VHDL MODEL OF ELECTRONIC-LOCK SYSTEM

Karel VLČEK[1], Brian R. BANNISTER[2], David MIKLÍK[1], Ian M. BELL[2], Ernst BARTSCH[2] and Jiří NOGA[1]
[1]Department of Measurement and Control, FEI, VŠB Technical University of Ostrava, Czech Republic
[2]Department of Electronic Engineering, University of Hull, Great Britain

Abstract

The paper describes the design of an electronic-lock system which was completed as part of the Basic VHDL course in the Department of Control and Measurement, Faculty of Electrical Engineering and Informatics, Technical University of Ostrava, Czech Republic in co-operation with the Department of Electronic Engineering, University of Hull, Great Britain in the frame of TEMPUS project no. S_JEP/09468-95.

Keywords

System Design, Finite State Machine, Behavioural VHDL Modelling, FPGA Implementation

1. Behavioural Modelling Method

The solution was achieved by the use of a finite state machine model which described the behaviour of the electronic-lock. The required circuit was specified in terms of partial behavioural models which were then simulated, and the final circuit achieved by the interconnection of these partial models.

This method of design allowed the designers the freedom to modify the overall model description of the system as necessary, and to implement it in a chosen FPGA.

2. System Design

The electronic-lock is often introduced to illustrate the possibilities of design of an electronic system using a "top-down" approach. This is because it is not too complicated but must be designed as a complete system. The "customer" must be able to specify the key data sequence and the sequence must be easily modified when required.

As designs become more complex, it becomes more efficient to move away from ad hoc methods and use tools that allow the design to be carried out at a higher level of abstraction [1]. The description of the circuit by VHDL [2] allows the designers to prepare a behavioural algorithm and does not require the definition of a data sequence for the electronic key.

The key is subsequently specified by the user as a sequence of constants, which are used as declarations in the program description. Input of the electronic key is by means of a twelve contacts telephone number pad. When a key is activated, one of four "row" contacts is connected to one of three "column" contacts. The rows and columns are connected to a 2-out-of-5 encoder.

The bus *CD* is the output of the encoder, giving a non-linear five-bit error control code. This is applied to a decoder and decryptor via a multiplexer which can, alternatively, connect external inputs, *K*, to the decoder and decryptor if selected by the address input *M*. The circuit is thus able to accept data from remote circuits and provide communication by cryptographic data key. The circuit blocks contained within the dotted box in FIG. 1 are all included in the one-circuit FPGA implementation [3].

3. Model Description

The VHDL description of the electronic-lock system [4] can be divided into three entities. These are entities GEN, MULTIPL and DESIFRAT (see part 7: Source code

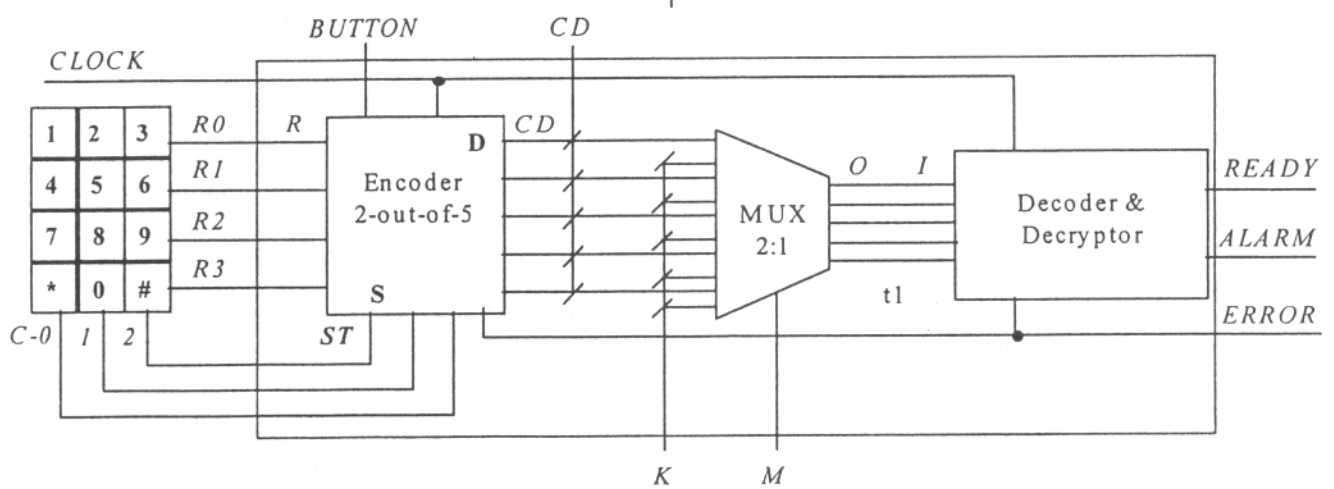

FIG. 1.: The block diagram of the electronic-lock system

of VHDL model). Entity GEN contains a description of the (2-out-of-5) encoder. The keypad is controlled by the signal vector *C*(0 *to* 2), which sequentially tests the individual columns. A shift register behavioural model is used for this purpose.

When a key is operated, a signal vector *R*(0 *to* 3) is generated. If the vector signal *R* is equal to zero then no key is pressed. The output of the (2-out-of-5) encoder on the *CD* bus is defined by the values of signal vectors *C* and *R*.

For subsequent processing it is necessary that the output of entity GEN is only impulses, and this is achieved as follows: The code of the active key is presented as signal *D* and, after de-activation of the key (C=111, *R*=0000), the value of output signal *CD* is determined by the rising edge of the next CLOCK signal. The signal *BUTTON* is active if any key is pressed.

The entity MULTIPL gives the VHDL model of the multiplexer, which allows the connection of another, remote, circuit and keypad. The multiplexer model uses inputs *V* in the description, which are the output bus *CD* of the entity GEN and the external inputs *K*.

The choice of inputs is controlled by the address signal *M*. The chosen inputs are switched to the multiplexer outputs. The outputs of the multiplexer, *O*, are connected to the inputs, *I*, of entity DESIFRAT.

This entity decodes the (2 of 5)-code and it is responsible for the electronic key sequence recognition. The decoding of the non-linear error-control code occurs at the beginning of the program routine describing the behavioural model.

4. Model Simulation

If the five-bit code word contains an error, output signal *ERROR* is activated. If the input word is a valid code word the value is passed to the decryptor. If the input word is not a valid code word, the model DESIFRAT waits for the next input word.

During the simulation the values of the keypad are inserted into signal *TLACZ*. To deactivate the key, the value 55 is inserted. The unlocking combination is pre-set in the simulated example to the sequence of values 1, 2, 3, 4, 5, 6 and 7. If the inserted sequence is not the same as the example sequence the signal *ALARM* is activated as part of the entity ZAMEK in the circuit model.

The finite state machine can be restarted from the *ALARM* active state by activation of button, or code, "*" on the keypad. This causes the binary "11111" to appear on the *CD* bus. When the "*" key is deactivated, the circuit re-initialises and a new sequence can be inserted.

The unlock sequence can be changed by altering of the values of the variables in the terms of test states *S*0 to *S*13 in entity DESIFRAT. These are values expressed in (2-out-of-5)-code. (See: part 6. Table of "keys to (2-out-of-5)-code relation").

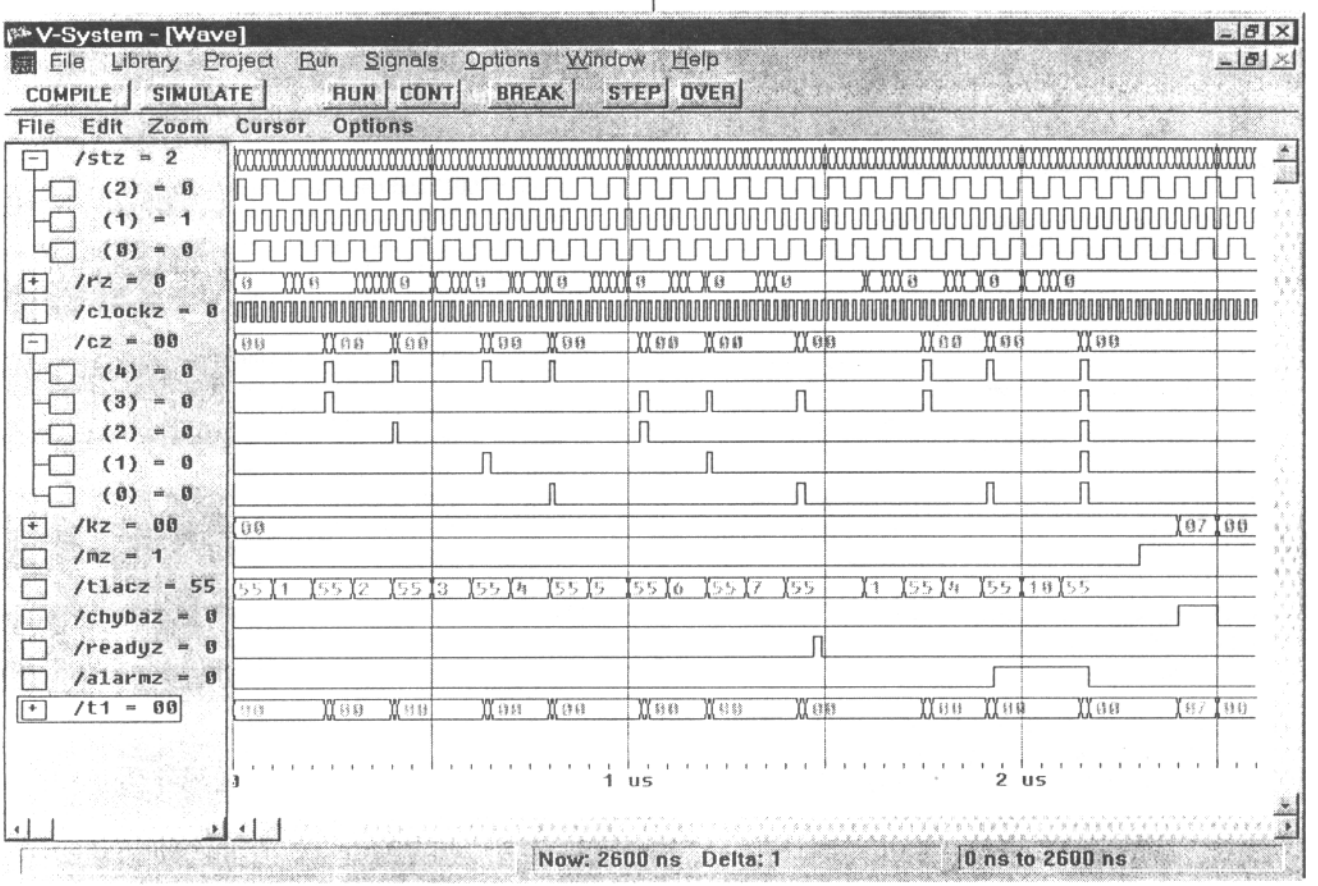

FIG. 2: Wave simulation diagram of electronic-lock circuit

5. Circuit Signals

PIN(S)	Function
C0 - C2	Column keypad outputs
CD0 - CD4	(2-out-of-5)-code outputs
BUTTON	Keypad activity output
READY	Unlock output
ALARM	Wrong unlock sequence output
R0 - R3	Row keypad inputs
K0 -K4	Remote code inputs
M	Multiplexer address
CLOCK	Circuit clock input
ERROR	Error in (2-out-of-5)-code output

6. Keypad and Related Code

Input of the electronic key is by means of a twelve contacts telephone number pad. When a key is activated, one of four "row" contacts is connected to one of three "column" contacts. The rows and columns are connected to a 2-out-of-5 encoder.

Key	Code	Key	Code
1	11000	7	01001
2	10100	8	00110
3	10010	9	00101
4	10001	0	00011
5	01100	"*"	11111
6	01010	"#"	00000

7. Source Code of VHDL Model

```
      -- Sedmimistny kodovy zamek, Jiri Noga
      -- odemykaci kombinace...1234567
entity zamek is
    port (stz: buffer bit_vector (2 downto 0);     -- vstup do klav.
      rz: buffer bit_vector (3 downto 0);          -- vystup z klav.
      clockz: in bit;                              -- hodiny
      cz: buffer bit_vector(4 downto 0);           -- -- vystup za
                                  -- koderem 2z5
      Kz: in bit_vector ( 4 downto 0 );            -- druhy vstup
do multiplexoru
      Mz: in bit;                          -- ovladani multiplexoru
      tlacz : integer;                     -- simulace stalac. tlacitka
                                           -- (55)pust,
                                           --  (10)vyp.alarmu
      CHYBAz, READYz, ALARMz: out bit);
     -- chyba kodu 2z5, odemknuti, alarm
     end zamek;

 entity gen is       -- klavesnice a koder 2z5
    port (st: buffer bit_vector (2 downto 0);
      c: buffer bit_vector(4 downto 0);   -- vystup z koderu
      clock: in bit;
   r: buffer bit_vector (3 downto 0);
   tlac : integer);
 end gen;
 architecture vysilac of gen is
   signal d : bit_vector ( 4 downto 0 );
   signal s : bit_vector ( 2 downto 0 );
   begin
 Sequence: process (clock, r)                -- posouvani s a st
     begin
       if (clock'event and clock = '1')then
          case s is
          when "100" => st <= "010"; s <= "010" after 1 ns;
          when "010" => st <= "001"; s <= "001" after 1 ns;
          when "001" => st <= "111"; s <= "111" after 1 ns;
          when "111" => st <= "100"; s <= "100" after 1 ns;
          when others => st <= "100"; s <= "100" after 1 ns;
          end case;
        end if;
     end process;
 Combinatorial : process ( r,s )
     begin
       if (r = "0001" and st ="001") then d <= "11000";      -- 1
       elsif (r = "0001" and st ="010") then d <= "10100";   -- 2
       elsif (r = "0001" and st ="100") then d <= "10010";   -- 3
       elsif (r = "0010" and st ="001") then d <= "10001";   -- 4
       elsif (r = "0010" and st ="010") then d <= "01100";   -- 5
       elsif (r = "0010" and st ="100") then d <= "01010";   -- 6
       elsif (r = "0100" and st ="001") then d <= "01001";   -- 7
       elsif (r = "0100" and st ="010") then d <= "00110";   -- 8
       elsif (r = "0100" and st ="100") then d <= "00101";   -- 9
       elsif (r = "1000" and st ="001") then d <= "11111";   -- *
       elsif (r = "1000" and st ="010") then d <= "00011";   -- 0
       elsif (r = "1000" and st ="100") then d <= "00000";   -- #
       end if;
       if (r = "0000" and s="111" and st="111") then
    -- nestlacene tlacitko
          c<=d;                          -- vznikne impuls na c
          d<="00000";
        else c<="00000";
        end if;
    end process;

 kalvesnice:  process (tlac, st)
     begin
       if (st = "001" and tlac = 1 ) then r <= "0001";
       elsif (st = "010" and tlac = 2 ) then r <= "0001";
       elsif (st = "100" and tlac = 3 ) then r <= "0001";
       elsif (st = "001" and tlac = 4 ) then r <= "0010";
       elsif (st = "010" and tlac = 5 ) then r <= "0010";
       elsif (st = "100" and tlac = 6 ) then r <= "0010";
       elsif (st = "001" and tlac = 7 ) then r <= "0100";
       elsif (st = "010" and tlac = 8 ) then r <= "0100";
       elsif (st = "100" and tlac = 9 ) then r <= "0100";
       elsif (st = "001" and tlac = 10 ) then r <= "1000";
       elsif (st = "010" and tlac = 0 ) then r <= "1000";
       elsif (st = "100" and tlac = 11 ) then r <= "1000";
       elsif (st = "111" and not(tlac = 55)) then r <="1111";
         else r<="0000";
       end if;
   end process;
end vysilac;
-------------------------------------------------------------------------
entity MULTIPL is                      -- mux mezi
  port ( V,K : in bit_vector ( 4 downto 0 );      --koderem 2z5
                                                  -- a desifratorem
      M : in bit;
      O : out bit_vector ( 4 downto 0 ));
end MULTIPL;
-------------------------------------------------------------------------
architecture MUX of MULTIPL is
      begin
        process (V, K, M)
          begin
```

```
            if (M = '0') then
               O <= V after 1 ns;
            else O <= K after 1 ns;
            end if;
        end process;
end MUX;

entity DESIFRAT is              -- desifrator odemykaciho kodu
   port ( CLOCK: in bit;        -- s kontrolou spravnosti
       I : in bit_vector ( 4 downto 0 );   -- kodu 2z5
       CHYBA, READY, ALARM : out bit);
end DESIFRAT;
      ------------------------------------------------------------------------
      -- Kontrola kodu 2z5 ... cisla klavesnice a jim odpovidajici
      -- 1 11000   7 01001      kombinace v kodu 2z5
      -- 2 10100   8 00110
      -- 3 10010   9 00101     (55)..pusteni tlacitka
      -- 4 10001   0 00011
      -- 5 01100 10* 11111
      -- 6 01010 11# 00000
      ------------------------------------------------------------------------
architecture DESIF of DESIFRAT is
    type StateType is
(S0,S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14,S15,S16)
;
    signal State, NextState : StateType;
      begin
Sequence: process (CLOCK)
          begin
            if (CLOCK'event and CLOCK = '1') then
                  State <= NextState;
             end if;
          end process;
Combinatorial : process (I, State)
          begin
            READY <= '0';
            ALARM <= '0';
             if ( I = "11000"              -- konrola kodu 2z5
               or I = "10100"--  jestli je kod 2z5 spatny
               or I = "10010"  --  zadava se cislo znovu
               or I = "10001"
               or I = "01100"
               or I = "01010"  -- jestli je spatna odemykaci
               or I = "01001"  --  kombinace, aktivuje se alarm
               or I = "00110"  --  ktery se da zrusit zmacknutim
               or I = "00101"  --  10*..11111
               or I = "00011"
               or I = "11111"
               or I = "00000" ) then
                 CHYBA <= '0';
    case State is
      when S0 =>
          if (I = "11000") then                    -- 1cislo
              NextState <= S1;
          elsif (I /= "11000" and I /= "00000") then
              NextState <= S15;
          end if;
      when S1 =>
          if (I = "00000") then NextState <= S2 ; end if;
      when S2 =>                                   -- 2cislo
          if (I = "10100") then
              NextState <= S3;
          elsif (I /= "10100" and I/= "00000" )then
              NextState <= S15;
          end if;
      when S3 =>
          if (I = "00000") then NextState <= S4 ; end if;
      when S4 =>                                   -- 3cislo
          if (I = "10010") then
              NextState <= S5;
          elsif (I /= "10010" and I/= "00000" ) then
              NextState <= S15;
           end if;
      when S5 =>
          if (I = "00000") then NextState <= S6 ; end if;
      when S6 =>                                   -- 4cislo
          if (I = "10001") then
              NextState <= S7;
          elsif (I /= "10001" and I/= "00000" ) then
              NextState <= S15;
          end if;
      when S7 =>
          if (I = "00000") then NextState <= S8 ; end if;
      when S8 =>                                   -- 5cislo
          if (I = "01100") then
              NextState <= S9;
          elsif (I /= "01100" and I/= "00000" ) then
              NextState <= S15;
          end if;
      when S9 =>
            if (I = "00000") then NextState <= S10 ; end if;
      when S10 =>                                  -- 6cislo
          if (I = "01010") then
              NextState <= S11;
          elsif (I /= "01010" and I/= "00000" ) then
              NextState <= S15;
          end if;
      when S11 =>
          if (I = "00000") then NextState <= S12 ; end if;
      when S12 =>                                  -- 7cislo
          if (I = "01001") then
              NextState <= S13;
          elsif (I /= "01001" and I/= "00000" ) then
              NextState <= S15;
          end if;
      when S13 =>
          if (I = "00000") then NextState <= S14 ;end if;
      when S14 =>
          READY <= '1';                            -- odemknuti
              ALARM <= '0';
              NextState <= S0 ;
      when S15 =>                                  -- alarm
              ALARM <= '1';
          if (I = "11111") then
              NextState <= S16;
          end if;
      when S16 =>
          if (I = "00000") then NextState <= S0 ;end if;
          end case;
      else CHYBA <= '1';
          end if;
     end process;
end DESIF;

architecture structural of zamek is              -- vnitrni signaly
      signal t1 : bit_vector ( 4 downto 0 );
      -- pomocny signal mezi multipl. a desifratorem
      -- deklarace lokalnich komponentu
      component gen
          port (st: buffer bit_vector (2 downto 0);
                    c: buffer bit_vector(4 downto 0);
                    clock: in bit;
                    r: buffer bit_vector (3 downto 0);
                    tlac : integer);
```

```
    end component;
    component MULTIPL
        port ( V,K : in bit_vector ( 4 downto 0 );
                M : in bit;
                O : out bit_vector ( 4 downto 0 ));
    end component;
    component DESIFRAT
        port ( CLOCK: in bit;
                I : in bit_vector ( 4 downto 0 );
                CHYBA, READY, ALARM : out bit);
    end component;

-- popis struktury
    begin
u0: gen port map (st => stz, c => cz, clock => clockz, r => rz, tlac
    => tlacz);
u1: MULTIPL port map (V => cz, K => Kz, M => Mz, O => t1);
u3: DESIFRAT port map (CLOCK => clockz, I => t1, CHYBA
    => CHYBAz, READY => READYz, ALARM =>
    ALARMz);
end structural;
```

8. Conclusion

This example of an electronic-lock circuit is a design of average complexity, where it is necessary to be able to modify various aspects in response to customer requirements. The model was verified by functional simulation using the software suite V-System. It is possible to simulate many other details of the time response [6].

When implementing the circuit in an ANTI-FUSE type of FPGA, it is an advantage that the configuration file in memory is not necessary. In this case it is already verified that the circuit will be fully functional.

The more complicated models were simulated using the V-SYSTEM from Model Technology or EASY-VHDL of GALILEO system, from Mentor Graphics, running on HP workstations [7]. When a hazard occurs between two signals, it is possible to confirm the hardware operation by means of two independent probes which are connected directly to the circuit. This allows verification of actual electrical signals by direct connection to test points brought out through free pins on the circuit package.

Acknowledgement

The first author gratefully acknowledges discussions about this work with Prof. Brian R. Bannister. These discussions allow more exact expression of some formulations. The author thanks his student Jiri Noga for careful derivation of VHDL model and the simulation of this and his colleague David Miklik for his efforts in FPGA implementation using the Logic Explorer and the Time Explorer software suite GALILEO system. Last but not least the author thanks his colleagues Ian Bell and Ernst Bartsch for re-simulations of this model in the Cadence VHDL system.

Support for TEMPUS project "EQUATOR" (No. S_JEP/09468-95) is gratefully acknowledged.

References

[1] Scott, T.: "Adopting VHDL for PLD design and simulation", EDN, 1998, Vol. **43**, No. 8, p. 139.

[2] Vlček, K.: "VHDL makes CPLD & FPGA design easy". ST1/98,

[3] Vlček, K.: "Circuit design by VHDL support". Sdel. tech. 2/98,

[4] Vlček, K.: "The VHDL mixed models". Sdel. tech. 4/98,

[5] Vlček, K.: "The VHDL models of finite state machines". Sdel. tech. 6/98

[6] Vlček, K., Noga, J., Mitrych, J.: "The VHDL University Courses". Proc. of Conf. Radioelektronika, (April 28-29, 1998, Czech Republic),

[7] Vlček, K., Miklík, D.: "The VHDL Model of DEC-TED Memory Checker". Proc. of Int. Conf. EWDC-9, (May 14-16, 1998, Poland), ISBN 83-907591-1-X, pp. 54-58.

[8] Vlček, K.: The VHDL Model of Wyner-Ash Cannel Coding for Medical Applications. Proc. of International Workshop DDECS '98 (Sept. 2-4, 1998, Szczyrk, Poland)

About authors...

Karel Vlček has been Associate Professor of Czech Technical University in Prague for Telecommunication. At the time being, he is at the Department of Control and Measurement, Technical University of Ostrava. His main field of interests is investigated in Discrete Signal Processing, and Error Control Coding, Diagnostics and Reliability, Automation of Circuit Design by VHDL, and Biomedical Engineering.

Brian R. Bannister has recently retired as Director of the Microelectronics Lab at the University of Hull, England. He has been with the Department of Electronic Engineering, and more recently the School of Engineering, at the University since 1981. He was previously at Staffordshire University and The Marconi Company Research Labs, Great Baddow, where he was involved with radar data handling and computer systems research.

David Miklík was born in Zlín, (Czech Republic) in 1974. He received the Ing. (MSc.) degree at the Technical University of Ostrava, Faculty of Electrical Engineering and Informatics, Department of Control and Measurement (1991-1996). At the time being, he is a post-graduate student at the Department of Control and Measurement and a technical person in workstation CAD laboratory. His professional orientation interests in Multi-agent Systems, Distributed Control Systems, WWW Programming, and VHDL Circuit Modelling.

Ian M. Bell (see Radioengineering **vol. 8**, No. 4).

Ernst Bartsch (see Radioengineering **vol. 8**, No. 4).

Jan Noga was born in Karviná (Czech Republic) in 1976. He received the Ing. (MSc.) degree at the Technical University of Ostrava, Faculty of Electrical Engineering and Informatics, Department of Telecommunication and Electronics (1994-1999).