figure 6-3)

Profibus is Profibus error (e.g. disconnection of the bus)

6.2.6. Error detection

The most important problem is how to detect a fault. At first we must explain that there are two basic sorts of faults:

I/O error

It means that control algorithm is operating correctly, but due to the external effects the algorithm detects wrong input values or its output values are not properly interpreted. Typical example of such effect is logical stuck-fault on the bus [Haye85]. In such case the control algorithm must recognise what part of I/O subsystem is faulty and must reconfigure system such a way so the influence of the error part of the I/O subsystem will be eliminated.

Calculating error

It means that control algorithm in active PLC is wrong operating. In such case the passive PLC must recognise it and override the control of the task.

To detect I/O error is in the system using industrial network is simple thing. We have 3 source of the I/O information: S7-300, S7-200 and Wago. Using simple form of passive redundancy TMR control algorithm can decide what part of the I/O subsystem is faulty. In our case standard TMR is modified to TMR version with duplicated voters, see Figure 6-5. Thus we can also decide if the voter fails - we can't decide what it is, but we can perform some action, e.g. to terminate task or to inform supervisor system.

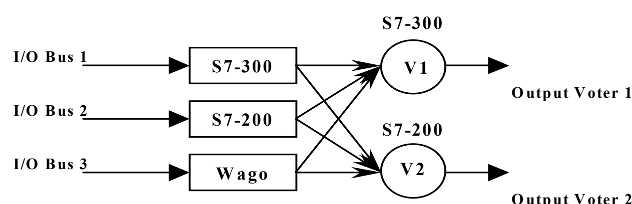

Figure 6-5: TMR with duplicated voters

6.2.7. I/O error

There is a flowchart of the control algorithm detecting I/O error in Figure 6-6. First, all I/O vectors are received through Profibus network. IOT means Input Output Table and it is the information about the state of the vector on appropriate I/O bus. Then, the correct state (i.e. the state with the highest probability) on the bus is calculated, using TMR method. If any of I/O vectors (I/O subsystem) is invalid, then is marked as faulty and is ignored for further use.

Concurrent voting result is received V2 (from the passive element) and it is compared with voting result of the active element V1. If disagreement is occurred (probably faulty CPU or network communication) the task is safely terminated. In other case, control structure of the task is reconfigured according to Table 6-1.

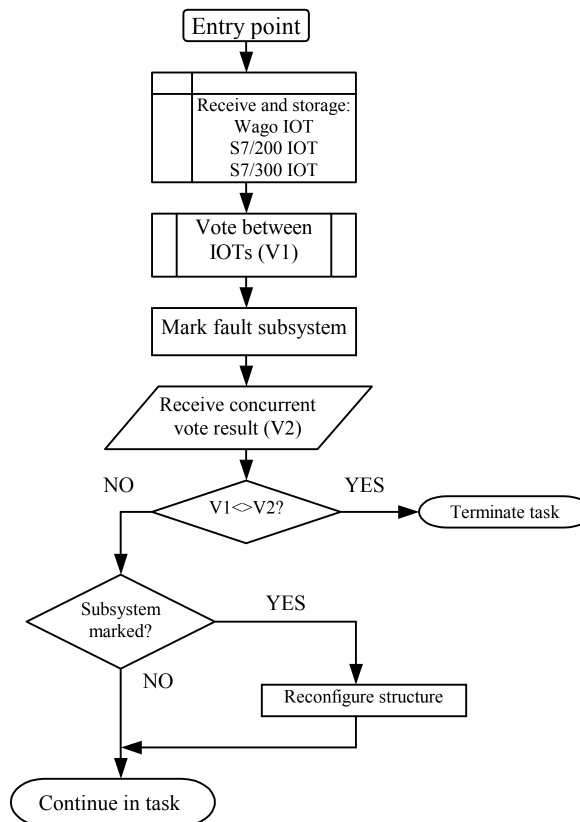

Figure 6-6: Flowchart of the control Algorithm

6.2.8. Calculating error

Detecting of calculating error is based on watching the active element by the passive element - this technique is often called backup with model. This watching is

either based on redundancy of algorithm in passive elements, i.e. both PLC calculate concurrently action values, but only active element controls task, or the watching is based on temporal look-up table. This table is in passive element and holds information about action values in time. Temporal look-up table is progressive method of the redundancy in end-state task, like is the railway model. Two basic methods were developed during creating of the FT model of control. First method is based on teaching. It means that during one cycle of task passive element is filing its temporal loo-up table by values of the active element. Second method consists in sending this temporal loo-up table from supervisor system (active element, 5412 card) through Profibus network to the passive element. In case of disagree between action values, the passive element overrides the control and either terminates the task safely or continue in operating.

6.2.9. Formal description of the control algorithm

As was mentioned at the beginning of the chapter, the timed-automaton methodology is used to model a critical part of the control algorithm. It si obvious that the critical part of the control algorithm is the information exchange between all communication partners. This information exchange is easy to describe by an event chain:

Receiving & storing IOTs→Voting V1→Sending V1→Receiving V2→Reconfiguring.

Control algorithm has to assure completion of this chain in finite and previous defined time in case of any of the conceivable error.

Timed-automaton of the system is into three different parts:

1. Timed automaton of the IOBus - Figure 6-7.
2. Timed automaton of the Profibus DP - Figure 6-9.
3. Timed automaton of the control algorithm - Figure 6-11.

Timed-automaton model of the IO bus is shown in Figure 6-7. Model has four states:

$$IOBusStates == \{Ready, Read, Stop, Write\}$$
$$IOBus : IOBusStates$$
$$time : R$$

$$time \geq 0$$
$$IOBus \in \{Read, Write\} \Rightarrow time = 0$$
$$IOBus = Stop \wedge \uparrow CPUReset \Rightarrow IOBus = Ready$$

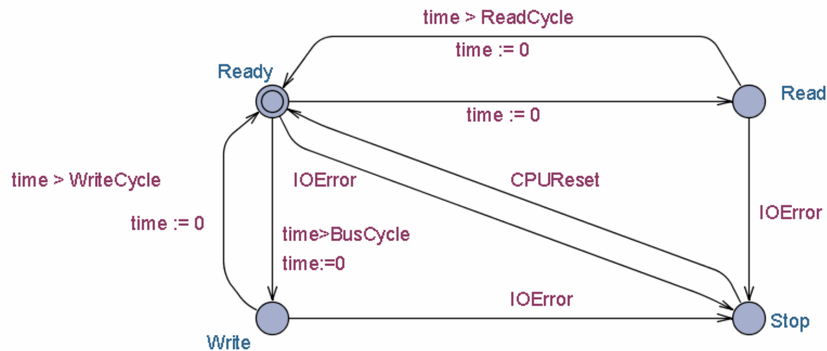

Figure 6-7: Timed-automaton of the IO bus

Local quantifier *time* determines temporal behaviour of the IO bus system. At the beginning the system is in the *Ready* state and if no errors are presented in the system, then immediately enters he *Read* state, where system remains until condition *time≤ReadCycle* is satisfied. *ReadCycle* is a basic interval during which the finalization of the input actions (opening input gates, filtering non-quantized oscillations, storing stable values into memory) is performed. System enters the *write* state when condition *time>BusCycle*. *BusCycle* is an interval during which the sequence of the states: $Ready \rightarrow Read \rightarrow Ready \rightarrow Write$ is finalized. *BusCycle* is an approximate model of the cycle time T_c - the time required by the operating system to run the cyclic program and all the program sections that interrupt the cycle and system activities [Siem96]. The word "approximate" is used with the regard to the fact that cycle time is not the same in every cycle - see Figure 6-8 for details.

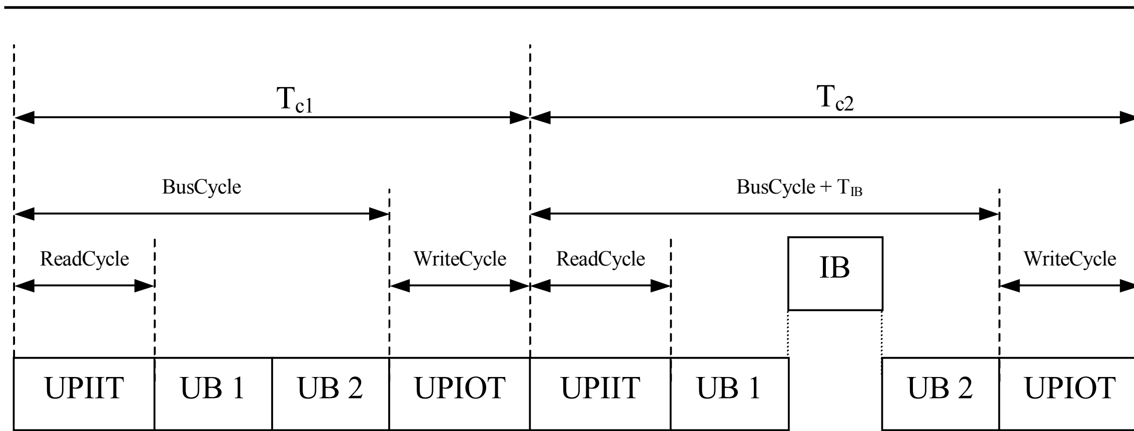

Figure 6-8: Cycle Times of different lengths

Update Process Image Input Table - UPIIT corresponds to the above defined *ReadCycle* interval in timed-automaton model and Update Process Image Output Table - UPIOT corresponds to the above defined interval *WriterCycle*. Apparently, $T_{c1} \neq T_{c2}$ due to the fact that in the second cycle of the PLC interrupt block IB is called in addition (T_{IB}). Thus User Blocks UB1 and UB2 have different duration time from the external point of view. This inaccuracy is acceptable if we consider fact, that interrupt blocks constitute less then 3% of the complete control algorithm.

System enters the *stop* state if the following condition is satisfied:

$$IOBus \in \{Ready, Read, Write\} \wedge \uparrow IOError \Rightarrow IOBus = Stop$$

This mechanism models situation when an external error is occurred in the I/O subsystem of the PLC. Possible external errors are: overloading due to the voltage incompatibility (EMC), power supply loss, short-circuit on the bus, etc. In this case, the only way how to restore timed-automaton of the model is to set up external signal (*CPUReset*):

$$IOBus = Stop \wedge \uparrow CPUReset \Rightarrow IOBus = Ready$$

Timed automaton of the Profibus DP is shown in Figure 6-9. Model has similarly as the previous model four states:

$$DPBusStates == \{Ready, Read, Stop, Write\}$$
$$DPBus : DPBusStates$$
$$time : R$$

$$time \geq 0$$
$$DPBus \in \{Read, write\} \Rightarrow time = 0$$
$$DPBus = Stop \wedge \uparrow CPUReset \Rightarrow DPBus = Ready$$

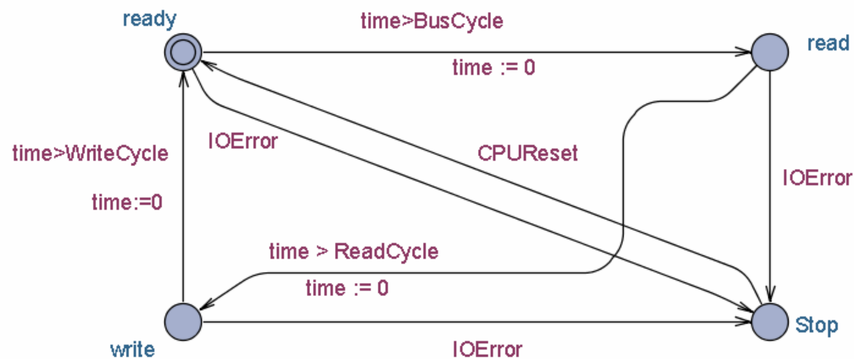

Figure 6-9: Timed-automaton of the DP Bus

Local quantifier *time* again determines temporal behaviour of the DP bus system. At the beginning the system is in the *Ready* state. Once condition $time > BusCycle$ is realized and if no errors are presented in the system, then enters the *Read* state where persists until $time \leq ReadCycle$. Then system enters the *Write* state, where persists until $time \leq WriteCycle$. *Message time* T_M is an interval during which the sequence of the states: $Ready \rightarrow Read \rightarrow Write$ is finalized. Contrary of the time behaviour of the IO bus, *message time* of the DP bus has precise and unique timing [PNOA91] and [PNOC94]:

$$T_M = (IT + OT) \cdot 11 \cdot TBit \;\; [s]$$
$$TBit = \frac{1}{Br} \;\; [s] \tag{6-4}$$

where

IT is Input part of the DET - see glossary or [LVPK02] for details

OT is Output part of the DET - see glossary or [LVPK02] for details

TBit the time to transmit one bit [s]

Br is *BaudRate* of the bus [s^{-1}]

Basic time interval comparable with *BusCycle* for IO bus is called *Message Cycle Time* T_{MC}. This is the time between transmission of the first bit of an action frame (request telegram) and receipt of the last bit of the corresponding reply frame (response telegram):

$$T_{MC} = T_R + T_{SDR} + T_S \tag{6-5}$$

where

T_{SDR} is Station Delay of Responser - see glossary for details

T_S and T_R are send/request *frames* Time - see glossary for details

Generally

$$T_{MC} > T_M \tag{6-6}$$

because if we consider equation 6-5 where $T_{SDR} = 0$, i.e. the responser reacts without delay and $T_R = T_M$, i.e. *Request Frame* is equal to *Message time,* then the shortest possible answer from the responser is SD3 Data telegram with fixed data length (8 bytes long):

$$T_S = 8 \cdot 11 \cdot TBit \ \ [s] \tag{6-7}$$

The relations among T_c, *Bus cycle* and T_{MC} is evident from the following Figure 6-10.

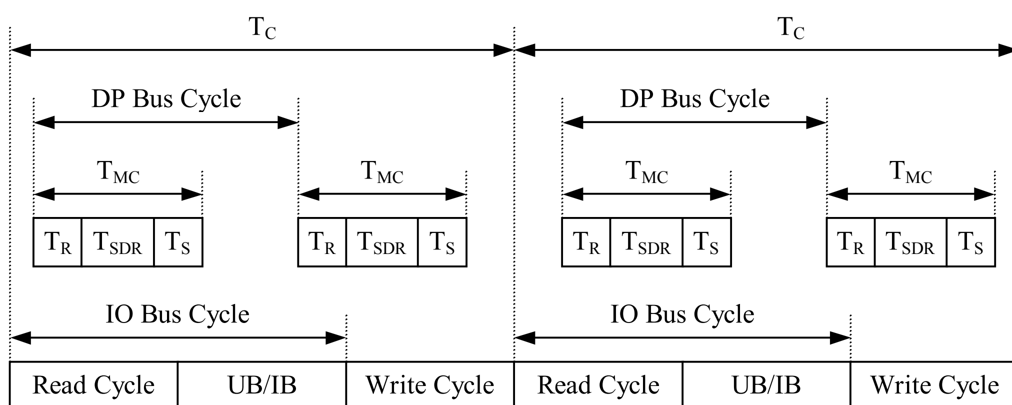

Figure 6-10: The relations among Tc, Bus cycle and T_{MC}

DP Bus system enters the *Stop* state if the following condition is satisfied:

$$DPBus \in \{Ready, Read, Write\} \wedge \uparrow IOError \Rightarrow DPBus = Stop$$

This mechanism models situation when an external error is occurred in the DP bus system of the model. Possible external errors are: data violation due to EMI, power supply loss - e.g. in repeater(s) , short-circuit or disconnection of the bus - subset of EMC problems, etc. In this case, the only way how to restore timed-automaton of the model is to set up external signal (*CPUReset*):

$$DPBus = Stop \wedge \uparrow CPUReset \Rightarrow DPBus = Ready$$

DPBus and *IOBus* are implemented only because of possibility to identify behaviour of the control algorithm in the real environment of the PLC and industrial communication bus (Profibus DP). These models are only simplified approximation of the reality and the only parameters that have influence to our simulation are modelled.

Timed-automaton model of the control algorithm is shown in

Figure 6-11. States of the algorithm are defined as:

$$CAStates == \{Idle, WaitDP, Vote, RecV, Comp, Reconf, Terminate\}$$
$$CA : CAStates$$

This model starts form the flowchart of the algorithm in Figure 6-6. State *Idle* corresponds to “continue in task” in above mentioned flowchart. States *WaitDP*, *Vote*, *RecV* and *Comp* correspond to action “Receive and storage ...”, “Vote between ...”, “Receive Concurrent ...” and “V1<>V2?” comparison. Finally, the state *Terminate* corresponds to the action “Terminate task”.

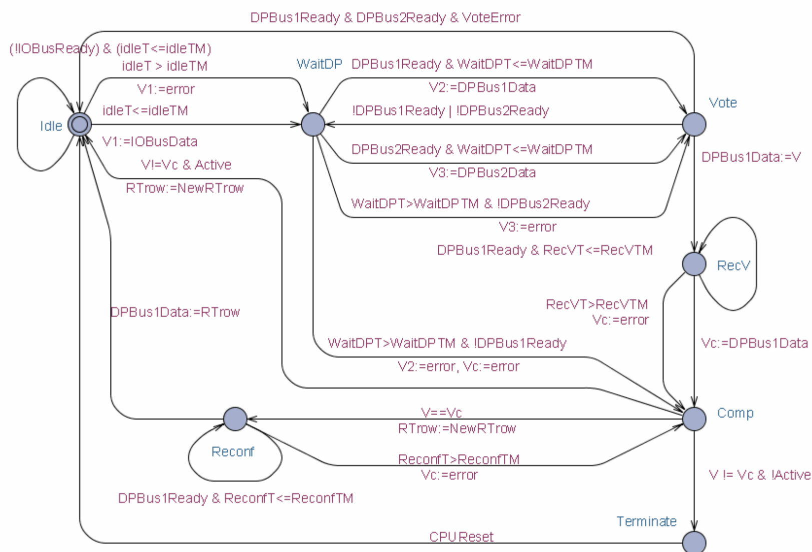

Figure 6-11: Timed-automaton of the control algorithm

The timed-automaton model of the control algorithm rises question: "Does it work?". Simple question but complicated answer. First it is necessary to define term "work".

According to the chapter 5.1, the verification of the system (control algorithm) is possible to divide into three basic conditions:

1. The safety conditions, e.g. those that must not occur in operation of the system:

   a, System must be deadlock free.

   b, There is never more than one CPU comparing the votes at any time instance.

   c, There is never more than one CPU reconfiguring the task at any time instance.

2. The liveness conditions, e.g. those that specify what the have to do:

   a, Whenever a CPU compares the votes, it will eventually reconfigure the task.

   b, Bounded Liveness: A system will reconfigure within the safe time Ts once error occurs.

   c, Both CPU can enter the Idle, WaitDP, Vote, RecV, Comp, Reconf and Terminate state.

   d, Whenever a CPU enters Terminate state it will eventually enters the Idle state.

3. The fairness conditions, e.g. those how nondeterministic conditions are to be resolved.

   a, A system will reconfigure even when IO Bus, DP Bus and Vote errors occur together.

   b, A system will not leave reconfiguration state once CPUReset signal occurs.

To verify these requirements the simulation model has to be established. To establish such model is easy task due to the fact that templates for IO Bus, DP Bus and

control algorithm have been created - see Figure 6-7, Figure 6-9 and Figure 6-11; then simulation model of the IO and DP buses is only the question to derive new local variables for above mentioned template - see Figure 6-12 and as well as simulation model for control algorithms in S7-200 and S7-300 - see Figure 6-13 and Figure 6-14.

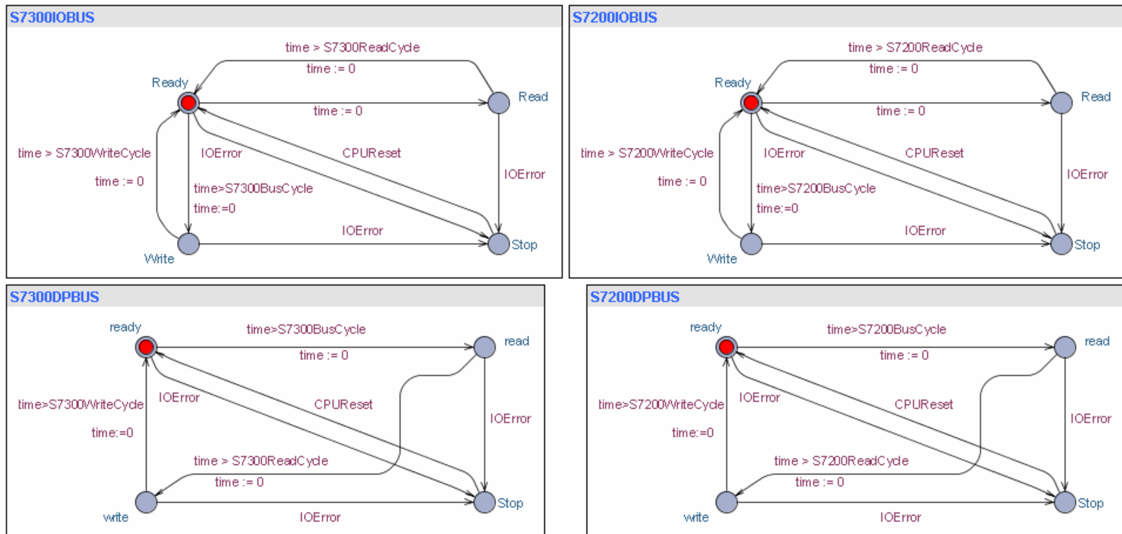

Figure 6-12: Simulation configuration for the IO and DP buses

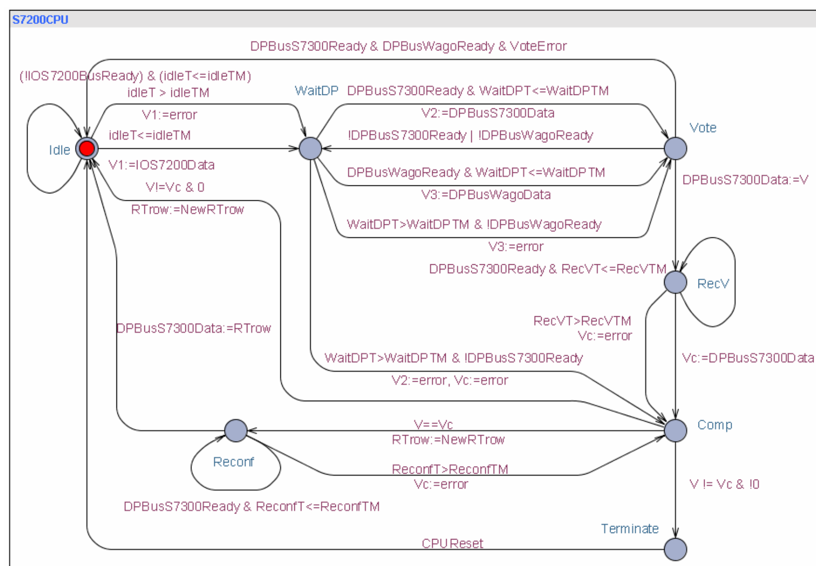

Figure 6-13: Simulation configuration for S7-200 PLC

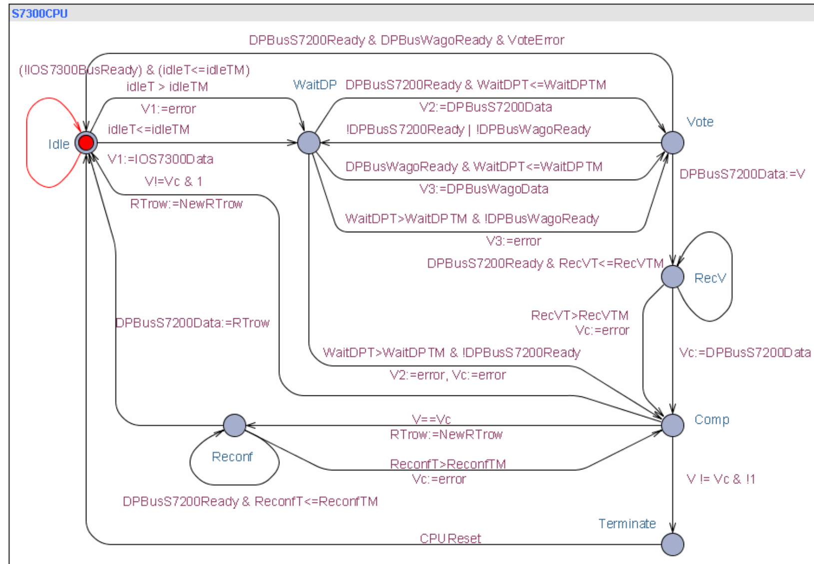

Figure 6-14: Simulation configuration for S7-300 PLC

The simulation model makes possible to observe if the above defined conditions are or are not satisfied. If we express these conditions in temporal logic formulas, than the automatic verifier is able to answer the basic question: "Does it work?". If we consider chapter 5 - Formal verification above defined informal conditions and simulation model, then:

1.a $\forall[]\ \neg(deadlock)$

1.b $\forall[]\ \neg(\mathrm{S7200CPU.Comp} \wedge \mathrm{S7300CPU.Comp})$

1.c $\forall[]\ \neg(\mathrm{S7200CPU.Reconf} \wedge \mathrm{S7300CPU.Reconf})$

2.a $\mathrm{S7200CPU.Comp} \rightarrow \mathrm{S7200CPU.Reconf}$

$\mathrm{S7300CPU.Comp} \rightarrow \mathrm{S7300CPU.Reconf}$

2.b

$$\forall[]\ \mathrm{S7200.Reconf} \rightarrow ((\mathrm{S7200.V2} = \mathrm{error} \vee \mathrm{S7200.Vc} = \mathrm{error}) \wedge \mathrm{S7200.T} \leq \mathrm{Ts})$$
$$\forall[]\ \mathrm{S7300.Reconf} \rightarrow ((\mathrm{S7300.V2} = \mathrm{error} \vee \mathrm{S7300.Vc} = \mathrm{error}) \wedge \mathrm{S7300.T} \leq \mathrm{Ts})$$

2.c $\exists<>(\mathrm{S7200.Idle})$, $\exists<>(\mathrm{S7300.Idle})$

$\exists<>(\mathrm{S7200.WaitDP})$, $\exists<>(\mathrm{S7300.WaitDP})$

$\exists <> (S7200.Vote)$, $\exists <> (S7300.Vote)$

$\exists <> (S7200.RecV)$, $\exists <> (S7300.RecV)$

$\exists <> (S7200.Comp)$, $\exists <> (S7300.Comp)$

$\exists <> (S7200.Reconf)$, $\exists <> (S7300.Reconf)$

$\exists <> (S7200.Terminate)$, $\exists <> (S7300.Terminate)$

2.d $S7200CPU.Terminate \rightarrow S7200CPU.Idle$

$S7300CPU.Terminate \rightarrow S7300CPU.Idle$

3.a

$\forall[]\ S7200.Reconf \rightarrow ((S7200.V1 = error \wedge S7200.V3 = error \wedge S7200.Vc = error)$
$\forall[]\ S7300.Reconf \rightarrow ((S7300.V1 = error \wedge S7300.V3 = error \wedge S7300.Vc = error)$

3.b $\forall[]\,S7200CPU.Idle \rightarrow (S7200CPU.Reconf \wedge St200.CPUReset)$

As soon as the verifier positively answers to the conditions defined above, the SFC interpretation of the model is created and in case that PLC does not support SFC it is transferred into LAD or IL. In case, that one or more conditions are not satisfied, the designer has to return design into one of the previous defined step.

Reliability model of the final project is easy to calculate with the help of the chapter 4. If Figure 4-1 and Figure 6-3 are compared, then architecture of the project is shown in Figure 6-15;

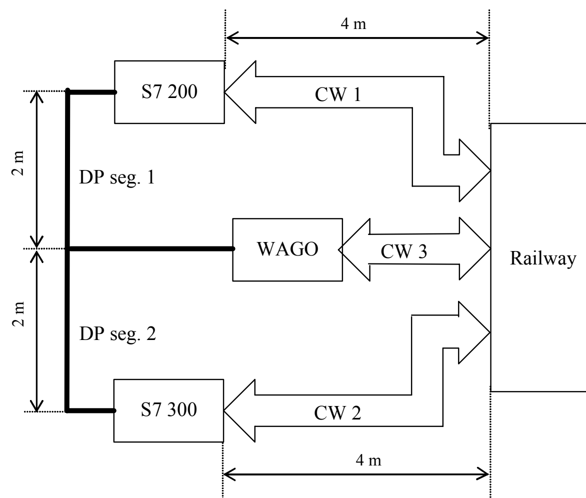

Figure 6-15: Architecture of the model with industrial communication network

and complete reliability model is shown in Figure 6-16.

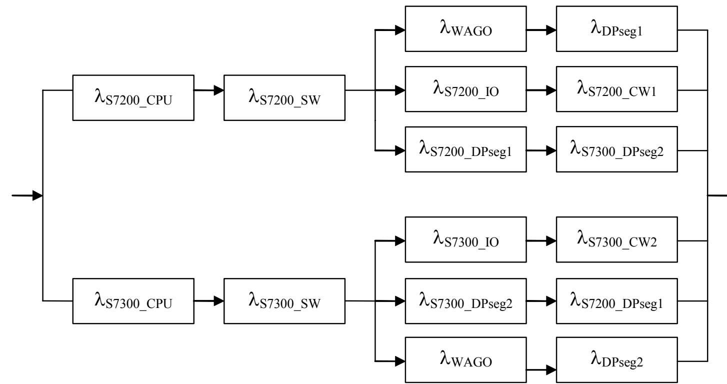

Figure 6-16: Reliability model of the railway application

Model consists of two redundant branches - active DP elements S7-200 and S7-300. According to the Table 6-1, either S7-200 or S7-300 PLC can control task through:

1. relevant DP bus segment and decentralised I/O Wago system, or
2. both DP bus segments and the second PLC, or
3. its I/O subsystem.

Wago is a passive slave on the bus and its SW equipment is hidden in the Profibus DP protocol standard. Length of the conventional wire segment CW3 is much shorter (app. 0.1 m) then length of the segments CW1 and CW2 and connection is hardwired (undetachable), therefore reliability model form the equation (4-27) is not considered. Reliability of the individual HW components (specified by producer) is shown in Table 6-2.

Table 6-2: Reliability of the HW equipment

Component	Symbol	λ [h]
S7-200 CPU	λ_{S7200_CPU}	$1/20\cdot10^3$
S7-300 CPU	λ_{S7300_CPU}	$1/30\cdot10^3$
S7-200 IO subsystem	λ_{S7200_IO}	$1/20\cdot10^4$
S7-300 IO subsystem	λ_{S7300_IO}	$1/30\cdot10^4$
WAGO	λ_{WAGO}	$1/10\cdot10^4$

Reliability models of the CW1 (S7 200) and CW2 (S7 300) are the same; if we consider equation (4-26), where M=2, nI=20, nO=21, then number of necessary wires is:

$$W = 2(20 + 2 \cdot 21) = 124\,. \tag{6-8}$$

If we consider equation (4-27), mechanical coefficient for laboratory environment $\pi_m = 0$ (no vibration), number of control elements $M = 2$, distance between them is $L = 4\,m$, number of wires in one segment is $W_s = W/2 = 62$, and constant electrical coefficient $\pi_{e0} = 0.1\,m^{-1}$ (bus is not shielded), then failure rate of one CW segment system is equal to:

$$\lambda_{S7200_CW1} = \lambda_{S3200_CW2} = 10^{-6} \cdot 62 \cdot 0.1 \cdot 4 = 2.48 \cdot 10^{-4}\,h^{-1}\,. \tag{6-9}$$

DP bus segments 1 and 2 also have the same reliability model. This model is also based on equation (4-27), where $\pi_m = 0$ (no vibration), number of control elements $M = 2$, distance between them is $L = 2\,m$, number of wires in one segment is $W = 2$, and coefficient of electrical coupling $\pi_e(l)$ has waveform from Figure 4-7A, where $\pi_{e0} = 0.1\,m^{-1}$ (laboratory environment) and terminal length $l_t = 0.05\,m$. Failure rate of one DP segment system is equal to:

$$\begin{aligned}\lambda_{S7200_DPseg1} &= \lambda_{S3200_DPseg2} = 10^{-6} \cdot 2\int_0^L \pi_e(l)\,dl = \\ &= 20^{-6}[2 \cdot \int_0^{l_t} (2\pi_{e0} - \frac{\pi_{e0}}{l_t} l)dl + \pi_{e0}(L - 2l_t)] = \\ &= 20^{-6}(0.015 + 0.19) = 4.1 \cdot 10^{-6}\,h^{-1}\,.\end{aligned} \tag{6-10}$$

Reliability of the SW equipment is calculated in the same way as before; DLOC for software equipment in each PLC is calculated from equation (4-18), where $NIO = nI + nO = 41$ and $\alpha = 310$; $DLOC = 1163$ lines and LOC from equation (4-19), where $\gamma = 0.96$; $LOC = 842$ lines of IL code. Number of defects D is calculated step by step from equations (4-13), (4-15) and (4-14); $D = 5$. Design time T is calculated from equations (4-20), (4-21), (4-22) and (4-23), where the Stroud's number $S = 18$; $T = 11$ days. Failure rates of the software are equal $\lambda_{S7200_SW} = \lambda_{S7300_SW}$. If we consider, that design is made by standard way, i.e. debugging time is one third of the developing

time, $T_d = T/2$, then the failure rate of the software λ_{SW} is calculated from equations (4-24) and (4-25), where fault detection rate $b = 0.03\ h^{-1}$:

$$\lambda_{SW|T_d=T/2} = 0.00261\, h^{-1}. \qquad (6\text{-}11)$$

If we consider, that design is made by way defined in Figure 6-1, then debugging time is equal to developing time, $T_d = T$, because formal model and formal verification of the model are debugging activities realized during development of the system. Then, the failure rate of the software λ_{SW} is equal to:

$$\lambda_{SW|T_d=T} = 4.9 \cdot 10^{-5}\ h^{-1} \qquad (6\text{-}12)$$

Reliability of the complete system from Figure 6-16, defined by Table 6-2 and equations (6-9), (6-10) and (6-11) - informal design, per 100 hours of operation is:

$$R(100h) = 0.9457\,. \qquad (6\text{-}13)$$

Reliability of the complete system from Figure 6-16, defined by Table 6-2 and equations (6-9), (6-10) and (6-12) - formal design, per 100 hours of operation is:

$$R(100h) = 0.9999\,. \qquad (6\text{-}14)$$

7. Formal model of the heterogeneous industrial communication bus

This chapter describes formal model of the heterogeneous industrial communication bus. The model has been developed during author's residency in ifak - Institut für Automation und Kommunikation e.V. Magdeburg [Hint03], [Zezu03]. As was mentioned before, heterogeneous structure of the factory automation system is a special example of communication problem. It consists of at least three parts:

1. Software specification of the communication standard.
2. Hardware specification of the communication interface.
3. Specification of the environment - especially problematic of the electromagnetic interference (EMI) or generally electromagnetic compatibility (EMC).

Each of these parts has its own faults and there exists many different methods how to find and fix them. However, if we put these error-free parts together then a new fault occurs. Therefore, it is necessary to describe whole system by convenient formal method that shall allow describe and simulate not only a high level of the communication process but also a low level flow of the signal and problematic of the electromagnetic interference.

7.1. Profibus DP

Profibus is a vendor independent, open field bus standard for a wide range of applications in manufacturing and process automation. Profibus DP is the most frequently used communications profile. The profile is optimized for speed, efficiency and low connection costs, and is designed especially for communications between automation systems and distributed peripherals. The ISO/OSI model of the Profibus is shown in Figure 7-1.

7.	LLI/FMS/PA/DP		Application Layer
6.	not used		Presentation Layer
5.	not used		Session Layer
4.	not used		Transport Layer
3.	not used		Network Layer
2.	Fieldbus Data Link (FDL)		Data Link Layer
1.	RS485/Fiber Optic	IEC1158-2	Physical Layer

Figure 7-1: Communication model of the Profibus

7.2. Structure of the model

Higher layers of the Profibus protocol were modelled in the ESTELLE; ESTELLE is a formal description technique developed by ISO for open distributed systems, especially for communication protocols with layer architecture corresponding to the ISO/OSI [Este87]. Lower layer of the Profibus protocol (Physical Layer) was modelled in the VHDL; VHLD is a widespread standard for the (hardware) design of digital systems. It is based on the structuring of models into hierarchical units that are communicating via signals. Concurrent signal assignment and process statements are used for the behaviour description. It is well suited for the modelling of hardware related units of a communication device (e.g. sender, receiver) as well as the bus medium. For the Physical Layer (i.e. first layer of the ISO/OSI model) two medium types have been modelled: wired medium (asynchronous transmission according to TIA/EIA RS 485 A) and wireless medium (RFieldbus). Structure of the model is shown in Figure 7-2.

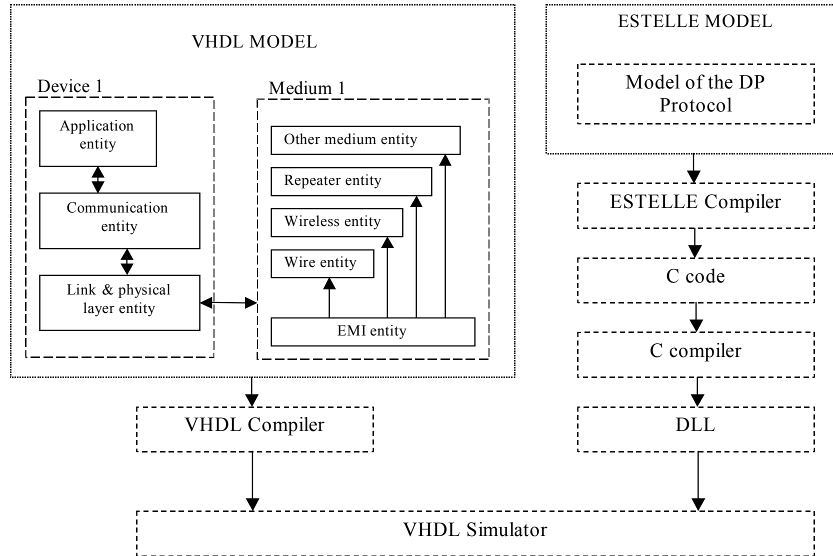

Figure 7-2: Structure of the model

7.3. Application entity

Application entity has two different models with two different architectures:

1. DpMasterAppl - for Master Application module
2. DpSlaveAppl - for Slave Application module

7.3.1. DP Master Application - DpMasterAppl

Interface of the Master Application entity is shown in Figure 7-3 and data types of its signals and generic variables are listed in Table 7-1.

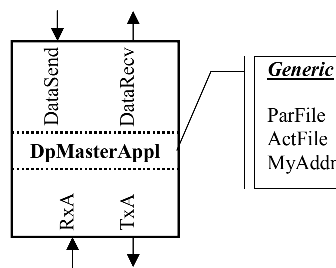

Figure 7-3: DpMasterAppl entity

Table 7-1: 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 Table 7-21 for details.

Master Application entity consists of the one process called MasterState. State machine of this process is shown in Figure 7-4. 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).

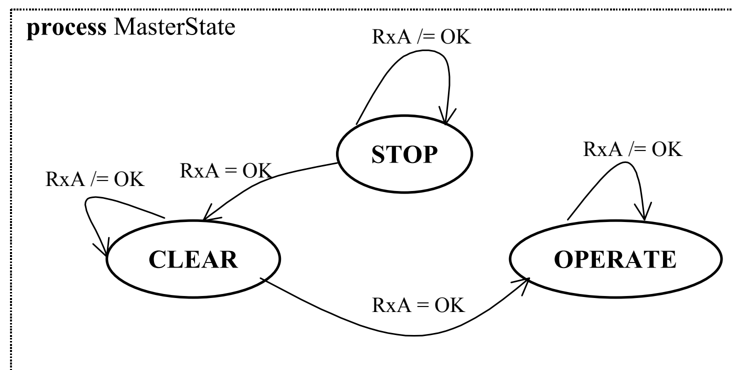

Figure 7-4: State machine of the Master Application

7.3.2. DP Slave Application - DpSlaveAppl

Interface of the Slave Application entity is shown in Figure 7-5 and data types of its signals and generic variables are listed in Table 7-2.

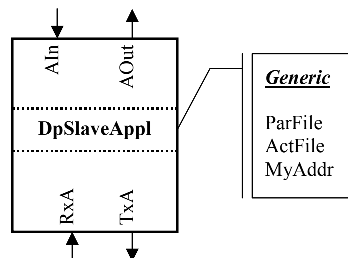

Figure 7-5: DpSlaveAppl entity

Table 7-2: 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 Table 7-21 for details.

Note: Implementation of the DpSlaveAppl architecture is not realized.

7.4. Communication entity

Communication entity has two different models with two different architectures:

1. DpMasterComm - for Master Communication module
2. DpSlaveComm - for Slave Communication module

7.4.1. DP Master communication - DpMasterComm

Interface of the Master Communication entity is shown in Figure 7-6 and data types of its signals and generic variables are listed in Table 7-3.

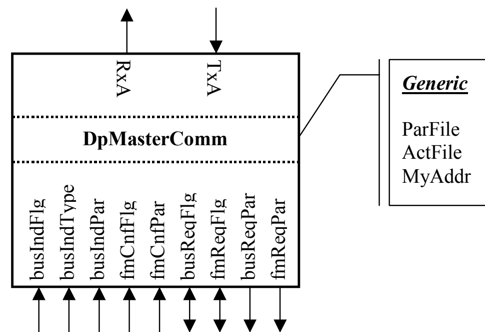

Figure 7-6: DpMasterComm entity

Table 7-3: 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 Table 7-21 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 [Hint01].

7.4.2. DP Slave communication - DpSlaveComm

Interface of the Slave Communication entity is shown in Figure 7-7 and data types of its signals and generic variables are listed in Table 7-4.

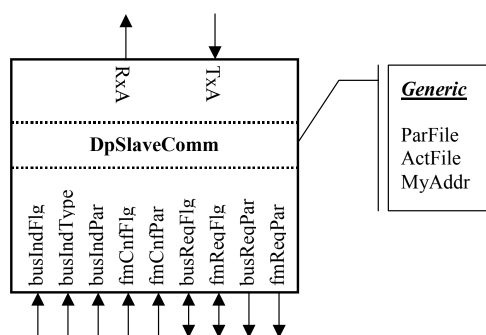

Figure 7-7: DpSlaveComm entity

Table 7-4: 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 Table 7-21 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 [Hint01].

7.5. 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
2. DpUART - for bit-oriented bus

and one entity for wireless communication medium - DpwlS.

7.5.1. DP Send/Receive Telegram - DpSRTelegram

Interface of the DP Send/Receive Telegram entity is shown in Figure 7-8 and data types of its signals and generic variables are listed in Table 7-5.

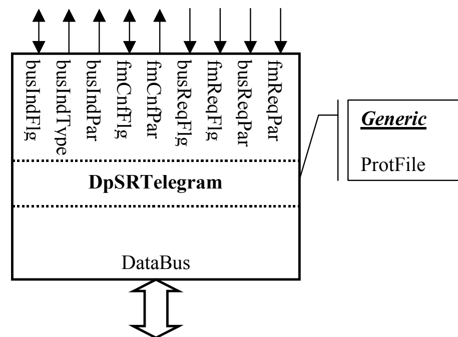

Figure 7-8: DpSRTelegram entity

Table 7-5: 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 Table 7-21 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 modelling '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.

7.5.2. DP UART - DpUART

Interface of the DP UART entity is shown in Figure 7-9 and data types of its signals and generic variables are listed in Table 7-6.

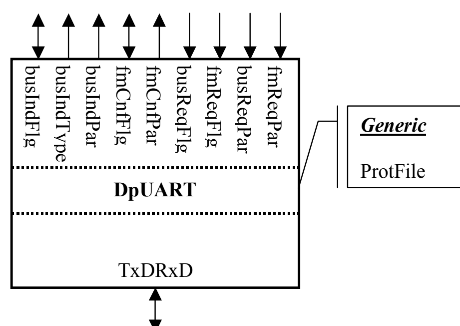

Figure 7-9: DpUART entity

Table 7-6: 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 Table 7-21 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 7.5.1, 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 7.8. 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.

7.6. Telegram Monitor - Monitor

This entity monitors the telegram-oriented bus. Its interface is shown in Figure 7-10 and data types of its signal and generic variables are listed in Table 7-7.

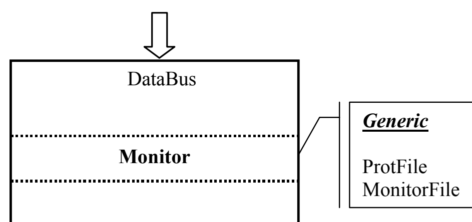

Figure 7-10: Monitor entity

Table 7-7: 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 Table 7-21 for details.

Generic variable ProtFile is used as entry value for BUMO_Init function. Detailed information about DataBus signal is in the chapter 7.5.1. 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.

7.7. UART Monitor - uartMonitor

This entity monitors the bit-oriented bus. Its interface is shown in Figure 7-11 and data types of its signal and generic variables are listed in Table 7-8.

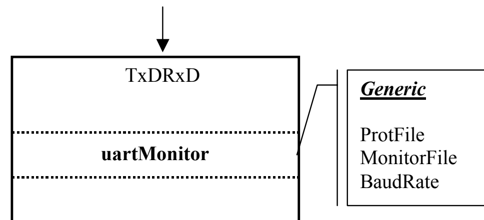

Figure 7-11: uartMonitor entity

Table 7-8: 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 Table 7-21 for details.

Generic variable ProtFile is used as entry value for BUMO_Init function. Detailed information about TxDRxD signal is in the chapter 7.5.2. 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.

7.8. UART Electromagnetic Interference - EMI

This entity stimulates bit-oriented bus by the defined EMI model. There exist three different models of this stimulation:

1. Static model with step-by-step defined disturbance.
2. Static model with stochastic disturbance.
3. Dynamic model with stochastic disturbance.

Following chapters describes above defined models.

7.8.1. Static model with step-by-step defined disturbance

This model is based on user-defined waveform of the disturbance, stored in the stimulation text file. The interface of the entity, ensuring reading and stimulation of the bus during simulation, is shown in Figure 7-12 and data types of its signal and generic variable are listed in Table 7-9.

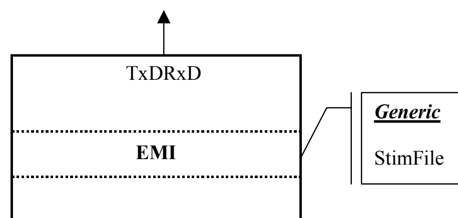

Figure 7-12: EMI entity

Table 7-9: 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 Table 7-21 for details.

Detailed information about TxDRxD signal is in the chapter 7.5.2. Architecture has only one process - Stimul. This process reads text file defined in generic variable (StimFile) and changes the bus state according to its content. Structure of the StimFile is shown in Figure 7-13.

StimFile

```
DLY1    EVT1
DLY2    EVT2
.
.
DLYn    EVTn
```

Figure 7-13: 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 the 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 list shown in Figure 7-14,

StimFile

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

Figure 7-14: Example of the StimFile

then process Stimul generates on the TxDRxD bus EMI as is shown in Figure 7-15.

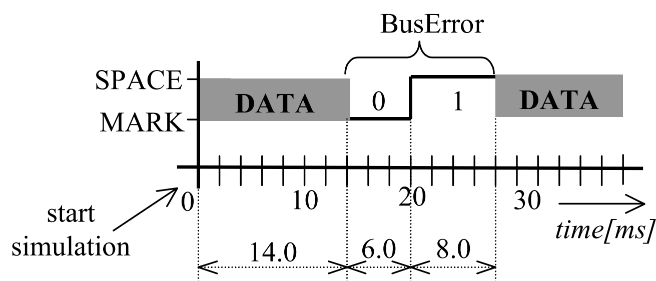

Figure 7-15: An example of the error on the bus

This stimulation is useful in the case we want to determine behaviour of the model in precisely defined EMI intervals. Usually, first we run the simulation without

EMI and from the result of the simulation the interesting time intervals are traced and according to their positions stimulation file is prepared.

7.8.2. Static model with stochastic disturbance.

The other way how to generate EMI is to use static model of the stochastic disturbance. This model is based on some distribution of discrete random variable X. Such a way it is possible to generate EMI automatically by predetermined function without human intervention. There are two main distributions used in technical practise: Binomial distribution and Poisson distribution.

The probability of the Binomial distribution is given by:

$$P(X=k)=\binom{N}{k}\cdot p^{k}\cdot(1-p)^{N-k}, \qquad (7\text{-}1)$$

where

N is number of trials (i.e. number of generated EMI),

k is a time step of the event, $k=0,1,2,...,N$,

p is the probability of the occurring of the EMI in a trial.

The typical symmetrical Binomial distribution with the parameters $p=0.5$ and $N=10$ is shown in Figure 7-16.

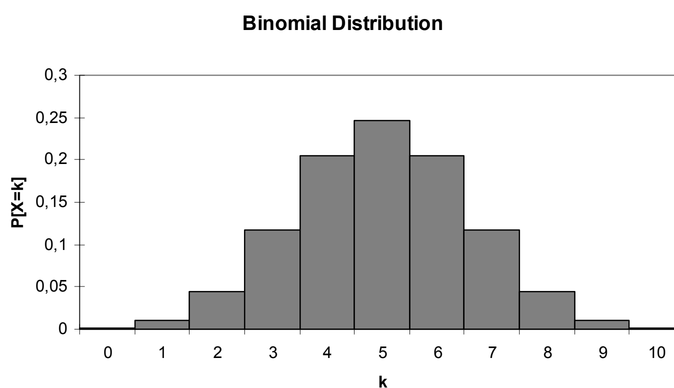

Figure 7-16: Binomial Distribution; p=0.5 and N=10

The interface of the entity EMIsBi stimulating of the bus by this stochastic disturbance is shown in Figure 7-16 and data types of its signal and generic variables are listed in Table 7-10.

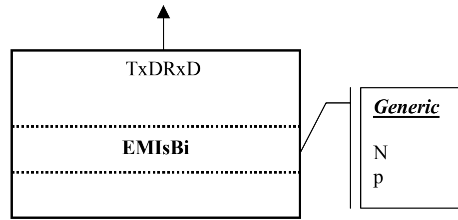

Figure 7-17: EMIsBi entity

Table 7-10: EMIsBi; list of signal/variables

Name	*Direction*	*Type*	*Notice*
N	generic	INTEGER	Number of trials
p	generic	REAL	The probability of the disturbance in a trial
TxDRxD	out	uart_level*	Bit oriented bus medium

* User-defined type; see Table 7-21 for details.

The probability of the Poisson distribution is given by:

$$P(X=k)=\begin{cases}\dfrac{\lambda^k}{k!}e^{-\lambda} & for \quad k=0,1,2,... \\ 0 & othervise\end{cases}, \qquad (7\text{-}2)$$

where

k is a time step of the event, $k = 0,1,2,...$,

λ is the average number of random occurrences (EMI disturbances) per interval.

An example of the Poisson distribution with the parameter $\lambda = 1$ is shown in Figure 7-18.

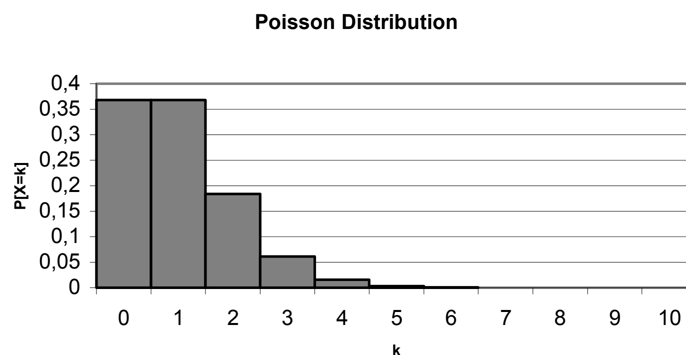

Figure 7-18: Poisson Distribution; λ=1

The interface of the entity EMIsPo stimulating of the bus by this stochastic disturbance is shown in Figure 7-19 and data types of its signal and generic variable are listed in Table 7-11.

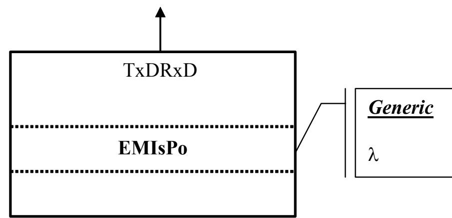

Figure 7-19: EMIsPo entity

Table 7-11: EMIsPo; list of signal/variable

Name	*Direction*	*Type*	*Notice*
λ	generic	REAL	Average number of random occurrences per interval
TxDRxD	out	uart_level*	Bit oriented bus medium

* User-defined type; see Table 7-21 for details.

7.8.3. Dynamic model with stochastic disturbance

Industrial communication buses are often exposed to strong EMI, generated by modern PWM AC drives [Skib99]. Efficient mitigation of EMI by techniques of a grounding, shielding noise form sensitive equipment (industrial buses and controllers), reduction of conducted ground noise current or attenuating the noise source do not guarantee zero value of the coefficient of electrical coupling π_e in the equation 4-27. Modern PWM AC drives produces high frequency noise current of hundred MHz and intensity of the electric field near the drive comes up to 200V/m. Typical areas of the noise current in the chain Transformer-AC drive-Motor-Groung are shown in Figure 7-20. L_1 to L_3 and L_{1d} to L_{3d} represent inputs/outputs phases of the main power from and to the AC drive and I_{lg} is the Line-to-Ground high frequency noise current - the source of the EMI. Industrial Communication Bus (ICB) passes at least through one high-risk EMI area: Transformer↔AC Drive or AC Drive↔Motor and it is target of the strong EMI which increases probability of the error on the bus.

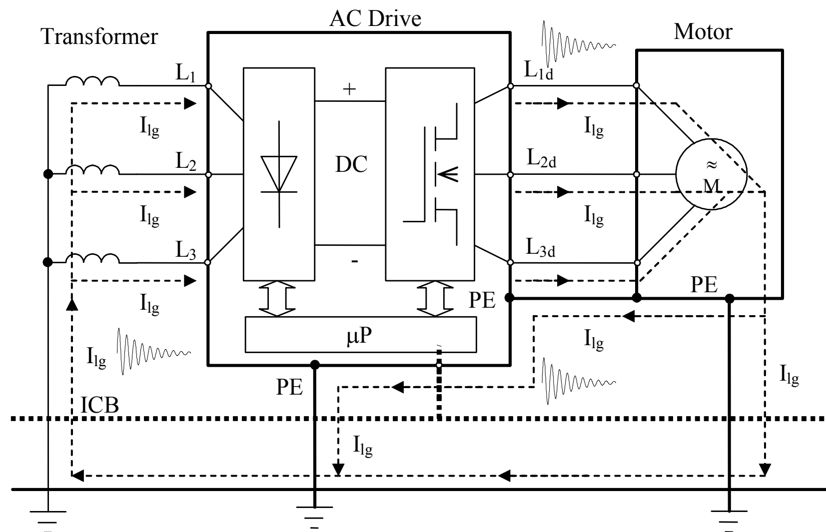

Figure 7-20: AC drive as a source of EMI

The above mentioned intensity of the electric field is not produced permanently but only when AC Drive produces high power energy to the motor; however, this information about the power of the drive is transmitted over the ICB. Therefore, it is possible to get information about the power of the drive and to change stochastic model of the disturbance during simulation; this techniques is called dynamic model with stochastic disturbance. In this case, above defined EMIsBi and EMIsPo entities are replaced by EMIdBi and EMIdPo entities. The difference is obvious - dynamic entities (EMIdBi and EMIdPo) affect parameters of the distribution of the random value (*p* for Binomial distribution and λ for Poisson distribution) by a coefficient *k*, according to the last value of the power transmitted into PWM Drive slave device.

The interface of the entity EMIdBi stimulating of the bus by dynamic Binomial stochastic disturbance is shown in Figure 7-21 and data types of its signal and generic variables are listed in Table 7-12.

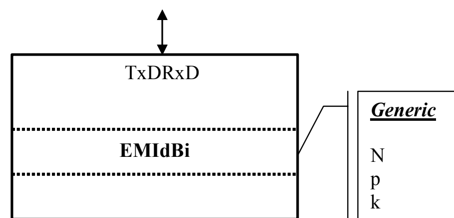

Figure 7-21: EMIsBi entity

Table 7-12: EMIdBi; list of signal/variables

Name	*Direction*	*Type*	*Notice*
N	generic	INTEGER	Number of trials
P	generic	REAL	The probability of the disturbance in a trial
k	generic	REAL	Coupling coefficient
TxDRxD	inout	uart_level*	Bit oriented bus medium

* User-defined type; see Table 7-21 for details.

The interface of the entity EMIdBi stimulating of the bus by dynamic Poisson stochastic disturbance is shown in Figure 7-22 and data types of its signal and generic variables are listed in Table 7-13.

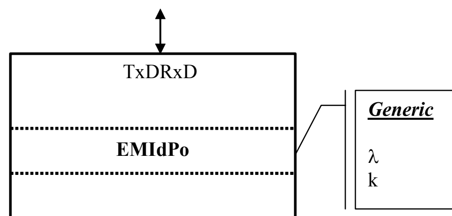

Figure 7-22: EMIdPo entity

Table 7-13: EMIdPo; list of signal/variables

Name	*Direction*	*Type*	*Notice*
λ	generic	REAL	Average number of random occurrences per interval
k	generic	REAL	Coupling coefficient
TxDRxD	out	uart_level*	Bit oriented bus medium

* User-defined type; see Table 7-21 for details.

7.9. Telegram Repeater - TelRep1

This entity models bus repeater with constant propagation delay for telegram oriented bus (TDataBus). Interface of the entity is shown in Figure 7-23 and data types of its signal and generic variable are listed in Table 7-14.

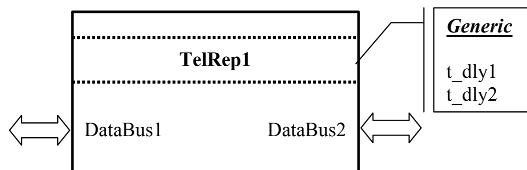

Figure 7-23: TelRep1 entity

Table 7-14: 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 Table 7-21 for details.

Meaning of the variables t_dly1 and t_dly2 is evident from the Figure 7-24.

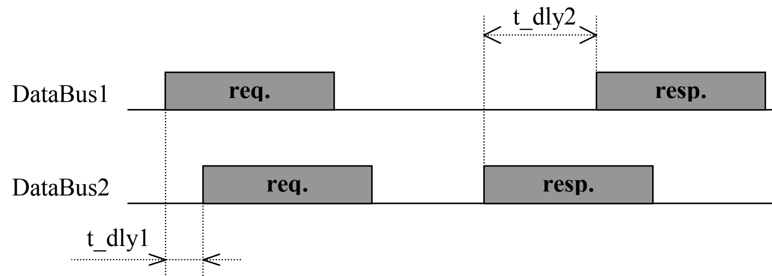

Figure 7-24: Time behaviour of the TelRep1

7.10. UART Repeater - UARTRep11

This entity models bit-oriented bus repeater. This repeater supports different communication speeds on the connected segments. Interface of the entity is shown in Figure 7-25and data types of its signal and generic variable are listed in Table 7-14.

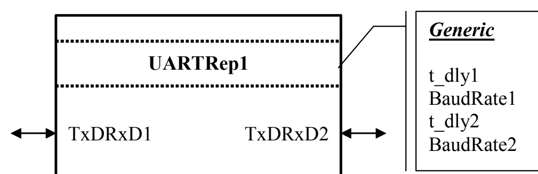

Figure 7-25: UARTRep1 entity

Table 7-15: 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 Table 7-21 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 Figure 7-26 for details.

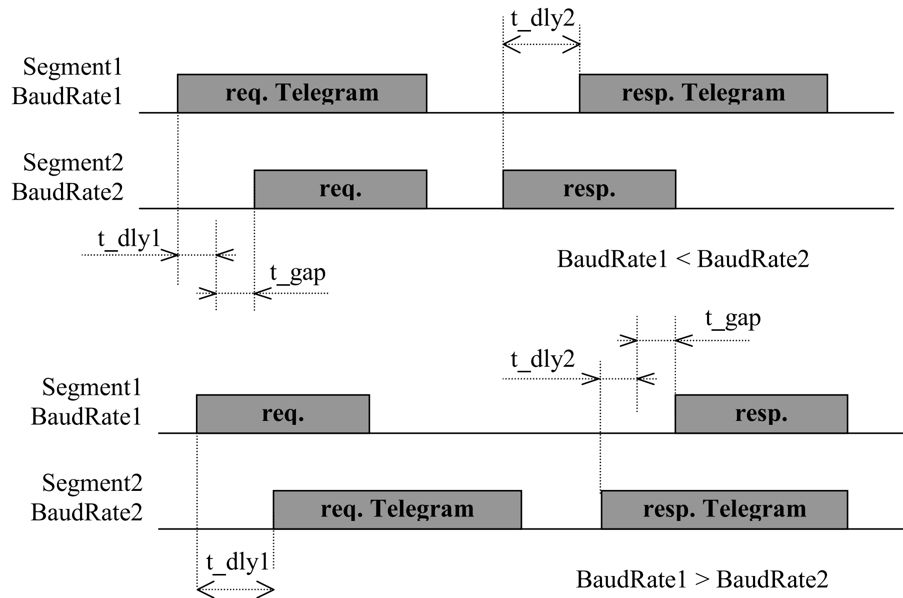

Figure 7-26: Time behaviour of the UARTRep1

7.11. Wireless Link Station - WILS

This entity models behaviour 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 Figure 7-27 and data types of its signal and generic variable are listed in Table 7-16.

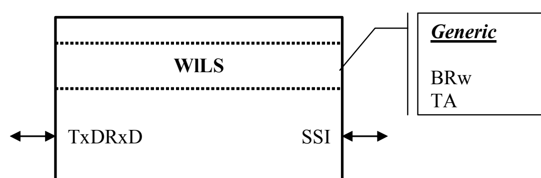

Figure 7-27: WILS entity

Table 7-16: WILS; 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 Table 7-21 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 [Adam01]. 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 7.15. 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 [Adam01] chapter 6, except RFieldbus Phy Preamble. In VHDL model, the Preamble is generated as 30 μs bit pattern of L0 and L1 states on the bus - see Figure 7-28.

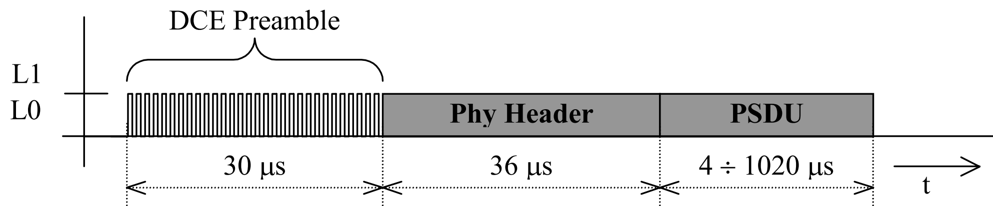

Figure 7-28: VHDL model of the PhLPDU

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:

$$BR = \frac{59}{\sum_{n=1}^{59} DLY_n} \quad s^{-1}, \tag{7-3}$$

where DLY is a time delay between the minimum state change on the bus during synchronization - see Figure 7-29.

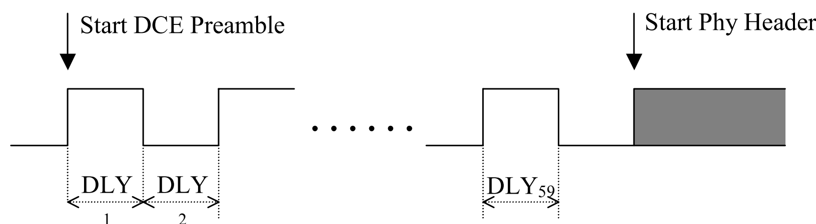

Figure 7-29: Detail of the DCE Preamble

Function behaviour of the LS model corresponds to the [Koul02] chapter 2.4.

7.12. Wireless Base Station - WIBS

This entity models behaviour of the Base Station (BS). BS is a repeater between two wireless segments. Interface of the entity is shown in Figure 7-30 and data types of its signal and generic variable are listed in Table 7-17.

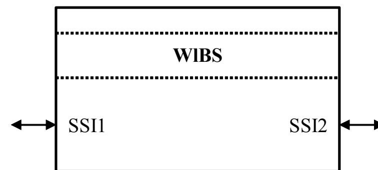

Figure 7-30: WIBS entity

Table 7-17: WIBS; list of signals

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

* User-defined type; see Table 7-21 for details.

Function behaviour of the BS model corresponds to the [Koul02] chapter 2.5, except the beacon mechanism that is not implemented and only two wireless segments (SSI1 and SSI2) are possible.

7.13. Wireless Link Base Station - WILBS

This entity models behaviour 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 Figure 7-31 and data types of its signal and generic variable are listed in Table 7-18.

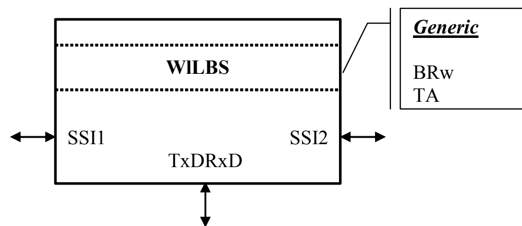

Figure 7-31: WILBS entity

Table 7-18: WILBS; 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 Table 7-21 for details.

Function behaviour of the LBS model corresponds to the [Koul02] chapter 2.4 and 2.5, except the beacon mechanism that is not implemented and only two wireless segments (SSI1 and SSI2) are possible.

7.14. DP Wireless Station - DPwlS

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 Figure 7-32 and data types of its signals and generic variables are listed in Table 7-19.

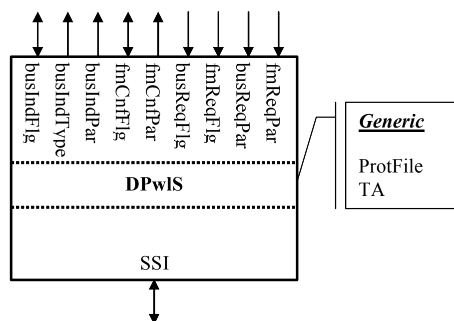

Figure 7-32: DPwlS entity

Table 7-19: DPwlS; list of signals/variables

Name	***Direction***	***Type***	***Notice***
ProtFile	generic	STRING	Pointer to the protocol file
TA	generic	bits8*	Transmitter Address
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 Table 7-21 for details.

7.15. Wireless Electromagnetic Interference - wlEMI

This entity stimulates bit-oriented wireless connection by the defined EMI model. There exist three different models of this stimulation:

1. Static model with step-by-step defined disturbance.

2. Static model with stochastic disturbance.

3. Dynamic model with stochastic disturbance.

Following chapter describes wireless Static model with step-by-step defined disturbance. Wireless static and dynamic models with stochastic disturbance are formally the same as was described in chapters 7.8.2 and 7.8.3., except the fact, that in/out signal TxDRxD is replaced by above defined SSI interface - chapter 7.11.

7.15.1. Static model with step-by-step defined disturbance

This model is based on user-defined waveform of the disturbance, stored in the stimulation text file. The interface of the entity, ensuring reading and stimulation of the bus during simulation, is shown in Figure 7-33 and data types of its signal and generic variable are listed in Table 7-20.

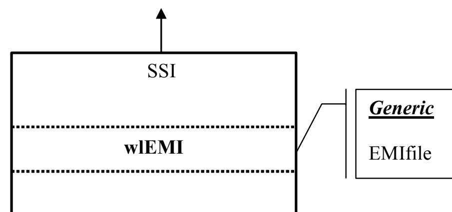

Figure 7-33: wlEMI entity

Table 7-20: 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 Table 7-21 for details.

Detailed information about SSI signal is in the chapter 7.11. 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 Figure 7-34.

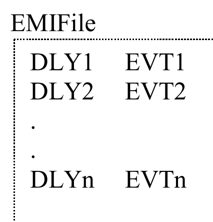

Figure 7-34: Structure of the EMIFile

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 list in :

EMIFile

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

Figure 7-35: Example of the EMIFile

then process Stimul generates on the SSI transfer medium EMI as is shown in Figure 7-36.

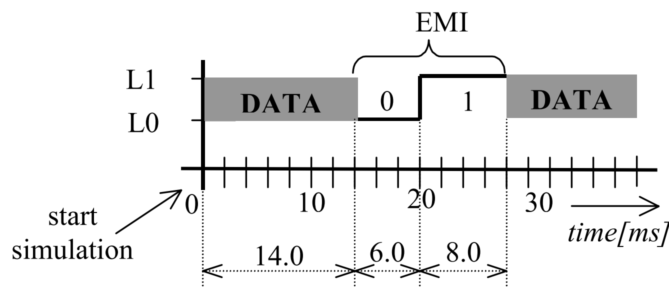

Figure 7-36: An example of the error on the SSI medium

7.16. 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.

7.16.1. Telegram Bus Test - test

Configuration entity of the model is shown in Figure 7-37.

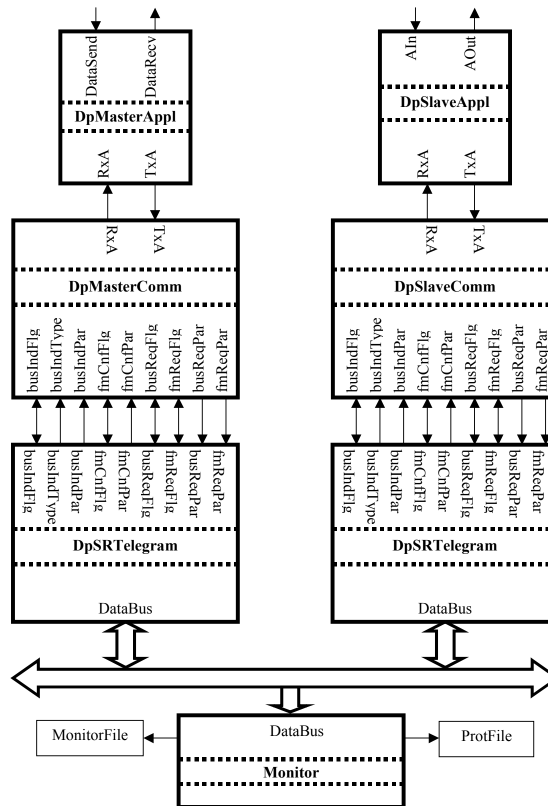

Figure 7-37: test configuration

This simulation enables observe basic time behaviour of two devices connected at the same segment by telegram oriented communication model. Entities DpMasterAppl↔DpMasterComm↔DpSRTelegram represent model of the master device and entities DpSlaveAppl↔DpSlaveComm↔DpSRTelegram represent model of the slave device. Common telegram oriented communication bus DataBus is monitored by entity Monitor. Results of the simulations are stored in the MonitorFile and ProtFile. Advantage of the simulation based on the telegram oriented model is speed of the simulation - disadvantage is impossibility to investigate bit behaviour and to stimulate the communication by EMI.

7.16.2. Telegram Bus Test - test8

Simplified configuration entity of the model is shown in Figure 7-38. Master's object (MASTER) is represented by a connection of three entities: DpMasterApll↔ DpMasterComm↔DpSRTelegram as is shown in Figure 7-37. Slaves' objects (SLAVE1 to SLAVE8) are also represented by a connection of three entities: DpSlaveAppl↔DpSlaveComm↔DpSRTelegram. All devices are connected at the same telegram oriented bus segment; advantages and disadvantages of this simulation are the same as for above mentioned configuration Test. Number of connected devices is only restricted by Profibus standard, however increasing number of devices decreases the speed of the simulation.

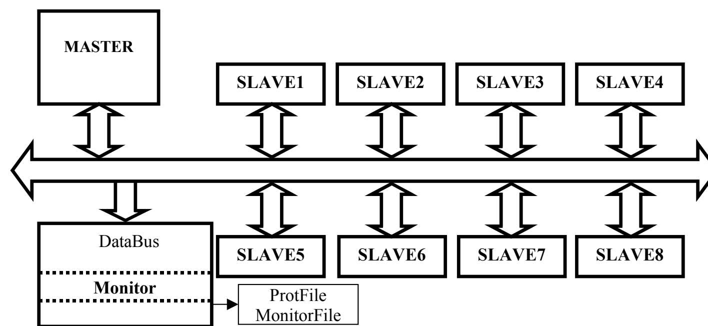

Figure 7-38: Test8 configuration

7.16.3. UART Bus Test - testUART

Configuration of the model is shown in Figure 7-39. This simulation model is formally the same as test configuration described in chapter 7.16.1. There are only two differences: communication medium is bit-oriented (TxDRxD) and model is appended by EMI entity. It enables to observe bit behaviour of the bus and to stimulate bus by user-defined EMI pattern (StimFile).

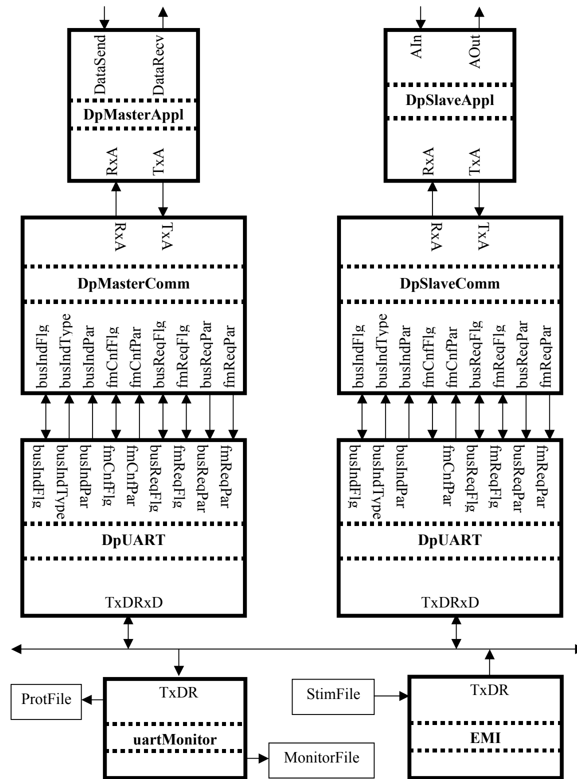

Figure 7-39: testUART configuration

7.16.4. UART Bus Test - test8UART

Simplified configuration of the model is shown in Figure 7-40. This simulation model is formally the same as test8 configuration described in chapter 7.16.2. There are only two differences: communication medium is bit-oriented (TxDRxD) and model is appended by EMI entity. It enables to observe bit behaviour of the bus and to stimulate bus by user-defined EMI pattern (StimFile).

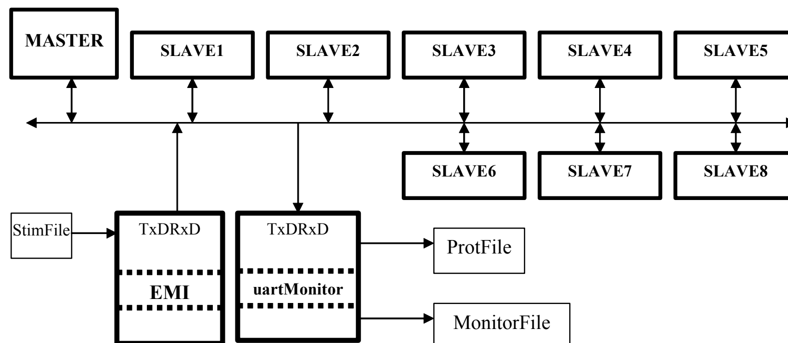

Figure 7-40: test8UART configuration

7.16.5. Telegram Repeater - testTelRep1

Configuration of the model is shown in Figure 7-41. This simulation separates bus into two segments by telegram repeater entity TelRep1. Generic variables t_dly1 and t_dly2 (see chapter 7.9) determine constant telegram propagation delay between segments. In this case, propagation delay of the telegram from master to slave is 3 μs and form slave to master device is 1 μs. The number of TelRep1 entities (i.e. number of segments) is not restricted in the simulation. This simulation enables obtain time characteristic of the network with one or more segments. The possibility of EMI stimulation or different communication speeds in the segments is not supported by TelRep1 entity.

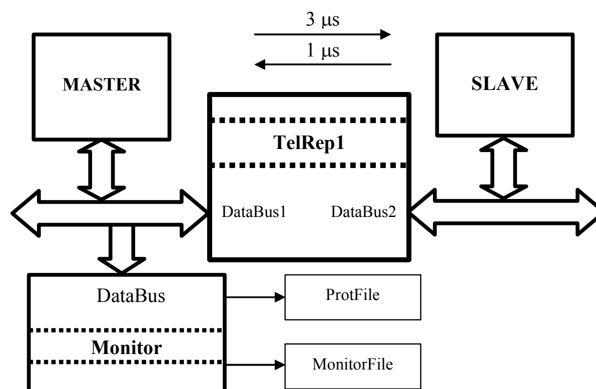

Figure 7-41: testTelRep1 configuration

7.16.6. UART Repeater - testUARTRep1

Configuration of the model is shown in Figure 7-42.

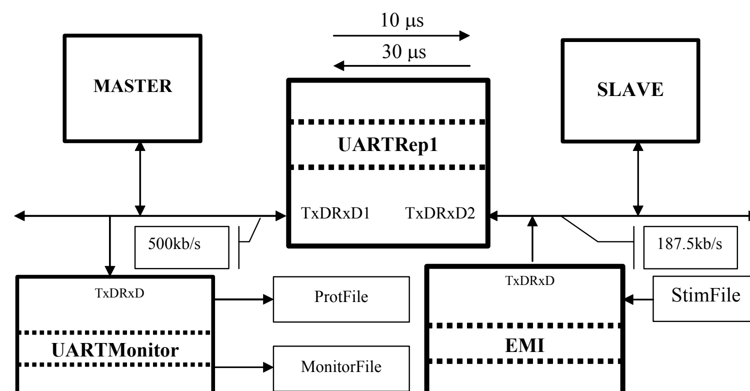

Figure 7-42: testTelRep1 configuration

This simulation separates bus into two segments by bit-oriented repeater entity UARTRep1. Generic variables t_dly1 and t_dly2 (see chapter 7.10) determine propagation delay between segments. In this case, propagation delay of the telegram from master to slave is 10 μs and form slave to master device is 30 μs. Because transfer medium is raw physical signal (RxDTxD), it is possible to model different communication speeds on the segments. In this case, communication speed of the master's segment is 500kb/s and slave's segment communicates at 187.5kb/s. EMI entity enables stimulate slave's segment by user-defined error pattern. The number of UARTRep1 entities (i.e. number of segments) is not restricted in the simulation.

7.16.7. Link Station - testLS

Configuration of the model is shown in Figure 7-43. Two wire segments are connected together via one wireless segment. The function of the communication repeaters between wire and wireless segments fulfil Wireless Link Stations LS1 and LS2. Besides, wireless segment is stimulated by EMI - entity wlEMI. System is monitored by UARTMonitor entity and results of the simulation are stored in ProtFile and MonitorFile.

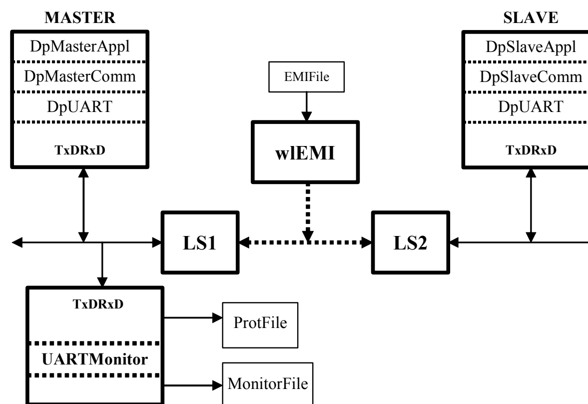

Figure 7-43: testLS configuration

7.16.8. Base Station - testBS

Configuration of the model is shown in Figure 7-44. Simulation contains two wire segments connected via Link Stations LS1 and LS2 and two wireless segments separated by Base Station BS. On of the wireless segment is stimulated by EMI - entity

wlEMI. Master's wire segment is monitored by UARTMonitor entity and results of the simulation are stored in ProtFile and MonitorFile.

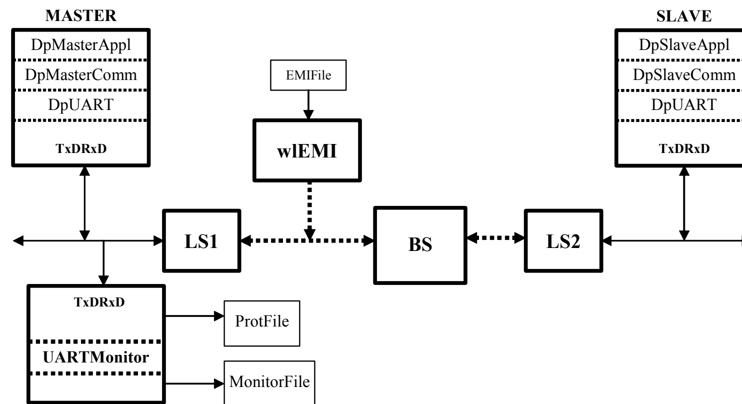

Figure 7-44: testBS configuration

7.16.9. Wireless Station - testwlS

Configuration of the model is shown in Figure 7-45. This simulation consists of one wire and one wireless segment. This feature is enable by Link Base Station (chapter 7.13) and Wireless Station (chapter 7.14). Wireless segment is stimulated by EMI pattern stored in EmiFile (wlEMI entity) and simulation is monitored by UARTMonitor entity at the wire segment.

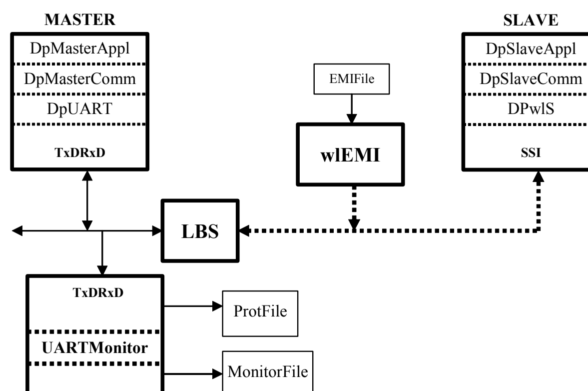

Figure 7-45: testwlS configuration

7.16.10. Base Station & Wireless Station - testBSwlS

Configuration of the model is shown in Figure 7-46. Three different segments are created in the model. One wire segment (master's segment) and two wireless segments:

first segment between Link Station (LS) and Base Station (BS), where EMI affects and second segment between Base Station (BS) and Wireless Station (DPwlS).

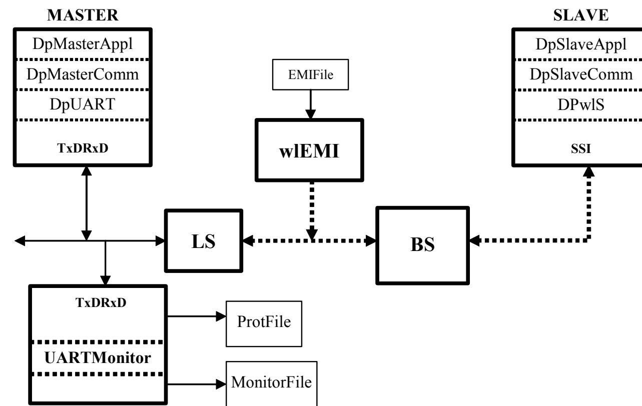

Figure 7-46: testBSwlS configuration

7.17. User-defined types

Table 7-21: 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)

8. Conclusion

This dissertation work deals with formal methods in industrial communication. Author's contributions into this area are:

1. Non-formal M-redundancy concept, based on real-time communication between single control elements.
2. Formal extension of this concept into the sphere of the reliability and availability of the software and hardware design.
3. Formal model of this extension together with the model of the heterogeneous communication system, enabling modelling environmental characteristics.

Non-formal M-redundancy concept is based on well known fact that redundancy generally gives a possibility to improve basic reliability and availability of the system by placing the same subsystem (HW, SW) into the system several times. This concept is easy understandable and mathematically verifiable, however it raises serious problems in practical applications; e.g. price of the final system and cooperation of the subsystem in critical moments. M-redundancy concept is based more in existing system architecture then adding M same subsystem according to the calculated redundancy scheme. Every modern industrial controller is equipped by one or more communication standard and so effective interconnection between them is not question of the price but designer's imagination. This interconnection enables to transfer various information over the system and to effective detect problem as well as to effective reconfigure inner structure [Honz02]. This information exchange is not inherent but it is designer's responsibility to program it and to use it. However, programming such of effectively communication system is not simple and sometimes it is more difficult than to program the control task. Thus the formal extent is necessary, because it helps to fix designer's attention more on the algorithm then on the implementation details. Another serious sphere where the formal methodology helps the designer to keep the system under control is a friction area between hardware and software design - the perpetual problem

of the embedded or real-world oriented systems. An efficient example of the formal description tool convenient for this purpose is the Asynchronous Specification language (ASL) of Prof. Miroslav Sveda from the Department of Computer Systems, Faculty of Information Technology, Technical University of Brno. Practical application of this tool is demonstrated in appendix 9.2. Another way how to break through the barrier between HW and SW design is to extent purely HW description tool into the sphere of the SW design. This example is presented by author's work on formal description and simulation of the real-time communication protocol [Kucp03] by VHDL. VHDL is inestimable in the sphere of the electronic design and under some conditions it is easy to use its hierarchical, parallel and sequential execution for description of the communication rules and data transfer. However, for more complex communication systems VHDL tasks designer's mind with worthless problems - especially rules.

Chapter 3 points author's view on the problematic of suitable description tools. Very important conclusion: "... designers should not adapt the algorithm to the possibilities of the programming language," which just reveals permutation algorithm written in C. Chapter 4 presents M-redundancy concept based on the fact that redundant subsystems are connected together by a serial industrial communication bus. Following subchapters describe tools for validation of this concept and besides standard reliability function based on exponential failure law the software reliability model is presented - this work is based on Yu's [YuSh98] and Compton's model [Comp90] as well as on Halstead's earlier work [Hals77]. Reliability model of the industrial buses is mostly based on well known and respected standard MIL-HDBK-217 [Milh79]. Chapter 5 introduces basic terminology in the sphere of the formal languages and explains warranties correctness of the UPPAAL tool for our purposes. Chapter 6 reveals flowchart of the project's management that is strictly necessary if we accept M-redundancy concept and this flowchart is explained step-by-step on the example. Measurable effect of this concept is comparison of reliability of the system designed by standard way - see equation (6-3), where $R(100h) = 0.9$; M-redundancy way without formal verification - see equation (6-13), where $R(100h) = 0.9457$ and M-redundancy way with formal verification - see equation (6-14), where $R(100h) = 0.9999$. Chapter 7 deals with formal model of the heterogeneous industrial communication bus - Profibus DP for wire and wireless transfer medium [Zezu03], [Hint03]. Author's model enables not only to study bit-level (physical level) flow of the information through both above

mentioned transfer medium but also to stimulate this flow by predefined, random or backward error function and so simulate the problems of electromagnetic compatibility which is unique. This resesrch is not finished yet; detailed simulations or graphics user interface have to be realized.

The future of the design of the industrial communication system is in UML as the industry-standard language for specifying, visualizing, constructing, and documenting the parts of software systems, because it simplifies the complex process of software design. However the problematic of HW design, especially for embedded industrial systems still remains and this work shows one of the possible solutions.

9. Appendix

9.1. SFC to STL transfer

Miroslaw Wcislik (TU Kielce) in [Wcis03] introduced the method of programming SFC diagrams in ladder. For instance, simultaneous sequence is performed in block showed in Figure 9-1.

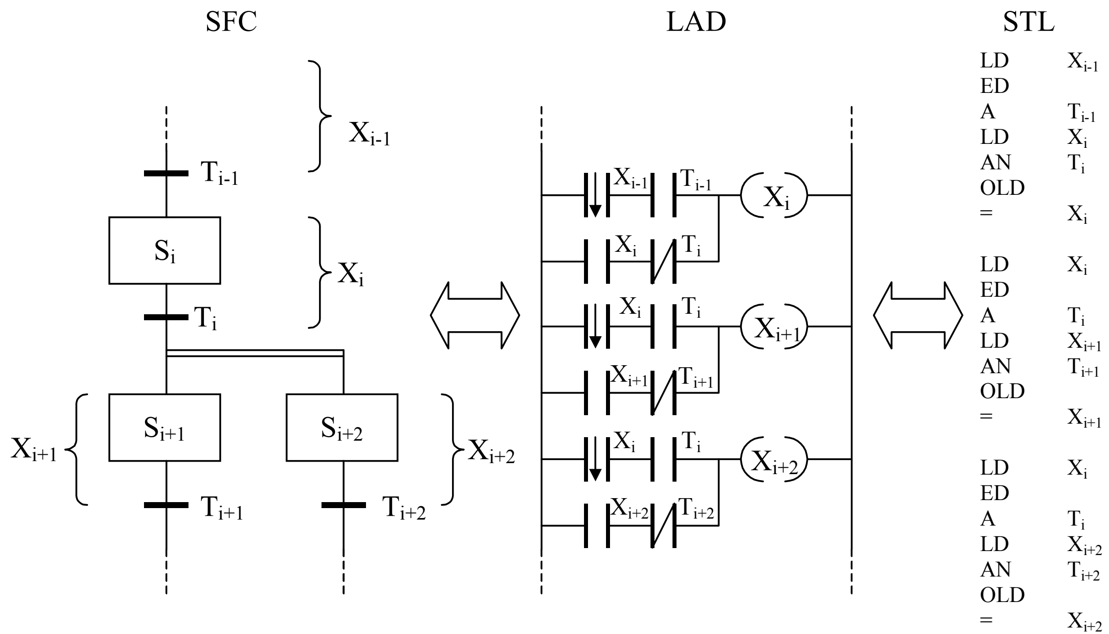

Figure 9-1: An example of the transfer SFC to LAD and STL

9.2. TLAKAN

Tlakan is a pressure analyser that was developed at the Brno University of Technology, Faculty of Electrical Engineering and Computer Science. This research has been funded by Grant Agency of the Czech Republic in the frame of the grant GACR 102/00/0938 - Tlakan and by the Czech Ministry of Education in the frame of the Research Intention JC MSM 262200012 - Research in information and control system.

Four departments were co-operating to design a measurement system that is based on the IEE 1451 standard [IEEE96], [IEEE97]. Developed system (Tlakan) allows scanning the pressure information from the system, using smart pressure sensors, standard Ethernet network and a network application (client server).

The Tlakan architecture is shown in Figure 9-2. and it consists of four basic components:

1. Analogue part that ensures signal processing of analogue pressure information.
2. STIM (Smart Transducer Interface Module - IEE 1451.2) processor. STIM transforms analogue values from sensors to their digital form according to the IEE 1451.2 specification (chapter 3.3 and 5). STIM also communicates with a NCAP (Network Capable Application Processor – IEEE 1451.1) using TII (Transducer Independent Interface – IEE 1451.2 – chapter 6).
3. NCAP communicates by TII with STIM on the side of sensors and on its network side by Ethernet-based Intranet protocol with CLIENT server.
4. CLIENT server representing a user application. This application periodically reads 10 pressure values and shows them on the screen of the PC.

From Figure 9-2 is evident, the Analogue part and STIM components belong to lower level, microelectronic domain while the NCAP and CLIENT components relate to upper level, application domain. Following section presumes essential familiarity of the Tlakan, entire description in [Kuce02] and ASL. ASL is Asynchronous

Specification language; author of this tool is Prof. Miroslav Sveda from the Department of Computer Systems, Faculty of Information Technology, Technical University of Brno. Corresponding references are [Sved01], [Sved02].

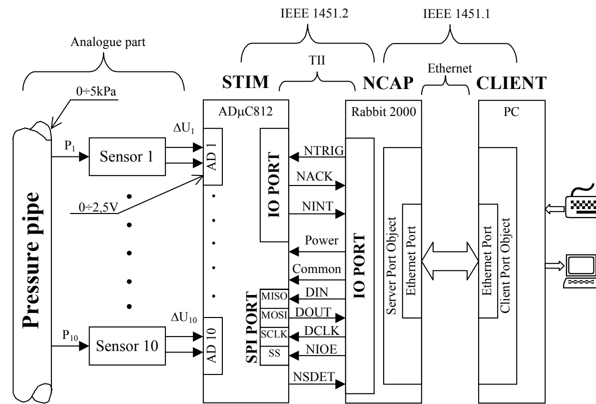

Figure 9-2: Tlakan's schematic diagram

Using standard engineering tools the Analogue part and IEEE 1451.2 part are described by tools according Table 9-1:

Table 9-1: Tlakan's description tools

Segment	*Tool*	*Remark*
Pressure Pipe	drawing documentation	CAD
Sensors (1-10)	drawing documentation, electric circuit, technical report	CAD, OrCAD,Text editor
STIM	electric circuit, user program, technical report	OrCad, Source code,Text editor
NCAP	electric circuit, user program, technical report	OrCad, Source code,Text editor

The ASL specification of Tlakan consists of four basic processes that describe behaviour of the system (this appendix deals with two of them).

Process STIM, which is started by NIOE signal from NCAP process, provides communication of the functional address and channel address during the write transfer protocol from the NCAP to the appropriate data space in STIM, and responses using the read transfer protocol (macros "bit_transfere" and "byte_transfere" are examples of the description of an electric signal on TII port of the STIM, functions "write_data()" and "read_data()" are functions that apply the same transfer protocol to write or read the information into/from the STIM data space).

Process TRIG, which is started by NTRIG signal from NCAP process, provides updating pressure information on selected channel.

Process NCAP provides communication functions on the network side of the processor, and TII functions on the side of the STIM (macros “bit_transfer” and byte_transfer” have the same meaning as macros in STIM process but respecting the NCAP as a Master device on TII interface, functions “Decode_inputs_arguments()”, “Encode_outputs_arguments()”, “MarshalArguments()” and “DemarshalArguments()” implement appropriate IEEE1451.1 functions in the communication object classes with Client-Server communication model).

Process CLIENT provides connecting the application to the NCAP using Ethernet network (it implements IEEE 1451.1 behaviour complementary to the NCAP as the master device)

Description of the process STIM and NCAP follows. Comments are typed italic in brackets {}. Keywords are typed bold.

process STIM(**is**: DIN, DCLK, NIOE;
os: NACK, NINT, DOUT):

{
This process is according to the IEEE 1451.2. Process header contains:
4 input signals - they follow “is:” statement and
3 output signals - they follow “out:” statement.
There are no inputs or outputs messages in header.
Following signals are defined in IEEE 1451.2 - 6.2
Signal DIN - address and data transport from NCAP to STIM
Signal DCLK - Positive-going edge latches data on both DIN and DOU
Signal NIOE - Signals that the data transport is active and delimits data transport framing
Signal NACK - Serves two function:
(1) Trigger acknowledge
(2) Data transport acknowledge

Signal NINT - Used by a STIM to request service from the NCAP
Signal DOUT - Data transport from STIM to NCAP
}

macro bit_transfer(received, send)
{ *This macro ensures bit data transfer protocol between NCAP and STIM. Data shall be transferred in bit-serial from the NCAP to the STIM via DIN - IEEE 1451.2 – 6.3.2* }

wait(DCLK,_);   { *DCLK idles high* }
wait(~DCLK,_);   { *On the falling edge of DCLK, the first bit to be transferred is* } DOUT = send;   { *asserted by STIM on DOUT* }
wait(~DCLK,_);
wait(DCLK,_);   { *On the subsequent rising edge of DCLK, the bit is latched by the* }
received = DIN;   { *STIM on DIN* }
macroend;

```
macro byte_transfer(byte_in, byte_out)
{
This macro ensures byte (8 bits) read transfer protocol from STIM to NCAP - IEEE 1451.2 – 6.3.4.
All data shall be transferred from STIM to NCAP in groups of 8 bits, using the bit transfer protocol -
macro bit_transfer.
}
    last:=NACK;  { State of the NACK signal is stored }
    i:=0;         { Number of transferred bits }
    byte_in:=0;   { Input byte is cleared }
    loop        { this loop ensures 8 bits transfer }
                bit_transfer(bit, (byte_out shr (7-i)) and 1); { macro writes corresponding bit in }
        byte_in:=(byte_in shl 1) + bit;           { byte_out to the NCAP and reads}
        i:=i+1;                   { corresponding bit to the byte_in }
        when i = 8 action exit;           { loop is exited upon 8 bits is transferred }
    endloop;
    if last then NACK:=FALSE;    { STIM negates NACK - IEEE 1451.2 - 6.3.5 - G}
    else NACK:=TRUE;      { If an odd numbers of bytes has been transferred, }
                                        { NACK will already be negated due to the byte read }
                                        { transfer protocol }
    fi;
macroend;

{ necessary declarations }
var last, received, send, bit: boolean;
var byte_in, byte_out, i, function, channel :integer;

        loop  { indefinite main cycle of the STIM process IEEE 1451.2 - 6.3.5}
          wait(NIOE,_);            { NIOE idles high }
                wait(~NIOE,_);      { NCAP asserted NIOE }
        NACK:=FALSE;     { STIM asserts NACK }
        byte_transfer(function,0);   { first function address is received IEEE 1451.2 -}
    byte_transfer(channel,0);    { 4.4.3, then channel address is received  4.4.2}
        if (function shr 7) and 1 write_data();
            else read_data();
                { According to IEEE 1451.2 - 4.4.3.1 is decoded the msb of the functional address }
                { and relevant operation  (read/write)  on the selected channel is performed. }
                { read_data/write_data are symbolical names for these operation and due to scope}
                { of this abstract are not described }
        fi;
    endloop;
endprocess;

process NCAP(   is: NACK,NINT,DOUT;
            os: NTRIG, DIN, DCLK, NIOE;
                ic: request;
                oc: response ):
{
This process is according to the IEEE 1451.2.
Signals NACK, NINT, DOUT, DIN, DCLK and NIOE are described in STIM process.
Signal NTRIG performs triggering function IEEE 1451.2 - 4.6 }
Input messages request and response are according to the IEEE 1451.1-8.2.3
}
macro bit_transfer(received, send)
{ This macro ensures bit  data transfer protocol between NCAP and STIM. Data shall be transferred in
bit-serial from the STIM to the NCAP via DOUT - IEEE 1451.2 – 6.3.2 }

    DCLK:=TRUE;  { DCLK idles high }
    wait(_, tH);        { Wait for tH time  - IEEE 1451.2 - 6.4}
    DIN:=send;        { Output bit is asserted on the DOUT  }
```

```
    DCLK:=FALSE; { Falling edge on the DCLK }
    wait(_,tL);     { Wait for tL time  - IEEE 1451.2 - 6.4 }
    received:=DOUT;    { Read bit from the STIM }
    DCLK:=TRUE;  { DCLK idles high }
macroend;

macro byte_transfer(byte_in, byte_out, success)
{
This macro ensures byte (8 bits) read transfer protocol from NCAP to STIM - IEEE 1451.2 – 6.3.3.
All data shall be transferred from NCAP to STIM  in groups of 8 bits, using the bit transfer protocol -
macro bit_transfer. The NCAP shall proceed with a byte transfer only after it observe a transmission on
the NACK line.
}
    last:=NACK;  { State of the NACK signal is stored }
    i:=0;         { Number of transferred bits }
    byte_in:=0;    { Input byte is cleared }
    loop         { this loop ensures 8 bits transfer }
        bit_transfer(bit,(byte_out shr (7-i)) and 1 );   { macro writes corresponding bit in }
        byte_in:=(byte_in shl 1) + bit;            { byte_out to the NCAP and reads}
        i := i + 1;                       { corresponding bit to the byte_in }
        when i = 8 action exit;        { loop is exited upon 8 bits is transferred }
    endloop;
    if last then               { NCAP waits for acknowledgement }
        wait(NCAK,tHOLD,_);
        wait(~NACK, tHOLD, success);
    else
        wait(~NACK, tHOLD,_);
                wait(NACK, tHOLD, success);
    fi;
macroend;

macro packet_transfer(buffer_in, buffer_out, length, success)
{ Macro ensures packet transfer between NCAP and STIM - IEEE 1451.2 - 6.3.5 and 6.3.6 }

    NIOE:=FALSE;       { NCAP asserts NIOE }
    wait(F_NACK, tHS, success);   { NCAP waits until the STIM asserts NACK }
    if success then
        i:=0;
        loop    { whole transfer buffer is transferred by byte_transfer macro }
            byte_transfere(buffer_in[i], buffer_out[i], success);
            i := i + 1;
            when not success or i = length action exit;
        endloop;
    fi;
    NIOE:=TRUE; { NCAP negates NIOE }
macroend;

macro trigger(success)
{ Macro ensures triggering IEEE 1451.2 - 6.4. , 4.6 and 6.3.1}
    NIOE:=TRUE;         { NIOE and NTRIG shall be true at the beginning of the triggering }
    NTRIG:=FALSE;
    wait(_,tWS);        { NCAP waits for the duration of Channel Write Setup Time tWS }
    NTRIG:=FALSE;     { Then NCAP asserts NTRIG }
         wait(NACK,tU,_);
         wait(~NACK, tU, success);    { NCAP waits until STIM negates NACK }
    if success then
        NTRIG:=TRUE;
        wait(~NACK,tH,_);
                wait(NACK, tHS, success);
```

```
        wait(_,tRS);
                NIOE:=FALSE;
          else                  { STIM fails - NTRIG and NIOE to their initial conditions }

        NTRIG:=TRUE;
        NIOE:=FALSE;
    fi;
macroend;

macro IORead(channel, result, success)
{ Macro ensures basic operation with the measuring channels }
    buffer_out[1] = 3;          {Trigger channel address}
    buffer_out[2] = channel; {actual channel}
    packet_transfere(_,buffer_out,2,success);
    if success
      trigger(success);
        if success
          buffer_out[1] = 128; {Read from channel}
          buffer_out[2] = channel;
          packet_transfere(buffer_in,buffer_out,4,success);
          if success
              result = buffer_out;
          fi;
            fi;
        fi;
macroend;

{ necessary declarations according to IEEE 1451.2 - 6.4 }
const fclk = 10 000;
const tH = 0.5/fclk;
const tL = 0.5/fclk;
const tHOLD = 500e-6; {IEEE 1451.2 - 5.1.3.21}
const tHS = 10e-3; {IEEE 1451.2 - 5.1.3.19}
const tRS = 80e-6; {IEEE 1451.2 - 5.2.3.23}
const tWS = 80e-6; {IEEE 1451.2 - 5.2.3.22}
const tU = 200e-6; {IEEE 1451.2 - 5.2.3.21}
var request, response: ArgumentArray;
var success, received, send, last, bit:boolean;
var byte_in, byte_out, length, channel: integer;
type buffer = array[1..3] of UInteger8;          {IEEE 1451.1-6.1.1}
var result:Float32;
var buffer_in, buffer_out:buffer;
  loop
    wait(request,_);
    {IEEE 1451.1-8.2.3.5 - Ethernet}
    DemarshalArguments(request, server_operation_id, server_inputs_arguments);
    if server_operation_id = READ_VALUE then
        Decode_inputs_arguments(server_inputs_arguments, channel);
        IORead(channel, result, success);
        if success
            Encode_outputs_arguments(result,server_outputs_arguments);
            MarshalArguments(server_outputs_arguments, response);
            send(response, CLIENT);
        fi;
    fi;
  endloop;
endprocess;
```

From the ASL description of the processes STIM and NCAP is evident that the same technique is used for information interchange in Client/Server model and for SPI communication between NCAP and STIM processors.

10. Glossary

Acknowledge Frame

Conveys the status of a transaction from the remote (responding) *FDL* entity to the local (initiating) *FDL* entity [PNOA91].

Action Frame A data or request frame, the first frame transmitted in all transactions [PNOA91].

ASL Asynchronous Specification Language - Prof. Sveda's description tool for embedded and distributed systems [Sved02], [Sved03].

Bathtub curve the variation of the failure rate with time for electronic components. The failure rate is assumed to be constant during the useful life of the component.

BaudRate is a data transmission rate (bits/second) for serial communication line.

Combinatorial Models

a method of developing an analytical expression for a system's reliability.

CW Conventional Wiring.

DET Data Exchange Table - IOT for Profibus DP cyclic transfer [TLPK02].

DLOC Deliverable Lines of Code - total number of text lines in a program, including comments and blank lines.

EMC ElectroMagnetic Compatibility.

EMI ElectroMagnetic Interference.

Exponential Failure Law

a relationship whereby reliability varies exponentially with time.

Failure rate is the expected number of failures of a type of device or system per a given time period [Shoo68].

Fault Coverage

a generic term used for the probability of fault recovery given that a fault has occurred.

FDL Fieldbus Data Link layer - ISO/OSI layer 2.

Frame the packet of data transmitted on the bus [PNOA91].

FT Fault Tolerant.

Hardware redundancy

is the addition of extra hardware, usually for the purpose of either detecting or tolerating faults [John84].

IL Instruction List - EC61131-3 standard PLC controller programming language.

I/O Inputs/Outputs.

IOT InputOutput Table - memory storage for peripheral I/O in the PLC.

ISO International Organization for Standardization.

LOC Lines Of Code - all lines of code excluding comment and blank lines.

M-of-N System

a system in which *M* out of *N* components must operate correctly for the system operate correctly.

MTBF Mean Time Between Failure is the average time between failures of a system. *N* systems are operated for some time *T* and number of failures encountered by the ith system is recorded as n_i. The average number of failures n_{avg} is computed as [Milh79]:

$$n_{avg} = \sum_{i=1}^{N} \frac{n_i}{N}, \qquad (9\text{-}1)$$

then MTBF is

$$MTBF = \frac{T}{n_{avg}}. \qquad (9\text{-}2)$$

NHPP Nonhomogeneou Poissson process.

OSI Open System Interconnection.

PLC Programmable Logic Controller.

Profibus DP Decentralised Peripherical fieldbus is based on the European standard EN 50170 and it is progressive and speed industrial network that makes possibility to exchange blocks of information between network partners (PLC, distributed I/O) at the device level [PNOA91], [PNOB91], [PNOC94].

Redundancy is the addition of information, resources, or time beyond what is needed for normal system operation [John84].

Reliability is defined as the conditional probability that the system will perform correctly throughout the interval <t_0, t>, given that the system was performing correctly at time t_0 [Shoo68].

Replay Frame a response or *Acknowledge Frame*, this frame completes a confirmed transaction [PNOA91].

Response Frame

a *Replay Frame* that carries a request for *Request Data* from a remote *FDL* user to a local *FDL* user [PNOA91].

Request a request primitive e.g. passes a request, to send *Send Data* and/or request *Request Data* or load *updata*, from the local *FDL* user to the local *FDL* entity [PNOA91].

Request Data a data provided by the remote *FDL* user to the local *FDL* user [PNOA91].

Request Frame

an *Action Frame* that carries a request for *Request Data* from a local *FDL* user to a remote *FDL* user [PNOA91].

Send Data data provided by a local *FDL* user to a remote *FDL* user [PNOA91].

SFC **S**equential Function Chart - EC61131-3 standard PLC controller programming language.

Software redundancy

is the addition of extra software, beyond what is needed to perform a given function, to detect and possibly tolerate faults [Chen78].

SRGM Software Reliability Growth Model.

STL Statement list - programming language for Siemen's PLCs.

Testability the ability to verify that a system is operating correctly.

TBit BIT Time, FDL symbol period - the time to transmit one bit on the Profibus [PNOA91].

T_{MC} the time between transmission of the first bit of an action frame and receipt of the last bit of the corresponding reply frame [PNOA91].

TMR Triple Modular Redundancy, the basic concept is to triplicate the SW/HW and perform a majority vote to determine the output of the system.

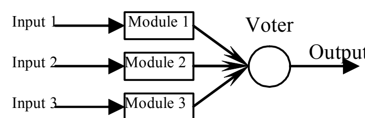

Figure 10-1: Principle of the TMR

T_R request *frame* Time - the time to transmit a request *frame* [PNOA91].

T_S send *frame* Time - the time to transmit a send *frame* [PNOA91].

T_{SDR} station Delay of Responser - the actual time this responser waits before generation a reply frame [PNOA91].

UML the Unified Modelling Language is the industry-standard language for specifying, visualizing, constructing, and documenting the artefacts of software systems.

UPPAAL an integrated tool environment for modelling, validation and verification of real-time systems modelled as networks of timed automata, extended with data types [Pett00].

VHDL *t*he VHSIC Hardware Description Language (VHDL) is a formal notation intended for use in all phases of the creation of electronic systems.

WAGO decentralised I/O system. It saves material cost when the control system is connecting to the application. Also it increases reliability of the entire control system. Reducing number of wires decreases the probability of wrong connection (see equation 4-2) or interference of the signal [Kuce01].

References

[Adam01] 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, 1999.

[Akiy71] Akiyama, F.: An example of software system debugging. Proceedings of IIFIP Congress, 1971.

[Alur94] Alur, R. and Dill, D.L.: A theory of timed automata. Theoretical Computer Science, 1994.

[Aviz84] Avizienis, A. and Kelly, J.P.J.: Fault Tolerance by design diversity: concepts and experiments. Computer, Vol 17, No.8, August 1984.

[Behr01] Behrmann, G., Fehnker, A., Hune, T.S., Larsen, K., Petterson, P. and Romijn, J.: Effcient guiding towards cost-optimality in UPPAAL. In Proceedings of TACAS'2001, Lecture Notes in Computer Science. Springer-Verlag, 2001.

[Beng95] Bengtsson, J., Larsen K.G., Larsson F., Pettersson, P. and Yi, W.: Uppaal - a Tool Suite for Automatic Verification of Real-Time Systems. In Proceedings of Workshop on Verification and Control of Hybrid Systems III, number 1066 in Lecture Notes in Computer Science, Springer-Verlag, October 1995.

[Brad02] Bradac, Z., Fojtik, P., Cach, P., Fiedler, P., Zezulka, F., Kucera, P.: Modularni systemy sberu a prenosu procesnich dat. Konference Pragoregula-Elexpo 2002. Praha, Masarykova akademie prace, strojni spolecnost na CVUT, 2002.

[Brat95] Bratko, I.: Prolog Programming for Artificial Intelligence (International Computer Science Series), Addison-Wesley, Published August 2000.

[Brei95] Breijo, E., G., Civera, J., I. and Sanchez, L. G.: PSpice Simulacion y Analisis de Circuitos Analogicos asistida por Ordenador. Editorial Paraninfo sa, 1995.

[Chen78] Chen, L. and Aviziens, A.: N-version programming: A fault tolerant approach to reliability of software operation. Proceedings of the International Symposium on Fault Tolerant Computing, 1978.

[Clar86] Clarke, E.M., Emerson, E.A., and Sistla, A.P.: Automatic Verification of Finite-State Concurrent Systems Using Tem-poral Logic Specifications. ACM Transactions on Programming Languages and Systems 8(2). April, 1986.

[Clar87] Clarke, E. M. and Grumberg, O.: Research on Automatic Verification of Finite-State Concurrent Systems. Ann. Rev. Computer Science(2), 1987.

[Comp90] Compton, B., and Withrow, C.: Prediction of Control of ADA Software Defects. J. System Software 12, 1990.

[David03] David, A., Behrmann, G., Larsen, K.G. and Yi, W.: A tool Architecture for the next generation of UPPAAL. Technical report, Uppsala university, Sweden, February 2003.

[Este87] ISO: Estelle: A formal description technique based on an extended state transition model. International Standard ISO, IS 9074, 1987.

[Exup00] Saint-Exupery, A.: The Little Prince. Harcourt; 1st edition, May 2000.

[Goel79] Goel, A., L., and Okumoto, K.: Time-dependent error-detection rate model for software reliability and other performance measures. IEEE Transactions Reliability, R28, 1979.

[Hals77] Halstead, M., H.: Elements of Software science, NY, Elsevier, 1977.

[Haye85] Hayes, J. P.: Fault modeling, IEEE Design and Test, Vol. 2, No2, April 1985

[Hero95] Herout, P.: Ucebnice jazyka C 1.dil, Kopp, Ceske Budejovice, 1995.

[Honz02] Honzik, P., Kucera, P.: Atributes of internet technologies and their rating in decision making process: II Scientific conference "Telecommunication in XXI century". Kielce University of Technology, 2002.

[Hint01] Hintze E. and Donath, U.: Schnittstellenspezifikation; ifak e.V. Magdeburg, 2001.

[Hint03] Hintze, E. and Kucera, P.: Simulation of RFieldbus Networks. th IFAC International Conference on Fieldbus Systems and their Applications. Aveiro, 2003.

[Hugh68] Hughes, G.E. and Cresswell, M.J.: An Introduction to Modal Logic. Methuen and Co., New York, NY, 1968.

[Hune00] Hune, T., Larsen, K.G. and Pettersson, P.: Guided Synthesis of Control Programs Using UPPAAL. In Ten H. Lai, editor, Proceedings of the IEEE ICDCS International Workshop on Distributed Systems Verification and Validation, IEEE Computer Society Press, April 2000.

[IEEE96] IEEE P1451.1 D1.83, Draft Standard for a Smart Transducer Interface for Sensors and Actuators. Network Capable Application Processor (NCAP) Information Model, IEEE, New York, December 1996.

[IEEE97] IEEE P1451.2 D3.05, Draft Standard for a Smart Transducer Interface for Sensors and Actuators. Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEEDS) Formats, IEEE, New York, August 1997

[John84] Johnson, B.W.: Fault-tolerant microprocessor-based systems, IEEE Micro, Vol.4, No.6, December 1984.

[Koul02] Koulamas C., Koubias S.: Real-Time Characteristics of the RFieldbus Link Device Bridging Operation, 2002.

[Kuce01] Kucera, P.: Fault tolerant design of control. IWCIT'01, VSB Ostrava, 2001.

[Kuce02] Kucera, P., Zezulka, F., Sveda, M., Vrba, R.: Executable specification for Process Automation and Microelectronics. IEEE TC-ECBS and IFIP WG10.1 Joint Workshop on Formal Specifications of Computer-Based Systems. IEEE TC-ECBS and IFIP WG10.1 Joint Workshop on Formal Specifications of Computer-Based Systems. Lund, 2002.

[Kuce03] Kucera, P., Honzik, P., Zezulka, F.: Virtual Control of Robot. Programmable devices and systems. IFAC Workshop on Programmable Devices and Systems - PDS 2003. Ostrava, 2003.

[KucP03] Kucera, P.: Formal methods and industrial communication. Student EEICT 2003. VUT Brno, 2003.

[KuZe03] Kucera, P., Zezulka, F. and Bradac, F.: Formal Method for higher reliability of industrial automation. ICIT'03 International Conference on Industrial Technology, Slovenia 2003.

[Lamp83] Lamport, L.: What Good is Temporal Logic. Information Processing 83, 1983.

[Lamp86] Lamport, L. A: Simple Approach to Specifying Concurrent Systems. Technical Report, DEC, December 1986.

[Lars98] Larsen, K. G., Pettersson, P. and Yi, W.: UPPAAL in a Nutshell. In Interantional Journal on Software Tools for Technology Transfer 1(1-2), Springer-Verlag, 1998.

[Lipo82] Lipow, M.: Number of Faults per Line of Code. IEEE Transaction Software Engineering, 1982.

[LVPK02] Bradac, Z., Fieldler, P., Zezulka, F., Fojtik, P., Cach, P. and Kucera, P.: Laborator pro vyuku prumyslovych komunikacnich sbernic. Automa, ISSN 1210-9592, roc. 8, c. 5, 2002.

[Milh79] United States, Department of Defense, Military Standardization Handbook: Reliability Prediction of Electronic Equipment, MIL-HDBK-217, B, C, 1965, 1974, 1979.

[Mose90] Moser, L.E. and Melliar-Smith, P.M.: Formal verification of Safety-Crytical Systems. Software. Practice and Experience, 20(8), August 1990.

[Musa80] Musa, J., D.: Software Reliability measurement. J. System Software 1, 1980.

[Musa87] Musa, J., D., Iannino, A., Okumoto, K.: Software Reliability: Measurement Prediction Application. McGraw-Hill, New York, 1987.

[Otte79] Ottenstein, L., M.: Quantitative Estimates of Debugging Requirements. IEEE Transaction Software Engineering, 1979.

[Pett00] Pettersson, P. and Larsen, K.G.: Uppaal2k. In Bulletin of the European Association for Theoretical Computer Science, volume 70, 2000.

[Pham00] Pham, H.: Software reliability. Springer-Verlag, 2000.

[PNOA91] PROFIBUS Nutzerorganisation e.V., DIN 19 245 Part 1: Process Field Bus, Wesseling, Germany, April 1991.

[PNOB91] PROFIBUS Nutzerorganisation e.V., DIN 19 245 Part 2: Process Field Bus, Wesseling, Germany, April 1991.

[PNOC94] PROFIBUS Nutzerorganisation e.V., DIN 19 245 Part 3: Process Field Bus DP, Karlsruhe, Germany, 1994.

[Pnue85] Pnueli, A.: Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends. Lecture Notes in Computer Science #224, 1985.

[Resc71] Rescher, N. and Urquhart, A.: Temporal Logic. Springer-Verlag, Wien, New York, 1971.

[Shoo68] Shooman, M.L.: Probabilistic Reliability: An Engineering Approach, McGraw-Hill, New York, 1968.

[Siem96] Siemens AG: SIMATIC Software, System Software for S7-300 and S7-400, Program Design, Nurnberg, 1996.

[Siew82] Siewiorek, D.P and Swarz, R.Z.: The theory and Practise od Reliable System design. Digital Press, Bredford, Mass., 1982.

[Skib99] Skibinski, G., Kerkman, R. and Schlegel D.: Electromagnetic Emissions of Modern PWM AC Drives. IEEE Industry Applications Magazine, 1999.

[Sved01] Sveda, M., Vrba, R.: Executable Specifications for Distributed Embedded Systems. IEEE Computer, Vol. 34, No.1, Los Alamitos, CA, US 2001.

[Sved02] Sveda, M.: Rapid Prototyping of Embedded Distributed Systems. IEEE Design and Diagnostics of Electronic Circuits and Systems. Brno, FIT VUT, 2002.

[TLPK02] Bradac, Z., Zezulka, F., Fieldler, P., Fojtik, P., Cach, P. and Kucera, P.: Treninkova laborator prumyslovych komunikacnich siti. Pragoregula-Elexpo 2002. Praha: Masarykova akademie práce, strojní společnost na ČVUT, 2002.

[Tria95] Triantafyllos, G., Vassiliadis, S. and Kobrosly, W.: On the prediction of computer implementation faults via static error prediction models. Systems and Software, Volume 28, Issue 2, February 1995.

[Trie92] Triebel, W., A.: The 80386DX Microprocessor, hardware, software and Interfacing, prentice-Hall International Editions, 1992.

[Vrat97] Vratil, Z.: Assembler PC, Gethon audio and computer, Sokolov, 1997.

[Wcis03] Wcislik, W.: Programming of sequential system in ladder diagram language. Preprints of IFAC workshop on programmablke devices and Systems, PDS2003, VSB Ostrava, 2003.

[Yovi97] Yovine, S.: A verification tool for real time systems. In International Jornal on Software Tools for Technology, October 1997.

[Zezu01] Zezulka, F., Kucera, P. and Bradac, Z.: Uvod do problematiky prumyslovych komunikacnich siti. AT&P Journal, 2001.

[ZezK01] Zezulka, F., Kucera, P., Fojtik, P. and Bradac, Z.: Technicka reseni nejbeznejsich prumyslovych komunikacnich sbernic. AT&P Journal, 2001.

[Zezu02] Zezulka, F., Bradac, Z., Fiedler, P., Cach, P., Kucera, P. and Fojtik, P. Laborator pro vyuku prumyslovych komunikacnich sbernic. Automa, 2002.

[Zezu03] Zezulka, F., Hintze, E., Kucera, P.: Formal methods in fieldbus specification: SCI 2003 Conference. The 7th World Multiconference on Systemics, Cybernetics and Informatics. Orlando, 2003.

11. Curriculum Vitae

Name: Pavel KUČERA

Born: 12th of August, 1977 in Valtice

Contact: kucera@feec.vutbr.cz

Education

1995 – 2000 **Technical University of Brno / Department of Control and Instrumentation, MS.C. - Low cost control by SIMATIC S7.**

2000 – 2003 **Technical University of Brno / Department of Control and Instrumentation**
Ph.D. study of Cybernetic and informatics, State exam passed in August 2002

Languages

English, Spanish

Experiences

May 1999 KH Limburg, Department of Industrial Sciences and Technology - Engineering.

February - July 2001 The University of Huddersfield, School of Computing & Engineering, Electronics and Communications Research Group.
Final Report: The Tiny Home Automation System.

December 2002 Kielce University of Technology, Faculty of electrical and computer engineering.
Lecture: Formal Method in an Industrial Communication Problem.

September 2002 - February 2003

Institut für Automation und Kommunikation e.V. Magdeburg, The Department of Communication Systems.

Final Report: Profibus DP - VHDL BUS Model.