Bologna Master Degree in Electronics Engineering

Design, Test and Reliability of Electronic Systems

2nd Project

Group 2

Diogo Luís | 96922

Duarte Marques | 96523

Prof. Fernando Manuel Duarte Gonçalves

1st semester, P2

January 7th 2024

1 Introduction

The main objective of this project is to add BIST-per-scan to the circuit described in the file `circuito06.v`. This Verilog code describes a finite-state machine (FSM) which includes the following ports:

- **Inputs**: `clock`, `reset`, `request1`, `request2`, `request3`, `request4`
- **Outputs**: `grant_o[3:0]`

Moreover, its current state is defined by a 2-bit variable. However, one of the first steps in this project consists of synthesizing the module with a scan chain and storing the synthesis result in a Verilog file. This procedure indicated that the synthesized circuit includes a scan chain with 26 flip-flops (FFs), which will define the number of periods (N) in each sequence generated by the controller. In this context, and considering that the maximum number of test vectors (M) that could be applied was 1024, a total of 500 test vectors were enough to achieve the desired fault coverage of at least 95%.

2 Block diagram

This project included the development of the Verilog modules in Fig. 1. These were initially tested separately using a specific testbench for each one before being included in the top level along with the synthesized circuit with a scan chain (described in `circuito06_scan_syn.v`). The other main sub-blocks inside the top module include:

- **controller** - described in Project 1, used to generate:
  - an enable signal for the scan chain (`out`);
  - a control signal to define when the linear-feedback shift registers (LFSRs) and the multiple-input signature register (MISR) should update their values, and used to select which inputs are fed into the main circuit with a MUX (`bist_running`);
  - an output which indicates when a full sequence has been generated (`bist_end`);
  - a signal which marks the start of a full sequence in the controller, used to reset the values in the LFSRs, MISR, comparator, and main circuit (`new_seq`);
- **inputLFSR** - maximum-length Fibonacci LFSR with a primitive polynomial of order $n = 5$ used to generate the 4 inputs of the circuit under test (CUT) during each BIST session, as controlled by the **MUX**;
- **stateLFSR** - maximum-length Fibonacci LFSR with a primitive polynomial of order $n = 26$ used to generate the bits fed into the scan chain during each BIST session;
- **misr** - Fibonacci MISR with 9 FFs, used to generate a 9-bit output dependent on the values of `scan_out` and `grant_o[3:0]`;
- **comparator** - generates an output signal (`pass_nfail`) which goes to 1 at the end of each BIST session if the output of the MISR is the same as a pre-defined **signature**.

The controller included in this work is practically the same as described in Project 1, apart from the number of periods and number of sequences in each complete waveform ($N = 26$ and $M = 500$, respectively). Additionally, the signal `new_seq` now lasts for a complete clock period, while in Project 1 it had been defined to start the complete waveform in the very first rising edge in the clock after the rising edge in `bist_start`.

In the block diagram shown in Fig. 1, the modules with a yellow background were included in their own Verilog files, and then instantiated in the top level. On the other hand, the other elements (MUX, OR gate, signature and comparator) were directly implemented in the top level file (`top_level.v`). Moreover, it is essential no emphasize that the comparator and all the modules with a yellow background are sensitive to rising edges in the clock signal. For simplicity purposes, the connections between the input `clock` and these modules were not shown in the block diagram.

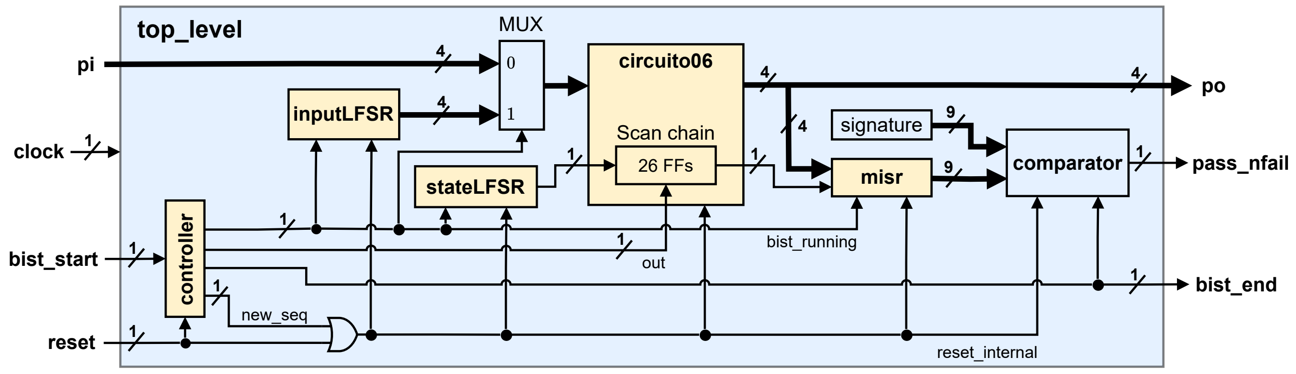

Figure 1: Block diagram of the circuit with BIST-per-scan implemented in the top level, including the internal signals generated by the controller.

3 LFSRs and MISR implementations

The test vectors used as inputs for the CUT during a BIST session are generated by a 5-bit Fibonacci LFSR (**inputLFSR**), serving as a pseudo-random pattern generator. With its feedback mechanism, this maximum-length LFSR systematically produces a sequence of 5-bit patterns, from which 4 bits are applied to the CUT's input nodes. By having the extra bit in the LFSR chain, it allows the use of "0000" as a possible test vector.

The primitive polynomial employed in the input LFSR is $x^5 + x^3 + 1$ (as indicated by Xilinx's Documentation Portal, along with all the other primitive polynomials mentioned in this report) and the test vectors correspond to the 4 least significant bits of the chain, as represented in Fig. 2. To ensure the correct operation of the BIST, the LFSRs must have a well-defined initial value. At the start of every BIST session, the **inputLFSR** is reset with the value "10010". The Verilog code for this module is shown in Listing 1.

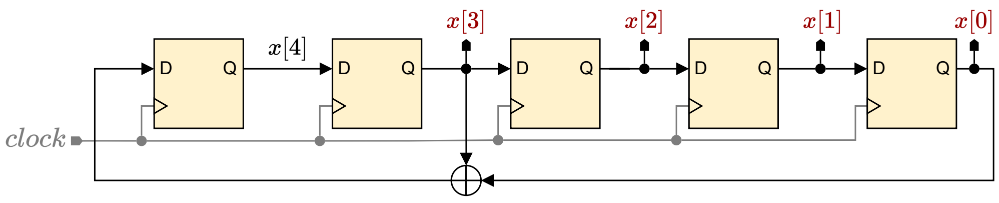

Figure 2: Fibonacci LFSR to generate inputs of the CUT.

Listing 1: Verilog code for the LFSR used to generate the inputs of the circuit.

```verilog
`timescale 1us / 1ps

//////////////////////////////////////////////////////////
//  Authors: Diogo Luis (96922), Duarte Marques (96523)
//  Module Name: inputLFSR
//  Description: Fibonacci LFSR (5 bits) for inputs
//////////////////////////////////////////////////////////

module inputLFSR(clock, reset_internal, control_input, vector);
    input clock, reset_internal, control_input;
    output [3:0] vector;
    localparam [4:0] init_val= 5'b10010;
    reg[4:0] x;

    always @(posedge clock)
    begin
        if (reset_internal == 1'b1)
            x <= init_val;
        else if (control_input == 1'b1)
            x <= {x[3]^x[0], x[4:1]};
    end

    assign vector = x[3:0];
endmodule
```

The **stateLFSR** is a 26-bit Fibonacci LFSR responsible for filling the scan chain during every BIST session. This maximum-length LFSR uses $y^{26} + y^6 + y^2 + y + 1$ as a primitive polynomial, and the second least significant bit, y[1], is fed into the scan chain through the pin `scan_in`, as depicted in Fig. 3. When the reset signal `reset_internal` is at 1, the LFSR assumes the initial value of "10000000001010010001000101". The implementation of this module in Verilog is shown in Listing 2.

In the module **inputLFSR**, the minimum order was taken into account, since 4 bits are needed as inputs, and using only 4 FFs would not generate the sequence "0000". However, different valid implementations could be used for **stateLFSR**. Even though the scan chain is formed by 26 FFs, the maximum number of test vectors imposed for this project does not possibilitate the generation of all 2^{26} combinations anyways. It was nonetheless decided to use an order of 26 for this module since it provided the wanted results.

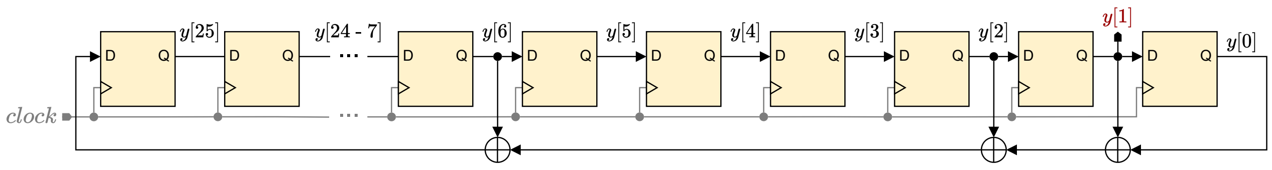

Figure 3: Fibonacci LFSR to generate bits for the scan chain.

Listing 2: Verilog code for the LFSR used to generate the inputs of the scan chain.

```verilog
`timescale 1us / 1ps

///////////////////////////////////////////////////////////
//  Authors: Diogo Luis (96922), Duarte Marques (96523)
//  Module Name: stateLFSR
//  Description: Fibonacci LFSR (26 bits) for scan_in
///////////////////////////////////////////////////////////

module stateLFSR(clock, reset_internal, control_state, y0);
    input clock, reset_internal, control_state;
    output wire y0;
    localparam [25:0] init_val= 26'b10000000001010010001000101;
    reg[25:0] y;

    always @(posedge clock)
    begin
        if (reset_internal == 1'b1)
            y <= init_val;
        else if (control_state == 1'b1)
            y <= {y[6]^y[2]^y[1]^y[0], y[25:1]};
    end

    assign y0 = y[1];
endmodule
```

Finally, the MISR shown in Fig. 4 was implemented in Verilog using the code included in Listing 3. This embedded data sink receives as inputs the outputs of the CUT and scan chain (`grant_o` and `scan_out`, respectively), and generates a signature which is then included in the top level definition. At the end of every BIST session, the value of `pass_nfail` goes to 1 if the output of the MISR is equal to that signature - this output then holds its value until `reset` or `new_seq` change to 1. In order to reduce the probability of aliasing, the considerable number of 9 FFs was used in this module. For this purpose, the primitive polynomial $z^9 + z^5 + 1$ was considered. In this case, an initial value of "0000000001" is given to the 10-bit register `z` shown below.

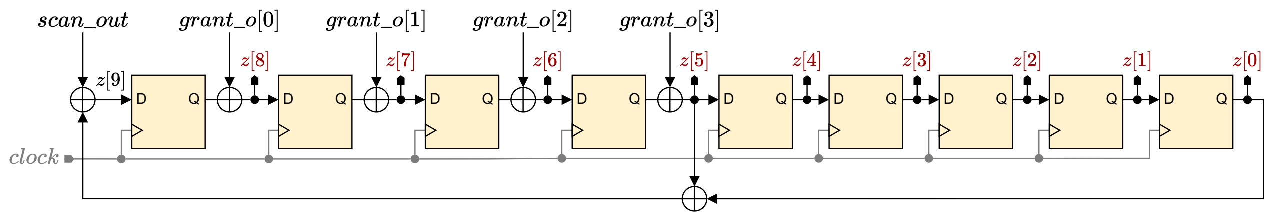

Figure 4: Fibonacci MISR used to generate the input for the comparator.

Listing 3: Verilog code for the MISR.

```
`timescale 1us / 1ps

///////////////////////////////////////////////////////////
//  Authors: Diogo Luis (96922), Duarte Marques (96523)
//  Module Name: misr
//  Description: Fibonacci MISR with 9 FFs
///////////////////////////////////////////////////////////

module misr(clock, reset_internal, scan_out, grant_o, control_misr, output_misr);
    input clock, reset_internal, scan_out, control_misr;
    input [3:0] grant_o;
    output [8:0] output_misr;
    localparam [9:0] init_val = 10'b0000000001;
    reg[9:0] z;

    always @(posedge clock) begin
        if (reset_internal == 1'b1)
            z <= init_val;
        else if (control_misr == 1'b1)
            z <= {scan_out^z[5]^z[0], grant_o[0]^z[9], grant_o[1]^z[8], grant_o[2]^z
    [7], grant_o[3]^z[6], z[5:1]};
    end

    assign output_misr = z[8:0];
endmodule
```

4 Synthesis reports

The original circuit (described in `circuito06.v`) was initially synthesized without a scan chain (and without any BIST modules) in order to evaluate its gate count and area, which is described in the Genus report included in Listing 4. As shown below, a total of 119 gates, corresponding to a total area of 15 524.6 µm^2, were used in this circuit. Interestingly, the analogous report obtained after synthesis with the scan chain manifested no modifications - in general, a scan chain does not directly increase the physical area of the circuit.

Listing 4: Genus report for the original circuit with no scan chain, obtained with 'report gates'.

```
============================================================
  Generated by:           Genus(TM) Synthesis Solution 19.14-s108_1
  Module:                 circuito06
  Technology library:     c35_CORELIB_TYP 3.02
  Operating conditions:   _nominal_ (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

  Gate   Instances    Area          Library
-------------------------------------------------
AOI211          4    291.200    c35_CORELIB_TYP
AOI221         14   1274.000    c35_CORELIB_TYP
AOI311          1     91.000    c35_CORELIB_TYP
CLKIN2          5    182.000    c35_CORELIB_TYP
CLKIN3          3    109.200    c35_CORELIB_TYP
DFS1           26   9464.000    c35_CORELIB_TYP
IMUX21          9    819.000    c35_CORELIB_TYP
INV2            9    327.600    c35_CORELIB_TYP
INV3            4    145.600    c35_CORELIB_TYP
NAND22          8    436.800    c35_CORELIB_TYP
NAND31          4    291.200    c35_CORELIB_TYP
NAND41          1     91.000    c35_CORELIB_TYP
NOR21          17    928.200    c35_CORELIB_TYP
NOR31           4    291.200    c35_CORELIB_TYP
NOR40           1     72.800    c35_CORELIB_TYP
OAI2111         2    182.000    c35_CORELIB_TYP
OAI212          6    436.800    c35_CORELIB_TYP
OAI222          1     91.000    c35_CORELIB_TYP
-------------------------------------------------
total         119  15524.600


     Type       Instances     Area    Area %
------------------------------------------
sequential             26  9464.000    61.0
inverter               21   764.400     4.9
logic                  72  5296.200    34.1
physical_cells          0     0.000     0.0
------------------------------------------
total                 119 15524.600   100.0
```

After synthesis of the top level, the equivalent report shown in Listing 5 was obtained. As expected, the gate count and the total area increased significantly when compared to the original circuit - specifically, to about triple. Once again, the majority of instances correspond to physical cells, but the top level now includes a buffer. Using the Graphical User Interface (GUI) in Genus (with the command 'gui_show'), it was seen that this gate CLKBU2 is used to interconnect wires of the controller module.

Listing 5: Genus report for the top level (including the scan chain and all additional modules to implement the BIST), obtained with 'report gates'.

```
============================================================
  Generated by:           Genus(TM) Synthesis Solution 19.14-s108_1
  Module:                 top_level
  Technology libraries:   c35_CORELIB_TYP 3.02
                          c35_CORELIB_TYP 3.02
  Operating conditions:   _nominal_ (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================


  Gate   Instances     Area          Library
-------------------------------------------------
ADD22            9    1310.400    c35_CORELIB_TYP
AOI211           6     436.800    c35_CORELIB_TYP
AOI221          20    1820.000    c35_CORELIB_TYP
AOI311           1      91.000    c35_CORELIB_TYP
CLKBU2           1      54.600    c35_CORELIB_TYP
CLKIN2           4     145.600    c35_CORELIB_TYP
CLKIN3           5     182.000    c35_CORELIB_TYP
DF3             60   16380.000    c35_CORELIB_TYP
DFE1             1     327.600    c35_CORELIB_TYP
DFS3            26    9937.200    c35_CORELIB_TYP
IMUX20          40    3640.000    c35_CORELIB_TYP
IMUX21           8     728.000    c35_CORELIB_TYP
INV0             5     182.000    c35_CORELIB_TYP
INV2            20     728.000    c35_CORELIB_TYP
INV3            17     618.800    c35_CORELIB_TYP
INV6             1      54.600    c35_CORELIB_TYP
MUX22            1     109.200    c35_CORELIB_TYP
NAND22          28    1528.800    c35_CORELIB_TYP
NAND31           4     291.200    c35_CORELIB_TYP
NAND41           2     182.000    c35_CORELIB_TYP
NOR21           59    3221.400    c35_CORELIB_TYP
NOR31            4     291.200    c35_CORELIB_TYP
NOR40            5     364.000    c35_CORELIB_TYP
OAI2111          4     364.000    c35_CORELIB_TYP
OAI212          12     873.600    c35_CORELIB_TYP
OAI222           1      91.000    c35_CORELIB_TYP
OAI310           1      91.000    c35_CORELIB_TYP
OAI311           1      91.000    c35_CORELIB_TYP
XNR21            1     109.200    c35_CORELIB_TYP
XOR21            7     891.800    c35_CORELIB_TYP
XOR31            1     200.200    c35_CORELIB_TYP
XOR41            1     273.000    c35_CORELIB_TYP
```

```
-------------------------------------------------
total          356   45609.200

     Type      Instances     Area    Area %
------------------------------------------
sequential            87  26644.800    58.4
inverter              52   1911.000     4.2
buffer                 1     54.600     0.1
logic                216  16998.800    37.3
physical_cells         0      0.000     0.0
------------------------------------------
total                356  45609.200   100.0
```

5 Fault simulator reports

The observable nodes for the fault simulator are only the outputs `bist_end` and `pass_nfail`. During this procedure, the tool injects SA0 or SA1 faults in different wires in the synthesized top level. After running the concurrent fault simulation, the report shown in Listing 6 was obtained. Some of these faults are named "Untestable" because they are applied in UNCONNECTED wires, therefore they should not be considered for the fault coverage calculation.

Listing 6: Fault simulator report included in `xfr.log`.

```
Fault Injection Summary:
    Multiplicity: 1
    Total number of faults: 918/809 (total/prime)
    Number of fault runs: 1
    Number of faults injected: 695
        SA0(335), SA1(360), SEU(0), SET(0)
    Expected finish time: 136400us
        ON_TIME(18), DELAYED(0), PREMATURE(0), TIMEOUT(0), STOPPED(677), UNKNOWN(0)

Stuck-At (0/1) Fault Table
                        Total #     Prime #
Untestable              114         114
Detected                759         656
Potentially_detected    25          21
Undetected              20          18
Unobserved_detected     0           0
Unobserved_undetected   0           0
Dangerous_detected      0           0
Dangerous_undetected    0           0
Not_injected            0           0
Total                   918         809
```

Considering the results shown above and applying the formula

$$Fault\ coverage = \frac{\#Detected + \#Potentially\_detected}{\#Detected + \#Potentially\_detected + \#Undetected} \tag{1}$$

the results shown in Tab. 1 were obtained. As desired, a fault coverage above 95% was successfully achieved. For this purpose, it was not necessary to use BIST optimization techniques, such as re-seeding, multiple polynomials or bit-flipping.

Table 1: Fault coverage results.

	Total	**Prime**
# Detected	759	656
# Potentially detected	25	21
# Undetected	20	18
Fault coverage	97.51%	97.41%

Finally, the list of undetected faults recorded in the fault report is depicted in Listing 7. Due to the restrictions imposed on this project, the faults injected in the primary inputs cannot be detected, therefore, they are present in the list below. Most other undetected faults exist in wires whose purpose is not directly identifiable. As desired, the majority of faults injected in the original circuit were detected - only 2 undetected faults remain in this regard, from a total of 146 (registered in the complete list of injected faults).

Apart from the primary inputs - which are not considered in a BIST session due to the MUX implementation -, the other undetected faults may result from the simplicity of the Single Stuck-at fault model itself. Additionally, since the scan chain in this project has 26 FFs, it is impossible to have every single possible combination of 26 bits being fed into the scan chain during each sequence of the BIST session. This would require, at the bare minimum, $2^{26} = 67108864$ test vectors, which is much larger than the maximum number of test vectors allowed (1024). Moreover, every combination of these 26 bits would have to be paired with each combination of the 4 inputs, making this task even more unachievable.

Different values of the fault coverage can be obtained by changing the initial values in the LFSRs, since different combinations of bits at the inputs of the circuit and bits fed into the scan chain would be considered throughout the BIST session. However, every test made in this context led to fault coverage values very close to 97% - even though not all possible combinations were considered, since that corresponds to $2^4 \times 2^{26}$ possibilities of initial values.

Another alternative would be to increase the number of FFs in the MISR, since the probability of aliasing is given by $\approx 2^r$, where r is its order. Nevertheless, it was because of this reason that a considerable number of FFs (specifically, 9) were used in the MISR for this project - a larger number would also lead to a higher area of the synthesized module.

Listing 7: List of undetected faults reported in `fault_report.txt`

```
## fault_report.txt
## File generated by TOOL:  xfr(64)

{ <list of fault nodes> }
    <fault_seed> <fault_type> <fault injection time> <finish time> <result>
    <id equiv> <test> <seed>

"testbench_bist.top_level.circuito06_n_108" NA   SA1  NA   NA        <NOT_RUN,UNDETECTED>
                                            <41  40>   "NA"   NA
"testbench_bist.top_level.n_242"            0    SA0  0fs  136400us  <ON_TIME,UNDETECTED>
                                            <40  NA>   "NA"   NA
"testbench_bist.top_level.n_239"            0    SA1  0fs  136400us  <ON_TIME,UNDETECTED>
                                            <50  NA>   "NA"   NA
"testbench_bist.top_level.circuito06_n_116" 0    SA1  0fs  136400us  <ON_TIME,UNDETECTED>
                                            <79  NA>   "NA"   NA
"testbench_bist.top_level.n_234"            0    SA0  0fs  136400us  <ON_TIME,UNDETECTED>
                                            <82  NA>   "NA"   NA
"testbench_bist.top_level.n_199"            0    SA1  0fs  136400us  <ON_TIME,UNDETECTED>
                                            <174  NA>  "NA"   NA
"testbench_bist.top_level.n_174"            0    SA1  0fs  136400us  <ON_TIME,UNDETECTED>
                                            <228  NA>  "NA"   NA
"testbench_bist.top_level.n_188"            NA   SA0  NA   NA        <NOT_RUN,UNDETECTED>
                                            <351  350> "NA"   NA
"testbench_bist.top_level.n_183"            0    SA1  0fs  136400us  <ON_TIME,UNDETECTED>
                                            <350  NA>  "NA"   NA
"testbench_bist.top_level.pi[2]"            0    SA0  0fs  136400us  <ON_TIME,UNDETECTED>
                                            <353  NA>  "NA"   NA
"testbench_bist.top_level.pi[2]"            0    SA1  0fs  136400us  <ON_TIME,UNDETECTED>
                                            <354  NA>  "NA"   NA
"testbench_bist.top_level.pi[0]"            0    SA0  0fs  136400us  <ON_TIME,UNDETECTED>
                                            <357  NA>  "NA"   NA
"testbench_bist.top_level.pi[0]"            0    SA1  0fs  136400us  <ON_TIME,UNDETECTED>
                                            <358  NA>  "NA"   NA
"testbench_bist.top_level.pi[3]"            0    SA0  0fs  136400us  <ON_TIME,UNDETECTED>
                                            <359  NA>  "NA"   NA
"testbench_bist.top_level.pi[3]"            0    SA1  0fs  136400us  <ON_TIME,UNDETECTED>
                                            <360  NA>  "NA"   NA
"testbench_bist.top_level.pi[1]"            0    SA0  0fs  136400us  <ON_TIME,UNDETECTED>
                                            <361  NA>  "NA"   NA
"testbench_bist.top_level.pi[1]"            0    SA1  0fs  136400us  <ON_TIME,UNDETECTED>
                                            <362  NA>  "NA"   NA
"testbench_bist.top_level.n_166"            0    SA1  0fs  136400us  <ON_TIME,UNDETECTED>
                                            <370  NA>  "NA"   NA
"testbench_bist.top_level.n_4"              0    SA1  0fs  136400us  <ON_TIME,UNDETECTED>
                                            <500  NA>  "NA"   NA
"testbench_bist.top_level.n_8"              0    SA1  0fs  136400us  <ON_TIME,UNDETECTED>
                                            <534  NA>  "NA"   NA
```

6 Analysis and conclusion

Taking the results into account, the proposed objectives for this project were fully achieved. By adding BIST-per-scan to a given circuit, it was possible to obtain a fault coverage of 97.51%, corresponding to 20 undetected faults out of a total of 804 testable faults. These results were reached by using an LFSR with 5 FFs to generate the inputs of the circuit during the BIST session, for which 4 of the resulting 5 bits are considered. In order to feed the scan chain included in the circuit during synthesis, an LFSR of order 26 was used. This large order was considered because the scan chain was formed by 26 FFs. It could be a valid decision to use a smaller order for this LFSR, since only 1 bit is fed into the scan chain at a time, and not every single possible combination from this implementation is actually considered, since only 500 test vectors were used. Additionally, a lesser number of FFs would also lead to a smaller area of the synthesized top level. However, regarding the inputs, the smallest recommended order was taken into account, since the circuit possesses 4 inputs. By using 5 FFs, all possible combinations are generated - using only 4 FFs, the sequence “0000” would not be achievable.

The Fibonacci LFSRs were used as embedded data sources, while a Fibonacci MISR was implemented as an embedded (parallel) data sink. For this purpose, the input sequence consisted of the circuit outputs and the output from the scan chain, and 9 FFs were used to generate a signature. By using a MISR with a large order, the impact of aliasing is reduced. At the end of each BIST session, the 9-bit output of the MISR is compared with this signature, and the value of `pass_nfail` is determined at the rising edge of the clock. The observable nodes for the fault simulator are the outputs `pass_nfail` and `bist_end`, where the latter indicates the end of a BIST session.

In order to increase the fault coverage even more, different architectures or initial values could be used for the LFSRs and MISR. Additionally, using more observable nodes would also facilitate the fault detection. Using BIST-per-clock (for example), it would be possible to reduce the test time, since BIST-per-scan requires a significant time to shift all bits in the scan chain - however, these other methods present alternative disadvantages. More direct improvements include BIST optimization methods, such as re-seeding, multiple polynomials or bit-flipping.