

Formal Method in an Industrial Communication Problem

P. Kučera, P. Honzík and F. Zezulka
Dept. of Control and Instrumentation
Faculty of Electrical Engineering and Communication
Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech republic

Email: kucera@feec.vutbr.cz; honzikp@feec.vutbr.cz; zezulka@feec.vutbr.cz

Tel: ++420 5 4114 1308

Fax: ++450 5 4114 1123

Keywords: Formal Methods, Control System, Industrial Communication, VHDL

Abstract:

Industrial communication plays an important role in industrial automation due to trend of decentralisation of 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 contribution should show the reader one of the possible ways to solve this problem - using Formal Methods. The reader is familiarized with the railway model, control elements and communication bus. Suitable formal method is applied with the aim to check entire system.

1. System

Complete architecture of the system is shown in Figure 1.

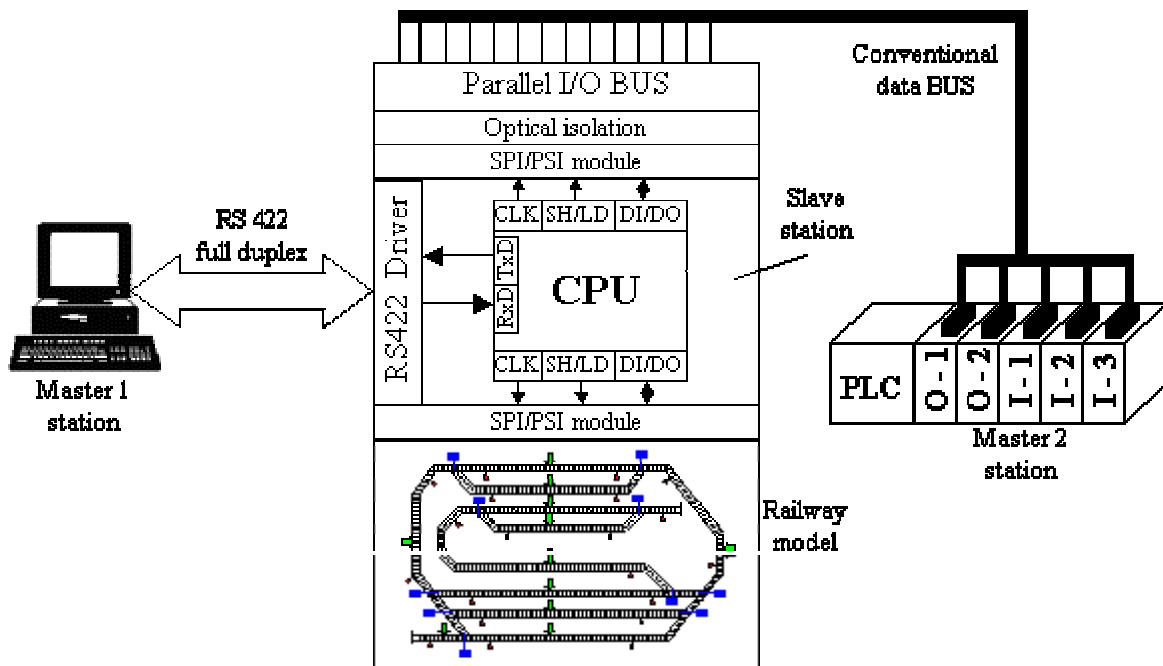


Fig 1 - Architecture of the system

System consists of 4 main parts:

1. Railway model - serves as physical interface between human and designed control structure. Model has approximately 70 inputs and outputs including signals for switches and fast action sensors. Detailed information is in [5].
2. Slave Station. This part of the system plays the most important rule. Slave station ensures interface between Railway model and master stations 1 and 2. Also control Railway model according to the desired function.
3. Master station 1 is PC with RS 422 serial interface.
4. Master station 2 is PLC linked with Slave station via conventional data bus.

Main idea of the system is to fault tolerant control of model. Therefore two Master stations are connected via diametrically physical interfaces - serial line and conventional parallel data bus. Inside both Master stations runs application, that ensures basic information exchange between appropriate master and Slave station. This information exchange is based on developed protocol FTCP (Fault Tolerant Control Protocol) that ensures correct behaviour of the system if unexpected events are occurred.

2. Verification

System consists of hardware part (RS 422 drivers, PLC interface, Parallel data BUS, SPI/PSI modules ...) and software part (Applications in CPUs of Slave station and Masters stations). It is not problem to describe, design, analyse and verify electronic and software part of the system separately but then we have no guarantee about implementation, validation and verification of the whole system. One of the ways to solve this problem is to use the same prototyping tool for the software and hardware part of the project.

To describe hardware part of the system by VHDL is not problem. But if we accept above-mentioned consideration then we must use VHDL for description of software behaviour of the elements too.

3. Communication model

Running of the system is separated into two basic phases:

1. Initialisation
2. Execution

During Initialisation phase Master station 1 or 2 tries to determine what Master station will control the system. There are three possibilities - Master 1 or Master 2 or both control system at the same time. During Execution phase appropriate Master station(s) controls system. Header of message from Master Station to the Slave station is an enumerate type:

type T_MFM is (STOP, INI, TAKE, HAND, JOIN, REQ, DATA);

- STOP - During Execution phase master blocks all drives. This message is prohibited during Initialisation phase.
- INI - During Execution or Initialisation phase master invokes/restarts Initialisation phase.
- TAKE - During Initialisation phase master tries to take up control. During Execution phase this message invokes Initialisation phase.
- HAND - During Initialisation phase master offers control to the second master. During Execution phase this message invokes Initialisation phase.

- JOIN - During Initialisation phase master tries to establish join control. During Execution phase this message invokes Initialisation phase.
- REQ - During Initialisation or Execution phase master requests data from Slave Station.
- DATA - During Execution phase master sends data to the Slave station. This message is prohibited during Initialisation phase.

Complete list of defines type is:

```

type T_MFM is (STOP, INI, TAKE, HAND, JOIN, REQ, DATA);
type T_MTM is (ERROR, JOIN, DATA, ACK);
type T_IOSTATE is (NONE, M1M2, M2M1, M1JOIN, M2JOIN, M1CNTRL, M2CNTRL, M1M2CNTRL);
subtype T_BYTE is bit_vector (7 downto 0);
constant ACKINI: bit_vector (47 downto 0) := X"AA0000000000";
constant ACKTAKE: bit_vector (47 downto 0) := X"BB0000000000";
constant ACKHAND: bit_vector (47 downto 0) := X"CC0000000000";
constant ACKJOIN: bit_vector (47 downto 0) := X"DD0000000000";
constant ACKSTOP: bit_vector (47 downto 0) := X"EE0000000000";
constant ACKDATA: bit_vector (47 downto 0) := X"FF0000000000";
type T_RFM is
  record
    MESSAGE: T_MFM;
    DATA: bit_vector (31 downto 0);
    CRC: T_BYTE;
  end record;
type T_RTM is
  record
    MESSAGE: T_MTM;
    DATA: bit_vector (47 downto 0);
    CRC: T_BYTE;
  end record;

```

Communication model is separated into two entities - **rail** (behaviour of the Slave station) and **test_rail** (behaviour of the Master station).

```

entity rail is
  port(RFM1: in T_RFM;           -- Message from master 1
        RFM2: in T_RFM;           -- Message from master 2
        SRTM1: out T_RTM;         -- Message to Master 1
        SRTM2: out T_RTM;         -- Message to Master 2
        ACTION: out bit_vector (31 downto 0); -- Action values
        READ: in bit_vector (47 downto 0)); -- Measurement values
end rail ;

```

There is only one architecture of the entity rail:

```

architecture rail of rail is
  signal EBM1, EBM2 : boolean := false;           -- Error flag of Master 1 and 2 BUS
  shared variable STATE: T_IOSTATE;              -- State of Slave station
  shared variable L_ACTION: bit_vector (31 downto 0);
  shared variable RTM1: T_RTM;                   -- Temporal message to Master 1
  shared variable RTM2: T_RTM;                   -- Temporal message to Master 2
begin
  IO1: process(RFM1,EBM1) -- Process ensures information exchange between Master 1 station and Slave station
  begin
    if (STATE <= M2JOIN) then
      RTM1.MESSAGE := ERROR;
      RTM2.MESSAGE := ERROR;
      L_ACTION := X"00000000";
      if (not EBM1) then
        case RFM1.MESSAGE is

```

```

when INI => RTM1.MESSAGE := ACK;
           RTM1.CRC := CRC(ACKINI);
           RTM1.DATA := ACKINI;
           state := NONE;
when REQ => RTM1.MESSAGE := DATA;
           RTM1.CRC := CRC(READ);
           RTM1.DATA := READ;
when TAKE => if (STATE = NONE or STATE = M2M1) then
             state := M1CNTRL;
             RTM1.MESSAGE := ACK;
             RTM1.CRC := CRC(ACKTAKE);
             RTM1.DATA := ACKTAKE;
           else state := NONE;
           end if;
when HAND => if (STATE = NONE) then
             state := M1M2;
             RTM1.MESSAGE := ACK;
             RTM1.CRC := CRC(ACKHAND);
             RTM1.DATA := ACKHAND;
           else state := NONE;
           end if;
when JOIN => if (STATE = M2JOIN) then
             state := M1M2CNTRL;
             RTM1.MESSAGE := ACK;
             RTM1.CRC := CRC(ACKJOIN);
             RTM1.DATA := ACKJOIN;
           else RTM2.MESSAGE := JOIN;
             state := M1JOIN;
           end if;
when others => assert true
             report "Master 1 inccorect action during initialization"
             severity error;
end case; --Master 1 message service
else
  assert true
  report "MASTER 1 BUS ERROR"
  severity error;
end if; -- Branch JOIN
else
  if ( EBM1 ) then state := NONE;
  else
    case STATE is
      when M1CNTRL => case RFM1.MESSAGE is
        when STOP => L_ACTION := X"00000000";
                    RTM1.MESSAGE := ACK;
                    RTM1.CRC := CRC(ACKSTOP);
                    RTM1.DATA := ACKSTOP;
        when INI => RTM1.MESSAGE := ACK;
                    RTM1.CRC := CRC(ACKINI);
                    RTM1.DATA := ACKINI;
                    state := NONE;
        when TAKE | HAND | JOIN => state := NONE;
        when REQ => RTM1.MESSAGE := DATA;
                    RTM1.CRC := CRC(READ);
                    RTM1.DATA := READ;
        when DATA => L_ACTION := RFM1.DATA;
                    RTM1.MESSAGE := ACK;
                    RTM1.CRC := CRC(ACKDATA);
                    RTM1.DATA := ACKDATA;
        when others => assert true
                    report "IMFMaster 1 - ACTION"
                    severity error;
      end case;
      when M2CNTRL => assert true
                    report "Master 2 has not access right."
                    severity error;
      when M1M2CNTRL => if ( RFM1.DATA = RFM2.DATA ) then

```

```

        L_ACTION := RFM1.DATA;
        RTM2.MESSAGE := ACK;
        RTM1.MESSAGE := ACK;
        RTM2.CRC := CRC(ACKDATA);
        RTM1.CRC := CRC(ACKDATA);
        RTM2.DATA := ACKDATA;
        RTM1.DATA := ACKDATA;
    else
        assert true
        report "Master1 \= Master2 - ACTION"
        severity note;
    end if;
when others => assert true
    report "incorrect STATE during ACTION"
    severity failure;
end case;--State machine
end if; --ERROR decision
end if; --Main decision
end process IO1;
--
-- process IO2 definition has the same structure like IO1 and is omitted
--
    EBM1 <= true when RFM1.CRC /= CRC(RFM1.DATA) else
        false;
    EBM2 <= true when RFM2.CRC /= CRC(RFM2.DATA) else
        false;
    ACTION <= L_ACTION;
    SRTM1 <= RTM1;
    SRTM2 <= RTM2;
    ST <= STATE;
end rail;

```

Entity **test_rail** stimulate entity **rail** by desired signals from user-defined file. Process of testing is out of scope of this paper; therefore a list of this entity is not included.

4. Conclusion

This paper presents a possibility how to describe software part of the communication system by hardware description tool. The effort is returned by fact, that we can simulate and validate whole system by one simulation tool. Of course, our communication model is only first approximation of the reality because many things were simplified or omitted.

This paper was supported by Ministry of Education, Youth and Sports by research intents JC MSM 262200012 Research of Information and Control Systems, JC MSM 262200022 Research of microelectronic systems and technologies, grants FRVS G1/1969/2002, FRVS G1/1967/2002, FRVS G1/1955/2002, FRVS G1/1961/2002. The research was also supported by FEEC, Brno University of Technology.

- [1] Johnson, B.W „Fault-tolerant microprocessor-based systems”, IEEE Micro, Vol.4, No.6, Dec. 1984
- [2] Chen, L., A. Aviziens. „N-version programming: A fault tolerant approach to reliability of software operation”, Proceedings of the International Symposium on Fault Tolerant Computing, 1978
- [3] Kohavi, Z. Switching and Finite Automata Theory
- [4] Kučera P., Zezulka F., Švéda M., Vrba R.: Executable specification for Process Automation and Microelectronics, In: IEEE TC-ECBS and IFIP WG10.1 Joint Workshop on Formal Specifications of Computer-Based Systems, Lund, SE, US, 2002, p. 91-98, ISBN 1-85769-169-5

- [5] Kučera P.: Fault tolerant design of control In IWCIT'01. International Workshop Control and Information Technology, IWCIT'01 Ostrava: VŠB Ostrava, 2001, s. 149-154, ISBN 80-7078-907-7
- [6] Craigen, D., Gerhart S., Ralston T., "An International Survey of Industrial Applications of Formal Methods," Volume 1 Purpose, Approach, Analysis and Conclusions; Volume 2 Case Studies NIST GCR 93/626, March 1993.