# VHDL MODELS WITH USAGE OF THE LFSR_PCKG PACKAGE

Jan KOVALSKÝ[1], Karel VLČEK[1], Jiří MITRYCH[2]

[1] Dept. of Measurement and Control
Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava – Poruba
Czech Republic

[2] PHOBOS s.r.o.
Horní 199, 744 01 Frenštát pod Radhoštěm
Czech Republic

## Abstract

*LFSRs (Linear Feedback Shift Registers) are very often used in the BIST (Built-In Self-Test) methodology. Implementation of the LFSRs to the design or application of digital system, which supports BIST techniques or which only uses these LFSRs, can be done by VHDL language. This paper presents VHDL models of the devices and subroutines (e.g. test pattern generators, signature analysers). Models are based on LFSR structures with usage of the LFSR_PCKG package described in the (Kovalský and Vlček, 2001), which can be used in the applications supporting BIST techniques. Devices are described as behavioural and structural models. These models and descriptions can be used e.g. in the (Kovalský, 2001). The LFSR_PCKG was modified and new approach is presented. Naturally, there are presented some synthesis conclusions of the VHDL models and applications in this paper.*

## Keywords

VHDL Architecture, VHDL Package, LFSR Circuit models (SISR, MISR, SRPG and PRPG), BIST

## 1. Introduction

In *Designs for Testability* (DFT) particularly in *Built-In Self-Test* (BIST) techniques are extensively used *linear feedback shift registers* (LFSRs), as well as their modified versions. LFSRs can be used in many ways. One application is as a source of pseudorandom binary test sequences – *test pattern generator* (TPG). The other is as a means to carry out response compression – *signature analysis* done by *signature analyser* (SA).

The designers very often use VHDL language for theirs digital system designs. There was presented one approach of the LFSR descriptions in VHDL language in (Kovalský and Vlček, 2001). This approach is generally based on the text string definition of the characteristic polynomial. This polynomial and the other necessary parameters are decoded and elaborated in the VHDL package LFSR_PCKG that was described too. However, the package was modified to make its usage more efficient and effective. This package and its subroutines – functions, will be briefly described in the third chapter.

The other chapters will deal with modelling of the designs based on the LFSR structures with usage of the LFSR_PCKG package and possible approaches in the designing. The last two chapters will deal with some synthesis results.

## 2. The LFSR Theory

The main goal why the LFSR_PCKG package was created was to prepare effective subroutines and functions for writing VHDL models with minimum effort and maximum clarity. Two steps solved these requirements. In the first step, the typical and necessary parameters that have to be set (defined) by the user of this package were defined. In the second step, some subroutines and types were coded (see chapter 3).

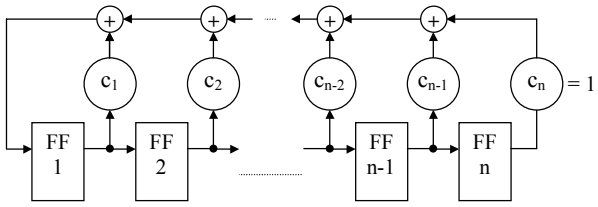


**Fig. 1** Type-1 of the LFSR with external XOR gates

The LFSR is a logic network constructed from the following basic components: D flip-flops, modulo-2 adders (XOR) and modulo-2 scalar multipliers. The theory of such LFSR and its characteristic polynomial is based on the description of the Galois fields (GF) and all operations of LFSR are done over GF(2) finite field. From this follows why we use modulo-2 multipliers and adders. We can see the type-1 LFSR with external XOR gates in the Fig. 1 (other types of LFSRs can be found e.g. in (Abramovici, *et al.*, 1995)). In this figure $c_i$ are binary constants for modulo-2 scalar multipliers. If $c_i = 1$ then connection exists, while $c_i = 0$ implies that no connection exists. Thus, we can discuss its typical parameters and parameters that are needed to be defined by user in this approach. The LFSR is defined by characteristic polynomial (1) – the first general parameter. It is derived from generation function $G(x)$ and this derivation is described in details in (Abramovici, *et al.*,

1995). The powers `i` of `x` define taps counted in the feedback. It means that $c_i$ is equal to one.

$$P(x) = 1 + c_1 x + c_2 x^2 + \ldots + c_n x^n \quad (1)$$

This characteristic polynomial can be in the normal or reciprocal form (the second general parameter) and relation between these two forms is defined by (2). These two parameters are naturally included in the LFSR_PCKG package.

$$P^*(x) = x^n \cdot P(1/x) \quad (2)$$

From equations follows that the LFSR goes through a cyclic or periodic sequence of states and that the produced output is also periodic. The maximum length of this period is $2^n$-1, where `n` is the number of stages – width of the LFSR, and it's called a *maximum-length sequence*. Width of the LFSR is the third general parameter (see chapter 3). The characteristic polynomial associated with a maximum-length sequence is called a *primitive polynomial*. The primitive polynomial is an extraordinary case of *irreducible polynomial*. An *irreducible polynomial* is one that cannot be factored, i.e., it's not divisible by any other polynomial other then 1 and itself. The *irreducible polynomial* $p(x)$ of degree $m$ is called *primitive* one, if there is the smallest positive integer $n$ that allows the polynomial to divide evenly into $x^n + 1$ where the number $n = 2^m - 1$. This theory in more details can be found in (Abramovici, *et al.*, 1995) and (Vlček and Musil, 2001). There are lists of primitive polynomials for many degrees of these polynomials presented in these references.

The start position of the periodic sequence can be set by *initial value*. However, the initial value for the structure in Fig. 1 has to be unequal of zero vector – "00....00". In this case, the LFSR will stay infinitely in this state. The default value defined in the LFSR_PCKG is "000....01". Thus, the user in his application can define the initial value (the start value).

# 3. The LFSR_PCKG Package

Presented approach needs that the user in the declaration part of the design entity defines the characteristic parameters of the LFSR. This enables both structural and behavioural descriptions with the one declaration of the constants – characteristic parameters. We can see this declaration in Tab. 1.

The meaning of each of the parameters is evident from comments in the fragment of the VHDL source code in the Table 1 but for clarity, `init_c` constant defines the initial value – start value of the LFSR. The `polynom_txt_c` constant defines, in the string form, characteristic polynomial that can be in the normal or reciprocal form. The type of the characteristic polynomial is defined by `typ_c` constant.

The last parameter is the width of the LFSR defined by `width_c` constant. One can challenge that the width doesn't need to be defined because it's equal to the highest power of the `x` in the (1). However, it's better for more universality to define it. It's very useful to define characteristic polynomial and width of the LFSR separately in some cases, for example in the constructions of the pseudo-exhaustive pattern generators (Novák, 1996; Kovalský, 2000). Such generator is constructed from LFSR and shift register (SR). Thus, the width of the whole generator and of the LFSR is not equal. Nevertheless, this generator can be effectively solved and coded with presented approach and LFSR_PCKG package.

```
entity LFSR_obecne is
  generic ( width_g : natural := 8);
  port( Clk, Rst : in STD_LOGIC;
        X_out    : out STD_LOGIC_VECTOR(1 to
                                  Width_g));
 -- THE LFSR PARAMETERS
 -- width
 constant width_c : NATURAL := width_g;
 -- initial value
 constant init_c : STD_LOGIC_VECTOR :=
                                  "100000000";
 -- type of the ch. pol. (normal/reciproky)
 constant typ_c : typy_polynomu_t := normal;
 -- characteristic polynomial
 constant polynom_txt_c : STRING :=
              "x^8 + x^4 + x^3 + x^2 + 1";
end LFSR_obecne;
```

**Tab. 1** Declaration of the LFSR model with characteristic parameters

The LFSR_PCKG provides one type and two functions that are necessary for descriptions of the models based on the LFSR structure. Declaration of the package can be seen in Tab. 2.

```
package LFSR_PCKG is
   type typy_polynomu_t is (normal, reciproky);
   function init_test_f (...) return
                           STD_LOGIC_VECTOR;
   function LFSR_f (....) return
                           STD_LOGIC_VECTOR;
end LFSR_PCKG;
```

**Tab. 2** Declaration of the LFSR_PCKG package

The main function of this package needed for realisation of the models is a function `LFSR_f` that implements one step of the LFSR structure. Input parameters of this function are characteristic polynomial in the text string form, type of this polynomial and vector that indicates actual value of the register. Actual taps in the feedback are decoded from the characteristic polynomial and with actual value of the register the new value of the feedback is counted that is shifted into LSB position of the register. This function implements the type-1 LFSR (Fig. 1). Then, the new value of the register output is returned back from the function. Moreover, some internal functions of the package are used in this function. Mainly functions for decoding string polynomial and for converting reciprocal form to the normal form.

The `init_f` function checks if the initial value defined by user is the same width as a width of the LFSR or

whole structure. If these widths are not equal, it returns default initial value "000….01". In the other case, it returns defined value. Using of this function isn't necessary but it's useful to include it in the design to prevent some mistakes. The type `typy_polynomu_t` defines values of possible polynomial forms as an enumerated type.

# 4. Modelling of the LFSR Based Designs

As follows from previous chapters, the modelling of the LFSR based designs may be very simple with usage of the prepared LFSR_PCKG package. Let's more concretise

```
-- Behavioural description
architecture PRPG_beh of PRPG is

begin  -- architecture PRPG_beh
 process (Clk, Rst) is
   -- internal register of the LFSR
   variable X_v : STD_LOGIC_VECTOR(1 to
width_c);
   -- initial value
   constant init_c :
              STD_LOGIC_VECTOR(X_v'range) :=
               init_test_f(init_c, width_c);
  begin  -- process
    if Rst = '0' then
      X_out <= init_c;
      X_v :=  init_c;
    elsif Clk'event and Clk = '1' then
      X_v := LFSR_f(X_v,polynom_txt_c,typ_c);
      X_out <= X_v;
    end if;
 end process;
end PRPG_beh;

-- Structural description
architecture PRPG_str of PRPG is
  signal Set_s, Reg_s : STD_LOGIC_VECTOR
                              (1 to width_c);
  signal ZV_s : std_logic;
begin  -- architecture PRPG_str
  X_out <= Reg_s;
  Set_s <= init_test_f(init_c, width_c) when
                                    Rst = '0'
           else (others => '0');
  ZV_s <= LFSR_f(Reg_s,polynom_txt_c,typ_c)(1);

  PRPG_l: for i in Reg_s'range generate
    D_RS_1: if i = 1 generate
      D_L: D_RS port map (
        Clk   => Clk,
        Rst   => Rst,
        Set   => Set_s(i),
        D_in  => ZV_s,
        D_out => Reg_s(i));
    end generate D_RS_1;
    D_RS_k: if i /= 1 generate
      D: D_RS port map (
        Clk   => Clk,
        Rst   => Rst,
        Set   => Set_s(i),
        D_in  => Reg_s(i-1),
        D_out => Reg_s(i));
    end generate D_RS_k;
  end generate PRPG_l;
end  PRPG_str;
```

**Tab. 3** Behavioural and structural descriptions of the LFSR (PRPG)

designs based on the LFSR principles. There were mentioned two groups of the designs in the chapter 1. The TPGs are one group. Basically, they are characterised by two devices in dependence on the type of the output. A *Pseudo-random Pattern Generator* (PRPG) is a multi-output device and *Shift Register Pattern Generator* (SRPG) is a single-output device. Let's take into consideration declaration of the LFSR entity in the Table 1. We can say that declaration of the PRPG can be the same. Thus, these two models are equal and both behavioural and structural descriptions are shown in Tab. 3. Only small changes have to be made in the realisation of the SRPG. The output signal `X_out` has to be declared as a STD_LOGIC and its actualisation is done by MSB position of internal register.

The SAs group is the second group mentioned in the chapter 1. Therefore, we can discuss two general devices that characterise this group – the *Single Input Shift Register* (SISR) and *Multi Input Shift Register* (MISR) sometimes called IE-MISR (*Internal Exclusive OR type linear feedback path MISR*) (Hlawiczka, 1993). Realisation of the SISR is very similar to realisation of the PRPG (LFSR) and SRPG. The differences are only in evaluation of the new value in the feedback, which is shifted into left position of inter register. These differences in both behavioural and structural descriptions are shown in Tab. 4. Also the additional single input `X_in` has to be declared in the entity declaration part.

Realisation of the MISR is a bit different than we can find e.g. in (Abramovici, *et al.*, 1995) with presented approach. The typical structure was modified to meet utilisation of the LFSR_PCKG and its functions (see Fig. 2). The device was divided to two parts. The first part implements typical LFSR. The second part implements XORing of the multiple input. This division is possible because of the linear function XOR. Such device is called TB-MISR, which means *Top-Bottom XOR type linear feedback path MISR* and it was mentioned e.g. in (Hlawiczka, 1993).

```
-- behavioural description
...
X_v := LFSR_f(X_v,polynom_txt_c,typ_c);
X_v(X_v'left) := X_in xor X_v(X_v'left);
X_out <= X_v;
...

-- structural description
...
ZV_s <= X_in xor

LFSR_f(Reg_s,polynom_txt_c,typ_c)(1);
...
```

**Tab. 4** Differences in the SISR realisation

This modification effectively meets possibilities of the LFSR_PCKG package. Thus, the realisation is very simple. Firstly, it's necessary to declare input vector `X_in` in the declaration part of the entity. Then we can use description of the PRPG (see Tab. 3) and make some changes. The result of this modification for both behavioural and structural descriptions is shown in Tab. 5.

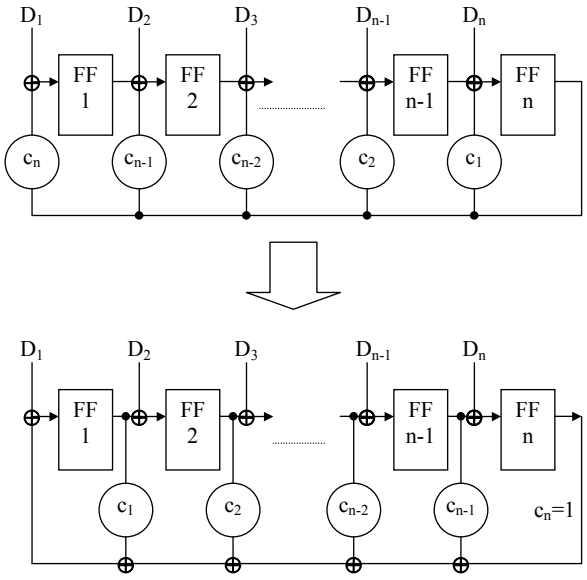


**Fig. 2** Modification of the MISR to the TB-MISR

## 5. The TB-MISR Example

In this chapter will be shown example of the TB-MISR realisation. Because of presented RTL schema possible extensiveness, the simple 4-bit wide TB-MISR was chosen. The primitive polynomial for this width, in this case it means degree of the primitive polynomial, was chosen from (Vlček and Musil, 2001) and it's

$$P(x) = 1 + x + x^4 \tag{3}$$

Thus, now it's necessary to describe MISR in the declaration part of the entity (see Tab. 5).

The TB-MISR descriptions are derived from PRPG descriptions. Realisation of the multiple input XOR with the internal register is done by `for – loop` syntax in the behavioural description. Firstly, the new internal register value is calculated by `LFSR_f` function. Then, the input vector is XORed. The last step is actualisation of the output `X_out`. These steps are going sequentially in the simulation because the syntax is in the process.

There is a similar solution in the structural description. Multiple input XOR is implemented by `for – generate` syntax. The simulation or synthesis tool extracts all `for – generate` loops into the separated instances and equations firstly. Thus, the model works in the simulation correctly and the synthesis results are similar with the results of the behavioural model synthesis.

As an example, the synthesis RTL schema of behavioural description is shown in Fig. 3. The leftmost XOR gate implements feedback of the TB-MISR and through the others XOR gates the multiple input is XORed to the register. As a final technology for this example, FPGA device S10PC84 from Xilinx Spartan family was chosen.

```
entity MISR is
  generic (width_g : natural := 4);
  port(Clk, Rst : in  STD_LOGIC;
       X_in : in STD_LOGIC_VECTOR(1 to
                                    width_g);
       X_out : out STD_LOGIC_VECTOR(1 to
                                   width_g));
  -- MISR definition
  constant width_c : NATURAL := width_g;
  constant init_c : STD_LOGIC_VECTOR :=
                                      "1000";
  constant typ_c : typy_polynomu_t :=
                                       normal;
  constant polynom_txt_c : STRING :=
                                "x^4 + x + 1";
end MISR;

-- Behavioural description
architecture MISR_beh of MISR is
begin  -- architecture MISR_beh
 process (Clk, Rst) is
  variable X_v : STD_LOGIC_VECTOR(1 to
                                    width_c);
  constant init_c :
                STD_LOGIC_VECTOR(X_v'range)
            := init_test_f(init_c, width_c);
 begin  -- process
  if Rst = '0' then
   X_out <= init_c;
   X_v :=  init_c;
  elsif Clk'event and Clk = '1' then
   X_v := LFSR_f(X_v,polynom_txt_c,typ_c);
   for i in X_v'range loop
     X_v(i) := X_in(i) xor X_v(i);
   end loop;  -- i
   X_out <= X_v;
  end if;
 end process;
end MISR_beh;

-- Structural description
architecture MISR_str of MISR is
 signal Set_s, ZV_s, Reg_s :
            STD_LOGIC_VECTOR (1 to width_c);
 begin  -- architecture MISR_str
  X_out <= Reg_s;
  Set_s <= init_test_f(init_c, width_c) when
             Rst = '0' else (others => '0');

  In_XOR: for j in Reg_s'range generate
   xor_l: ZV_s(j) <= X_in(j) xor
       LFSR_f(Reg_s,polynom_txt_c,typ_c)(j);
  end generate In_XOR;

  MISR_l: for i in Reg_s'range generate
      D_L: D_RS port map (
        Clk   => Clk,
        Rst   => Rst,
        Set   => Set_s(i),
        D_in  => ZV_s(i),
        D_out => Reg_s(i));
  end generate MISR_l;
end  MISR_str;
```

**Tab. 5** TB-MISR descriptions

## 6. Synthesis and Its Constraints

During synthesis in (Kovalský and Vlček, 2001) some constraints were found in some synthesis tools. Presented solution was synthesised and solved in the Leonardo Spectrum tool from Exemplar Logic. However, problems were in dc_shell from Synopsys, because it doesn't support any

attributes and VHDL structures used in the LFSR_PCKG package. The problematic attributes `'pos` were replaced in presented solution in this paper. Encapsulation of the LFSR characteristic parameters in record structure was removed – the constants are declared directly in the entity or architecture. Therefore, presented solution was synthesised in four synthesis tools – Leonardo Spectrum (Exemplar Logic), Webpack (Xilinx), Synplify (Actel) and dc_shell (Synopsys). Each of these synthesis tools synthesised presented solution correctly without any errors and constraints. Summary of the unsupported attributes and VHDL structures can be found in (Cohen, 1995).

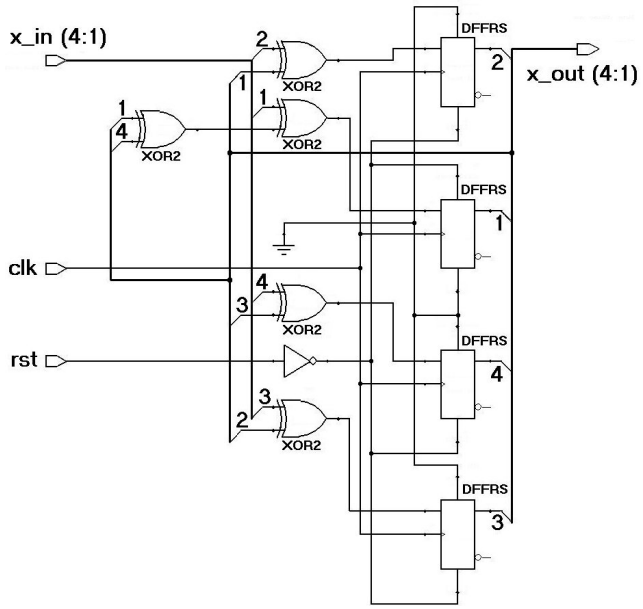


**Fig. 3** RTL schema of the TB-MISR behavioural model synthesis

One can challenge that using of strings in the designs is not very efficient. Therefore, synthesis tool can encode used strings and characters to the words – binary vectors. However, while this is still true, the situation is a little bit different in presented approach. Even the string type is used for definition of characteristic polynomial and even this string is used in other functions, this string doesn't really exist in the final design after synthesis. The string is during analysing and elaborating phase in the synthesis process converted by internal functions of the LFSR_PCKG package to the boolean vector. Thus, there is a boolean vector considered in the final design after synthesis. Moreover, this boolean vector is during optimisation phase of the synthesis process optimised to the logic, which presents linear feedback in the LFSR. This logic is implemented by XOR gates (see Fig. 3). From every active position in the boolean vector is created equation, which presents XOR gate of the linear feedback.

Finally, we can say that the string is not a right constraint in this approach. This is why the solution is synthesisable. The situation with decoding of the string to the boolean vector in the (Kovalský and Vlček, 2001) was the same, but decoding could not be done because of using unsupported attributes and structures in any synthesis tools (see above).

Naturally, all mentioned designs as TPGs and SAs were synthesised and simulated. Simulations were done in ModelSim tool from Model Technology.

# 7. Conclusion

In this paper was presented one approach of the universal LFSR based devices modelling. This approach is characterised by text string definition of the characteristic polynomial and the function that implements LFSR structure defined by this text string polynomial. For this, the LFSR_PCKG package, presented in the (Kovalský and Vlček, 2001), was created let us say modified. This modification solved the general nuisance – recompilation of the LFSR_PCKG that the definition of the characteristic parameters consisted. The another constraint of this package was impossibility of the definition more than one LFSR structure in the design.

Mentioned constraints were solved to create efficient tool – package, which can be reused in many designs and applications. Necessary recompilation was removed, in other words replaced, by `LFSR_f` function. The usage of this function was presented above. As follows from presented VHDL models, only four constants define the LFSR based structure. However, it's possible to define more than one LFSR structure in the design by declaration more than one constant's group. The user can choose naturally only one description (behavioural or structural). In this case, constant's group (groups) can be declared in the architecture instead of declaration in the entity.

As mentioned in the chapter 6, solution presented in the (Kovalský and Vlček, 2001) was not supported by any synthesis tools – namely dc_shell from Synopsys and Webpack from Xilinx. The problem was with `'pos` attribute, which is not supported by these tools. This constraint was solved in this presented solution by another inner functions in the LFSR_PCKG package.

The advantage of this solution is possibility of user-friendly declaration of the LFSR structure by text string polynomial (and others three constants) that is very readable and simply understandable. Still both descriptions (behavioural and structural) are possible and user can choose one, which fits his application and final technology better. Unforgettable advantage of the LFSR_PCKG package usage is possibility of the declaration more than one LFSR structure in the design too.

# 8. Acknowledgements

Support for GACR project "Formal processes in diagnostics of digital circuits – verification of testable design" (No. 102/01/1531) is gratefully acknowledged.

# References

[1] ABRAMOVICI, M., BREUER, M. A., FRIEDMAN, A. D. Digital System Testing and Testable Design. Piscataway: IEEE Press, 1995. ISBN 0-7803-1062-4

[2] COHEN, B. VHDL Coding Styles and Methodologies: an In-Depth Tutorial. Boston/Dordrecht/London: Kluwer Academic Publishers, 1995. ISBN 0-7923-9598-0

[3] HLAWICZKA, A. Latwo testowalne uklady i pakiety cyfrowe. Warszawa (Poland): Wydawnictwa Naukowo-Techniczne, 1993. Pages 105 – 121. ISBN 83-204-1518-7

[4] KOVALSKÝ, J. Vestavěné diagnostické prostředky pro FPGA. (BIST for FPGA). Sdělovací technika. 2000, vol. 48, no. 9, p. 27 – 29. ISSN 0036-9942

[5] KOVALSKÝ, J. VHDL Model of BIST Controller. In Proceedings of the International Conference RADIOELEKTRONIKA 2001. Brno (Czech Republic): Brno University of Technology, 2001, p. 366 – 369. ISBN 80-214-1861-3

[6] KOVALSKÝ, J., VLČEK, K. Efficient VHDL Descriptions of the LFSRs Defined by Text String Polynomial. In Proceedings of IWCIT' 01. Ostrava (Czech Republic): Technical University of Ostrava, 2001, p. 39 – 44. ISBN 80-7078-907-7

[7] NOVÁK, O. Pseudo-exhaustive Test Set Generators for FPGAs. In Proceedings of the International Conference on Programmable Devices and Systems PDS '96. Ostrava (Czech Republic): Technical University of Ostrava, 1996, p. 55 – 61. ISBN 80-85988-12-7

[8] VLČEK, K., MUSIL, V. The n-Bit Generic Model of LFSR. In Proceedings of the Conference EDS '01. Brno: Brno University of Technology, 2001, p. 165 – 176. ISBN 80-214-1960-1

# About authors...

**Jan KOVALSKÝ** was born in Bohumín (Czech Republic) in 1976. He received Ing. (MSc) degree from Technical University of Ostrava - Department of Electronics and Telecommunications - in 1999. Now he is a Ph.D. student at the Department of Measurement and Control of the same university. His interests include digital designs based on VHDL coding, DFT (Design for Testability) particularly BIST and automation insertion of BIST structures during development phase at the VHDL level.

**Karel VLČEK** was born in Zlín in 1948. He received the MSc degree at University of Technology of Brno, PhD., and Assoc. Prof. at the Czech Technical University in Prague. At the time being, he is at the Dept. of Computer Science, Technical University of Ostrava. His main fields of interests are investigated into Digital Signal Processing, Error-Control Coding, Cryptography, Digital Circuits Automation, and Design for Test.

**Jiří MITRYCH** was born in 1957, he graduated on Technical University of Brno in 1981. Since 1981 to 1992 he worked in Tesla Roznov like ASIC designer. At this time he works like designer of ASIC based on FPGA ACTEL in PHOBOS Company. In 2001, he apologised his Ph.D. thesis at the Department of Measurement and Control, FEI, Technical University of Ostrava. His main field of interests behind playing banjo is coding theory, design and diagnostics, and artificial intelligence.