Reliability of digital systems

Autonomous testing

Lecture notes for course PA192/12

Content

15 Autonomous tests generated in real time

These types of tests allow significant savings hardware of built-in diagnostic tools. The test is generated according to implemented algorithm in real time only during testing.

At the test unit is inserted only algorithm according to which the test is generated. These methods generate test is sometimes referred to as algorithmic.

Generating tests are used mainly in cases where the test unit has a regular structure. It is mainly used for memories, PLA circuits and structured designed sequential circuits.

Real-time tests can be generated by program or by HW. Test samples for RAM are primarily generated by programs. Generators built into the structure of the test circuit are used for other types of circuits.

15.1 Cyclic codes

Cyclic codes are linear codes that have the property that if $\mathbf{V = (v_{n-1}, v_{n-2},..., v_0)}$ is code-vector, is code-vector also vector $\mathbf{V' = (v_{n-2}, v_{n-3},..., v_0, v_{-1})}$.

Most of cyclic codes properties can be illustratively described in the representations of these codes as polynomials. There is unambiguous assignment between code-vectors of the cyclic code **(n, k)** with the coordinates of a finite body $\mathbf{GF_{(q)}}$ and the polynomials with the coefficients of $\mathbf{GF_{(q)}}$ of degree less than **n**. These polynomials are called code polynomials.

If $\mathbf{V = (v_{n-1}, v_{n-2},..., v_0)}$ is code-vector cyclic code **(n, k)**, so it has a corresponding polynomial shape $\mathbf{v(x) = v_{n-1}x^{n-1} \oplus_q ... \oplus_q v_1x^1 \oplus_q v_0x^0}$, where character "$\oplus_q$" indicates add modulo **q**.

15.1.1 Finite body $\mathbf{GF_{(2)}}$

Coordinates (resp. coefficients) $\mathbf{v_i}$ of the binary cyclic code of the infinite body $\mathbf{GF_{(2)}}$ can take the values **0** and **1** only. Their sum is performed by functions **mod 2**. Their product is identical to the conventional algebraic and logical multiplication.

Note:

In the arithmetic of the residual classes, the modulo 2 polynomials are added, subtracted, divided and multiplied as common polynomials, but above the resulting coefficients we perform the modulo 2 operations (the remainder after division by two). For example:

- *−2 modulo 2 is 0,*
- *−1 modulo 2 is 1,*
- *0 modulo 2 is 0,*
- *+ 1 modulo 2 is 1,*
- *+ 2 modulo 2 is 0,*
- *+ 3 modulo 2 is 1,*
- *+ 4 modulo 2 is 0,*
- *atd.*

The method of counting the elements of congruence classes linear algebra of polynomials over $\mathbf{GF_{(2)}}$ is for $\mathbf{Q(x) = x^2 + x + 1}$ and $\mathbf{P(x) = x^3 + x^2 + 1}$ following:

- $\mathbf{P(x) + Q(x) = x^3 + 2x^2 + x + 2 = x^3 + x}$
    - $P(x) + Q(x) = (x^3 + x^2 + 1) + (x^2 + x + 1) = x^3 + 2x^2 + x + 2$
    - $(P(x) + Q(x)) \oplus_2 = x^3 + x$

- **$P(x) - Q(x) = x^3 - x = x^3 + x$**
  - *$P(x) - Q(x) = (x^3 + x^2 + 1) - (x^2 + x + 1) = x^3 - x$*
  - *$(P(x) - Q(x)) \oplus_2 = x^3 + x$*
- **$P(x) * Q(x) = x^5 + x + 1$**
  - *$P(x) * Q(x) = (x^3 + x^2 + 1) * (x^2 + x + 1) = x^5 + 2x^4 + 2x^3 + 2x^2 + x + 1$*
  - *$(P(x) * Q(x)) \oplus_2 = (x^5 + x + 1)$*

    *If **n=5** it is in some applications useful or necessary to perform the operation **mod $x^5 + 1$** and result is **$P(x)*Q(x) = x$***
- **$P(x) / Q(x) = x$**
  - *$P(x) / Q(x) = (x^3 + x^2 + 1) / (x^2 + x + 1) = x - (x-1) / (x^2 + x + 1)$*
  - *$(P(x) / Q(x)) \oplus_2 = x + 1$*

15.1.2 Non decomposable polynomial in the infinite body $GF_{(2)}$

Polynomial **P(x)** of degree **m** is irreducible if it cannot be divided without a remainder by any other lower-order polynomial. Examples:

$x^4 + x^3 + x^2 + x + 1$
$x^4 + x^3 + 1$
$x^4 + x + 1$
$x^2 + x + 1$
$x + 1$

Be careful polynomial **x^2+1** is decomposable. Polynomial **$x^2 + 1$** is divisible by **$x + 1$**, because **$x^2 + 1 = (x+1)(x+1)$.**

15.1.3 Primitive polynomial

Polynomial **P(x)** of degree **m** is primitive if it can be divided completely by **$(1+x^k)$**, where **$k=2^m-1$** and cannot be divided completely by **$(1+x^i)$**, for all **$i<k$**. For every positive **k**, there is at least one primitive polynomial order **k**. Determining of the primitive polynomials of higher orders is not easy. Commonly are used tables.

m	polynomial example	Number of primitive polynomial order m
1	$x+1$	1
2	x^2+x+1	1
3	x^3+x+1	2
4	x^4+x+1	2
5	x^5+x^2+1	6
6	x^5+x+1	6
7	x^7+x^3+1	18
8	$x^8+x^4+x^3+x^2+1$	16
9	x^9+x^4+1	48
..	...	...
16	$x^{16}+x^{12}+x^3+x+1$	2048

Table 15.1: Table of the primitive polynomials

15.2 Signature analysis

The design of contemporary digital systems is primarily based on the application of bus structures linking Large Scale Integration (LSI) circuits - eg microprocessors, ROMs, RAMs, complex interfacing units, programmable structures. Both the bus and the ports of these systems are bi-directional.

Finding and locating malfunctions inside the microprocessor is difficult even if detailed documentation is available. That is why most repairs of microprocessor systems are solved by replacing the entire board.

In another situation are designers of electronic systems. It is necessary to detect whether the unsatisfactory function of the microprocessor system is due to an incorrect design of the technical structure or control program, or to a failure of the implemented circuits. Although they are armed with logic analyzers and other special instrumentation, the finding the cause of the unsatisfactory function of the microprocessor system with the bus structure requires a detailed knowledge of the circuits, the solved algorithm and the programming tool. Finding the occasional unwanted behavior of the system can take a very long time. The main problem is that test devices provide either too much or too little information.

Contemporary diagnostic devices provide many configuration options to monitor system behavior conditions, and it is often the case that the diagnostic expert needs to look for a suitable configuration of the fault finding conditions in the design of the electronic system.

In addition to other techniques, it is possible to compress the internal data to effectively find the malfunctioning part of the electronic system. Long data strings, which are cyclically repeated in an existing system running at normal operating speed, can be compressed into a compact, easy-to-interpret and characteristic expression. This term does not have to say anything other than that the relevant node examined or does not work properly. This principle uses flagship analysis of electronic systems.

15.3 Basis of a Signature Analysis

The basic components of the Signature Analysis (SA) are *data compression* and a *stimulus* generated by the test circuit.

The principle of signature analysis is as follows - each node of the electronic system is understood as a source of the sequential signal. In a certain time interval, can be traced a certain number of logical zero "log.0" and logic one "log.1" at this node. If a repeatable synchronization of this signal can be ensured, it can be easily, repetitively and reliably written to the serial register and then performed further operations over it. The signal entered in the registry can be understood as a code and we need to find a suitable tool to verify its authenticity. This can be verified by comparing the current code with the standard code, which is obtained by measuring on a fault-free electronic system.

The basic prerequisite for the successful application of the signature analysis is the fact that the code generated by an error-free circuit (and represented by the sequence log.0 and log.1) differs from the code affected by circuit failure - from the modified sequence log log.0 and log.1.

The basis of the signature analysis is, as with all diagnostic methods, the skill to force the failure to affect the behavior of the signal at a system node that is accessible to the probe of the measuring instruments.

The elegance of the signature analysis then consists in simply comparing code (signature) in the nodes of a properly functioning electronic system (etalon) with signature in the node of the electronic system with a failure. When using the signature analysis, it is not necessary to analyze the behavior of the circuit and to penetrate to the nature of the unwanted signal sequences. It is clear that signature analysis is a suitable method especially for repairing already designed and previously functioning electronic systems.

Authentication of the code stored in the logbook may be somewhat difficult for large segments of the diagnostic system capacities. To simplify authentication, data compression is used in symptomatic analysis. Compression principles are generally more. The simplest is to detect the number of "log 1" in the symptom – see Figure 15.1.

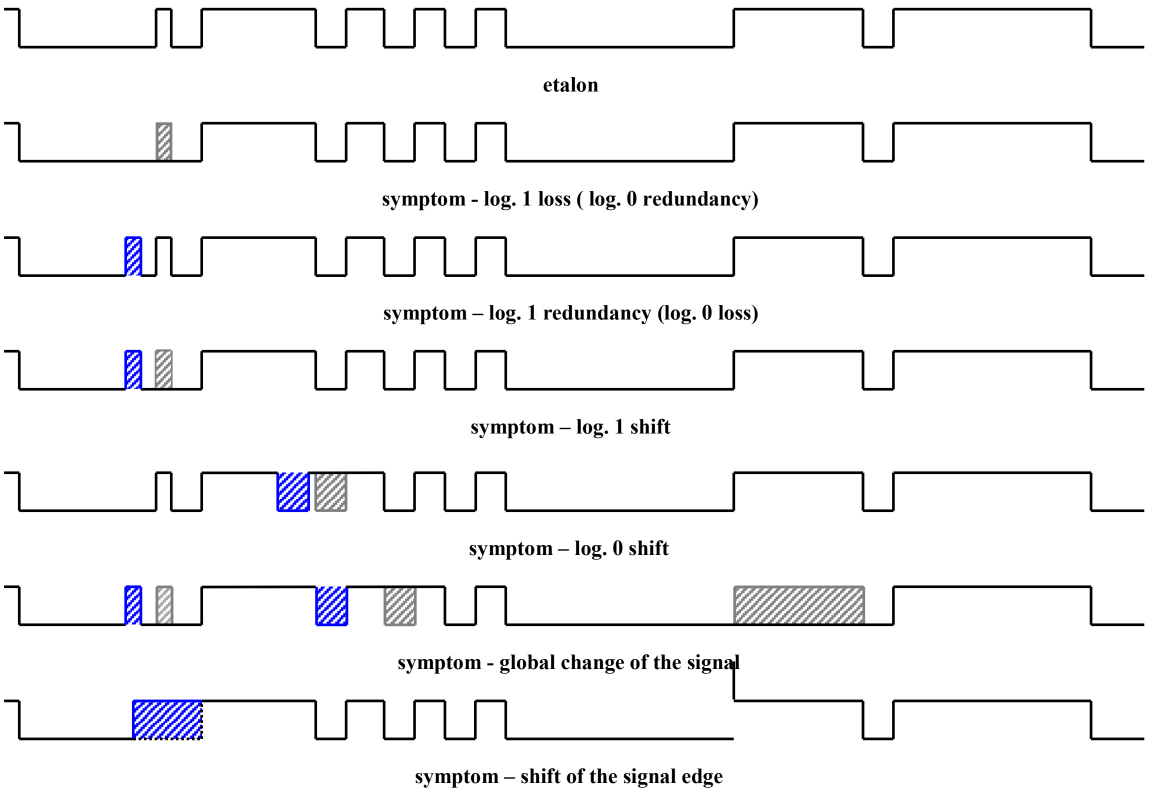

etalon
symptom - log. 1 loss (log. 0 redundancy)
symptom – log. 1 redundancy (log. 0 loss)
symptom – log. 1 shift
symptom – log. 0 shift
symptom - global change of the signal
symptom – shift of the signal edge

Figure 15.1: Examples of manifestations of failure symptoms in the test circuit

It can be seen from the previous figure that simply counting the number of log.1 may not always be successful, because especially for the symptoms of moving log.1 or moving log.0, the sum leads to incorrect authentication. To eliminate this phenomenon, a data compression mechanism is used. The checksum may be entirely satisfactory.

The compression mechanism interprets the serial data in the test node as sequence and these bits change into so-called security word-signature. The signature is generated by the generating polynomial. This is a mechanism commonly used for securing serially transmitted digital information. In common realizations, a code length of 216 bits is assumed. This word length follows from bus widths of the most commonly used single-chip processors that use sixteen-bit buses. The structure of the generating polynomial is determined by the desired attributes of the signature. The basic requirement is that two or more different signals do not have the same signature.

Unique identification of the input code of the signature analysis is provided by the synchronization signal - **CLK** and by the so-called time window. The sync signal defines the valid values of the test signal (signature analyzer input signal) denoted by **DATA**.

The time window defines part of the cyclically repeated operation of the system under test subjected to diagnostic analysis and is sometimes referred to in the signature analysis as the **START-STOP interval**. Most digital systems are designed as synchronous systems and therefore signal changes in the nodes of the circuit under test are synchronized with one of the edges of the synchronization signal.

For the exact time window definition, it is not enough just to select the signals defining its start and end, but it is necessary to define the edges of these signals. The same is true for selecting DATA signal sampling times. You must select the edge of the sync signal that does not cause the measured signals to change. It is desirable to select the edge of the synchronization signal, which expresses the end of the transition states and the stabilization of the measured signals amplitude. In this way, it is possible to clearly define the testing conditions of the digital system for a clear and repeatable sequence of diagnostic operations allowing localization of a failure of the tested digital system.

Provided that these conditions are met, DATA enters into the signature analyzer synchronously with the clock signal of the device under test during the START-STOP interval. Both the synchronization and the time window are defined by the test system.

The signature is generated by the shifting register, complemented by the circuits performing the polynomial function. Small coding distances of DATA signal are interpreted as large code distances by the signature, in other words small changes in the input data will be a major change in the signature.

15.3.1 Linear Feedback Shift Register - LFSR

The Linear Feedback Shift Register – LFSR – allows to perform modulo2 multiplication and division operations. These features have multiple uses for encoding serial interfaces, such as pseudo-random test sequence generator, signature analyzer, and other applications. These operations are easily accomplishing by integrating the nonequivalence circuits into shift register feedbacks. Since the feedback loop is only the non-equivalence circuit - **XOR**, which is a linear logic function, this feedback is referred to as linear feedback. Shift register is easily realized by chain of the D flip-flops.

The LFSRs are used as efficient test sequence generators too. For generating test sequences, counters can be used, but LFSRs are faster, they have a simpler structure, and the test sequence can be easily changed by setting the sliding register's initial state.

15.3.1.1 Implementation of multiplication and division in the finite body GF(2)

Both operations can be implemented in two ways. In the first variant, the **XOR** circuits are serially arranged in a chain of flip-flops. In the second variant, the XOR circuits are connected in parallel to the outputs of the flip-flops - in this variant, the parity generation circuit can be used, simplifying both the design and the realization. These options are listed in the following examples.

Example 1:

Circuit ***multiplication*** *by polynomial* $\boldsymbol{Q(x) = x^4 + x + 1}$.

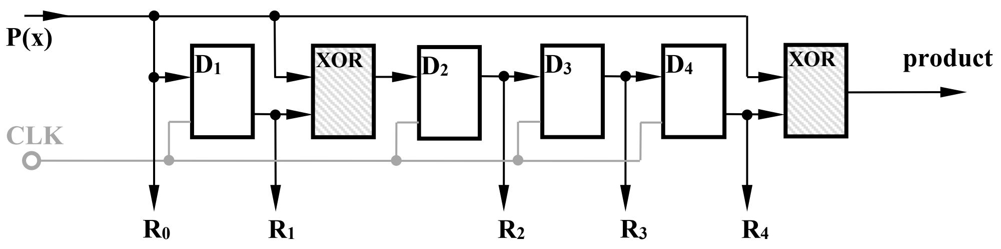


Figure 15.2: Circuit multiplication by polynomial $\boldsymbol{Q(x) = 1 + x + x^4}$ *relized by linear feedback shift register - variant 1*

The shifting register is connected in such a way that the multiplication is represented by a serial sequence of bits fed to the shifting register input, which is completed by the ***XOR*** *circuits connected according to the multiplier shape.*

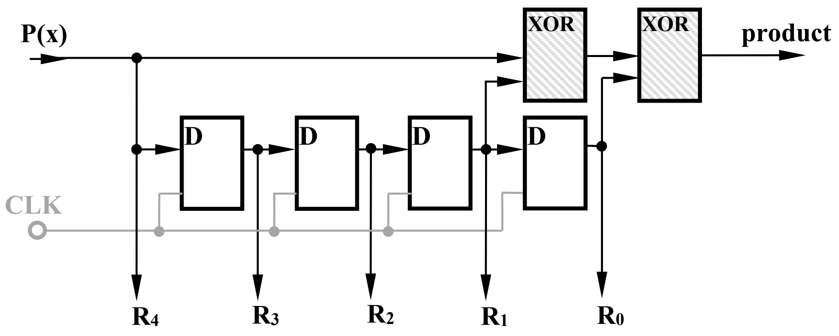
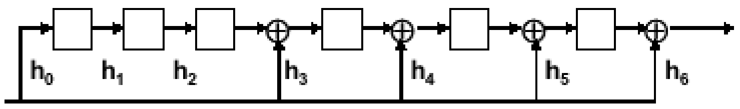


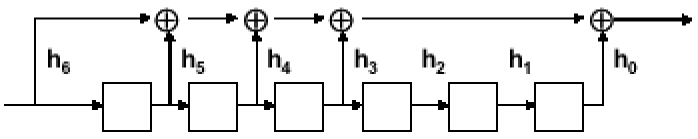
Figure 15.3: Circuit multiplication by polynomial $\boldsymbol{Q(x) = 1 + x + x^4}$ *realized by linear feedback shift register - variant 2*

Example 2:

Polynomial multiplication circuit by $\boldsymbol{1 + x^3 + x^4 + x^5 + x^6}$



First possible realization



Second possible realization

Figure 15.4: hardware implementation of multiplication of polynomials, first and second posible realizations

To realize the division, the divider is fed to the sliding register input as a bit sequence, and the divisor is represented by the XOR circuitry. The partitioning circuit can be connected again in two variants. Both options illustrate the following examples, each of which includes a wiring diagram accompanied by partition demonstrations in a table showing successive changes to the contents of each registry bit and arithmetic calculation.

Example 3:

*Circuit **division** by polynomial **Q(x)** = $x^4 + x + 1$*

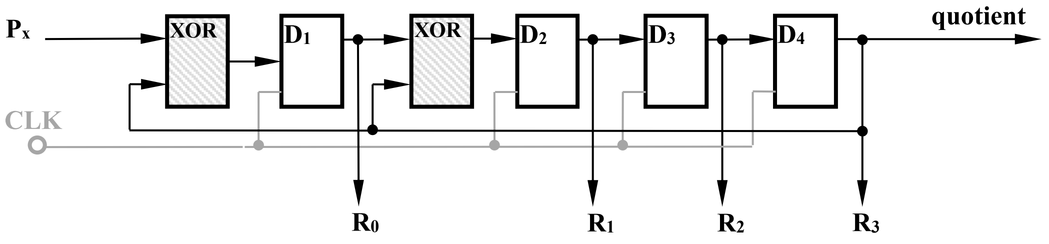


*Figure 15.5: Circuit division by polynomial **Q(x)** = $1 + x + x^4$ linear feedback shift register - variant 1*

Demonstration of polynomial division $x^7 + x^6 + x^5 + x^4 + x^2 + 1$ *by polynomial* $x^4 + x + 1$.

input sequence								outputs of flip-flops					serial output							
P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	R_0	R_1	R_2	R_3	R_4	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7
1	0	1	0	1	1	1	1 0	1	0 0	0	0	0								
	1	0	1	0	1	1	1 0	1	1 0	0	0	0	0							
		1	0	1	0	1	1 0	1	1 0	1	0	0	0	0						
			1	0	1	0	1 0	1	1 0	1	1	0	0	0	0					
				1	0	1	0 1	1	1 1	1	1	1	0	0	0	0				
					1	0	1 1	0	1 1	0	1	1	1	0	0	0	0			
						1	0 1	1	0 1	0	0	1	1	1	0	0	0	0		
							1 0	1	1 0	1	0	0	1	1	1	0	0	0	0	
							0 0	0	1 0	1	1	0	0	1	1	1	0	0	0	0
								R_0	R_1	R_2	R_3	R_4								

Table 15.2: Illustration of hardware division - variant 1

Algebraic expression of division modulo 2:

$$\begin{array}{l}
[(x^7 + x^6 + x^5 + x^4 + 0x^3 + x^2 + 0x + 1) : (x^4 + x + 1)]\oplus_2 = x^3 + x^2 + x \\
\underline{-x^7 \qquad\qquad -x^4 - x^3 \qquad\qquad\qquad} \\
\qquad x^6 + x^5 \qquad +x^3 \quad +x^2 \qquad\quad +1 \\
\underline{\qquad -x^6 \qquad\qquad -x^3 \quad -x^2 \qquad\qquad} \\
\qquad\qquad x^5 \qquad\qquad\qquad\qquad\quad +1 \\
\underline{\qquad\quad -x^5 \qquad\qquad\quad -x^2 \quad -x \qquad} \\
\qquad\qquad\qquad\qquad\qquad\quad x^2 \quad +x \quad +1
\end{array}$$

*Quotient **O(x)** = $x^3 + x^2 + x$ and its binary interpretation **O(x) = 0 1 1 1 0**.*

*The integer rest **R** = $(x^2 + x + 1)$, its binary interpretation = **0 1 1 1**.*

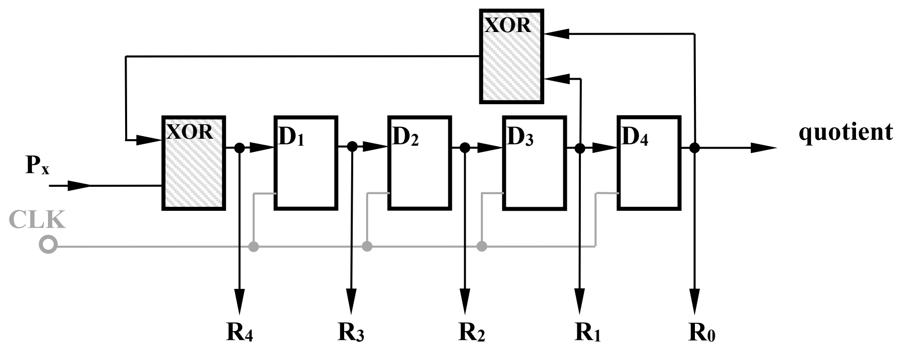


*Figure 15.6: Circuit division by polynomial **Q(x)** = **1** + **x** + **x⁴** linear feedback shift register - variant 2*

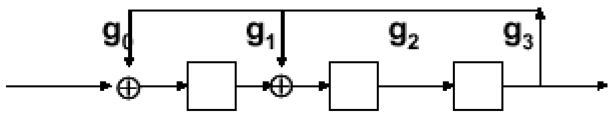
Demonstration of polynomial division $x^7+x^6+x^5+x^4+x^2+1$ *by polynomial* x^4+x+1.

input sequence								outputs of flip-flops					serial output							
P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	R_4	R_3	R_2	R_1	R_0	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7
1	0	1	0	1	1	1	1 00	1	0	0	0	0								
	1	0	1	0	1	1	1 00	1	1	0	0	0	0							
		1	0	1	0	1	1 00	1	1	1	0	0	0	0						
			1	0	1	0	1 10	0	1	1	1	0	0	0	0					
				1	0	1	0 11	0	0	1	1	1	0	0	0	0				
					1	0	1 11	1	0	0	1	1	1	0	0	0	0			
						1	0 01	1	1	0	0	1	1	1	0	0	0	0		
							1 00	1	1	1	0	0	1	1	1	0	0	0	0	
							1 10	0	1	1	1	0	0	1	1	1	0	0	0	0
								R_4	R_3	R_2	R_1	R_0								

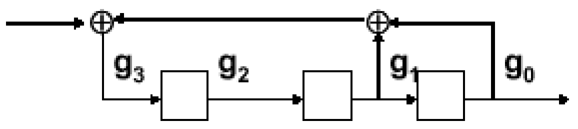
Table 15.3: Illustration of hardware division - variant 2

Example 4:

Polynomial division circuit by $1+x+x^3$



First posible realization



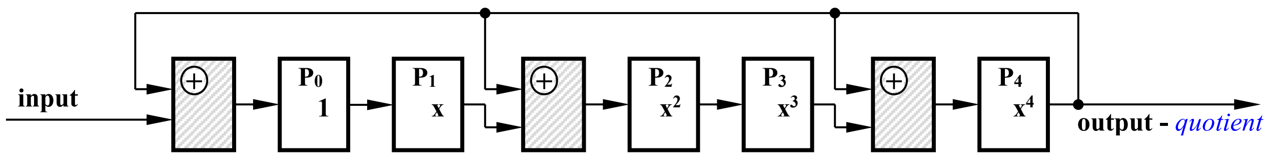
Second possible realization

$x^3+x^5+x^6 \quad : \quad 1+x+x^3$

input	flip flop	output
0001011	0 0 0	0
000101	1 0 0	0
00010	1 1 0	0
0001	0 1 1	0
000	0 1 1	1
00	1 1 1	1
0	1 0 1	1
-	1 0 0	1

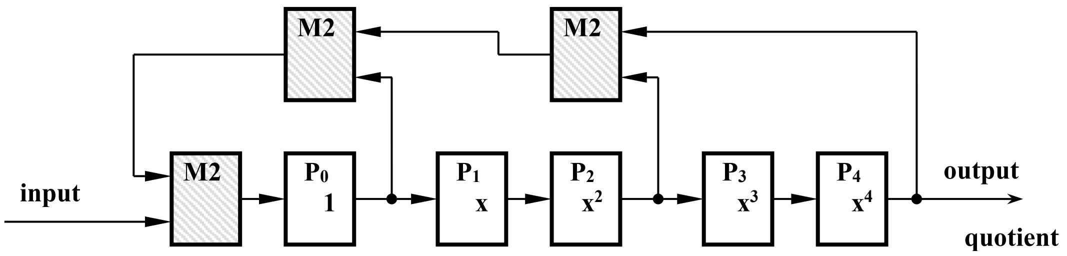
Figure 15.7: hardware implementation of dividing of polynomials

Example 4:



input sequence								register content					output		
V_0	V_1	V_2	V_3	V_4	V_5	V_6	V_7	P_0	P_1	P_2	P_3	P_4	O_0	O_1	O_2
1	0	1	0	1	1	1	1	0	0	0	0	0	0	0	0
	1	0	1	0	1	1	1	1	0	0	0	0	0	0	0
		1	0	1	0	1	1	1	1	0	0	0	0	0	0
			1	0	1	0	1	1	1	1	0	0	0	0	0
				1	0	1	0	1	1	1	1	0	0	0	0
					1	0	1	0	1	1	1	1	0	0	0
						1	0	0	0	0	1	0	1	0	0
							1	0	0	0	0	1	0	1	0
								0	0	1	0	1	1	0	1

Linear Feedback Shift Register – **LFSR** is diagnostics tool based on hardware implementation of dividing of polynomials. Maximum length sequence of states is **2^n-1** (**n** is number of bits). The maximum length can only be achieved if **LSFR** connected by a primitive polynomial.



input sequence								register content					output		
V_0	V_1	V_2	V_3	V_4	V_5	V_6	V_7	P_0	P_1	P_2	P_3	P_4	O_0	O_1	O_2
1	0	1	0	1	1	1	1	0	0	0	0	0	0	0	0
	1	0	1	0	1	1	1	1	0	0	0	0	0	0	0
		1	0	1	0	1	1	0	1	0	0	0	0	0	0
			1	0	1	0	1	1	0	1	0	0	0	0	0
				1	0	1	0	1	1	0	1	0	0	0	0
					1	0	1	0	1	1	0	1	0	0	0
						1	0	1	1	1	1	0	1	0	0
							1	0	1	1	1	1	0	1	0
								1	0	1	1	1	1	0	1

15.3.1.2 Structure of easy-to-tested digital systems

The good testability of digital systems is the result of a considerable effort by the designer.

Testability requires a number of technical adjustments and accessories for technical and software. Testability must be part of the initial design of the device, it can be stated that it must be part of the system design idea, that is, before its specific configuration and the peripheral and program solutions are stabilized. It is only during this period that a fully-fledged testing and repairing strategy can be implemented and only the space for their implementation can be effectively defined (e.g. the ROM space, the location for the switches, the measuring points and the additional technical equipment).

Testability of devices can be increased by rigorous division into functional units. E.g. by dividing the circuits into several mutually dependent modules, each of which will be on its own circuit board, we will get a more readily understandable and repairable device than in case of interconnected circuits on one large printed circuit board. This solution, on the other hand, is not entirely ideal because it introduces additional hardware to the system and reduces system failure intensity.

Microprocessor systems generally allow such divisions of functional units. However, if signature analysis is applied to such systems, large PCBs lose many of their poor testability, and their other exits are again highlighted (excluding transient resistors in connectors, low production costs of large series, etc.).

The basic component of testability is self-control, usually carried out in the form of tests and measurements carried out by built-in diagnostic tools, to verify the functionality of most circuits. The self-monitoring routine can be started automatically when you turn on the machine (POST - Power on Start Test) or by pressing a button on the control panel or other stimulus. The result of a typical self-check is a type indication (good - bad) and can be immediately displayed to the operator or transmitted to the remote service station after the diagnostic bus. Self-monitoring can provide diagnostic information at a certain level, if required.

The main features of self-control are:

- pre-alerts the operator of possible problems in system operation,
- convinces the operator that the system whose correct function was in doubt is in fact okay.

The second feature also protects inexperienced customers from sending a functioning (but very complex) system to unnecessary repairs.

If the device is found to be defective, it must either be repaired or discarded. Correction means "Detecting its malfunction and then locating the fault". If the device has been developed for ease of testability, most defects will be quickly found and corrected. This is the moment when efforts made to design and implement easy testability are effective in the speed and success of repairs.

15.3.2 Generating test sequences

LFSR can be used as the test sequence generator. In this case, the partition is not input to its input, but it is, for example, asynchronously set to the initial value. At the beginning of the activity, the initial, nonzero content must be entered into the **LFSR** because the **LFSR** does not get from the zero state. After running, **LFSR** generates the same sequence of internal states that can be applied as test vectors.

If the feedback is connected, for example, according to the primitive polynomial $\mathbf{x^4 + x + 1}$, the maximum length of the sequence of states can be $\mathbf{2^4 - 1 = 15}$ (zero status missing). This maximum length can only be achieved if the **LFSR** is connected according to the primitive polynomial. We can use this circuit as an internal test generator by fitting it into the test unit and connecting its parallel outputs permanently to the locations we want to stimulate. This test method can be used practically for all types of logic circuits, but is actually effective for combination circuits.

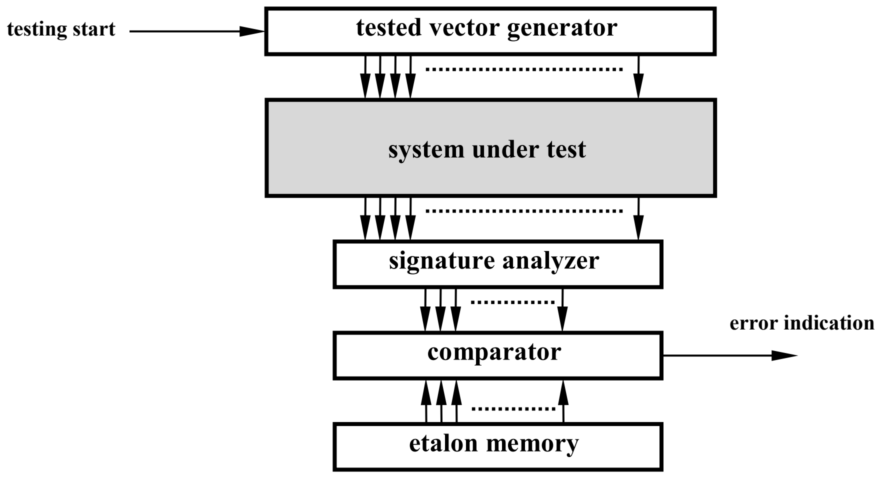


Figure 15.8: Principal scheme of embedding the signature analyzer into the numeric system

The **LFSR** is also used as a flag analyzer too, and is synchronized with common clocks to the test generator. The result of the test is compared to the symptom stored in the memory of the correct symptom-standards. In the event of nonconformity, the comparator signals the error – see Figure 15.8.

Note:

*This method of implementation of the internal test was used, for example, by Intel in its 80386 microprocessor. The length of the **LFSR** used ranges from 12 to 37. The diagnostic coverage achieved by the manufacturer is better than 98%.*

This circuit can be used as a test generator, so that it is incorporated into the unit under test and its parallel outputs connect permanently to the places that want to stimulate. It can be used as BILBO - Built-In Logic Observer.

15.4 Systems BILBO - *Built-In Logic Observer*

It is a design element of VLSI circuits that perform multiple functions. By default, it can work as a parallel register, a serial register, a **LFSR** to generate test vectors, and eventually as a multiple input flag analyzer - **MISA**.

To illustrate the construction of the **BILBO** circuits, there is an example of a dedicated register controlled by two signals $\mathbf{B_1}$ and $\mathbf{B_2}$, which define one of the above circuit configurations as follows:

- $\mathbf{B_1, B_2 = 00}$ multiple input signature analyzer,
- $\mathbf{B_1, B_2 = 01}$ test signature generator,
- $\mathbf{B_1, B_2 = 10}$ parallel register,
- $\mathbf{B_1, B_2 = 11}$ serial register.

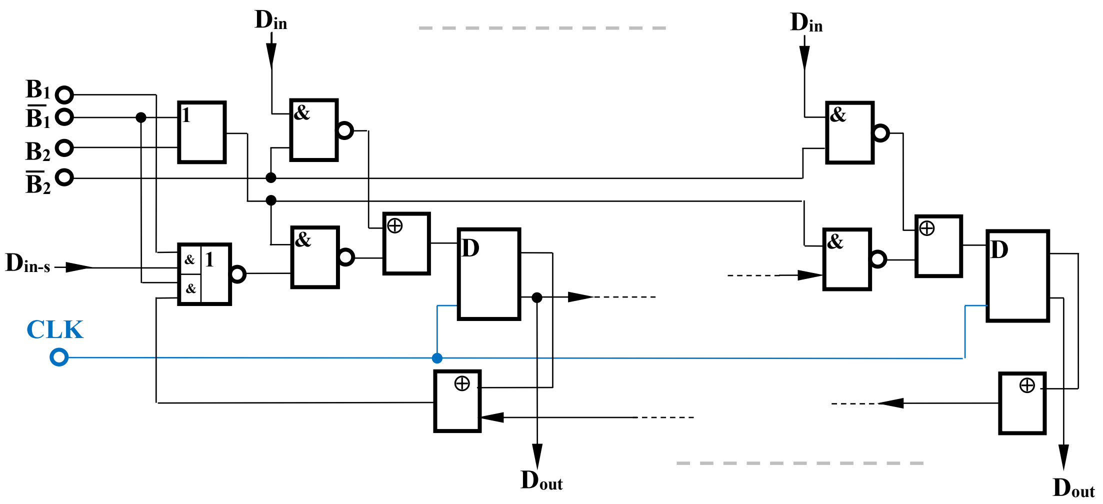


Figure 15.9. Diagram of the BILBO

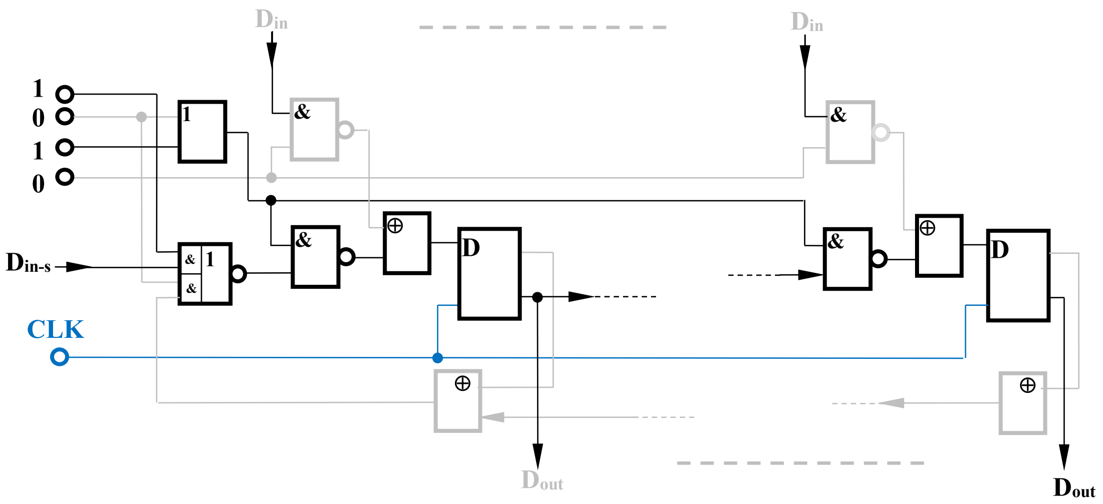


Figure 15.10: BILBO as scan register (serial registr)

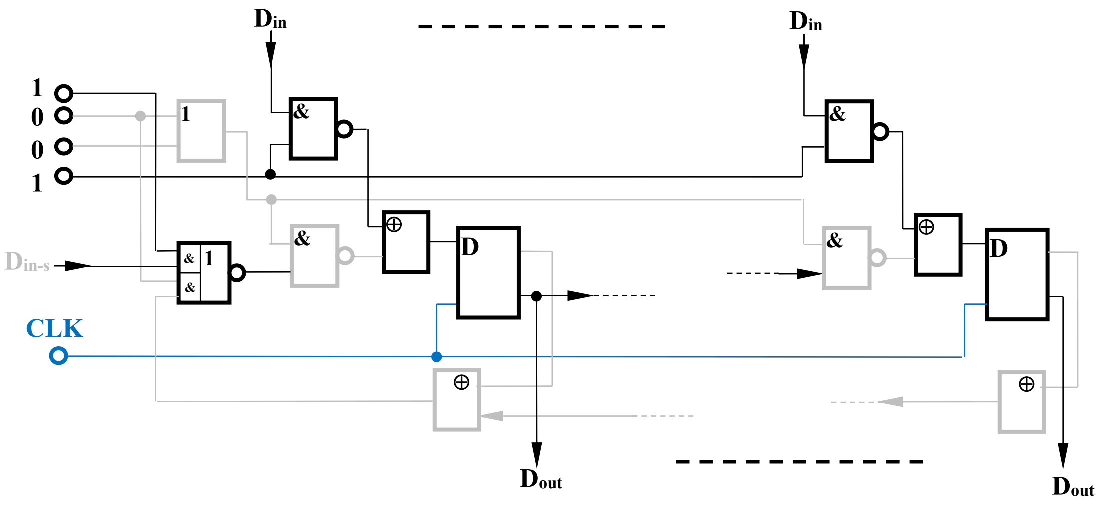


Figure 15.11: BILBO as paralel registr

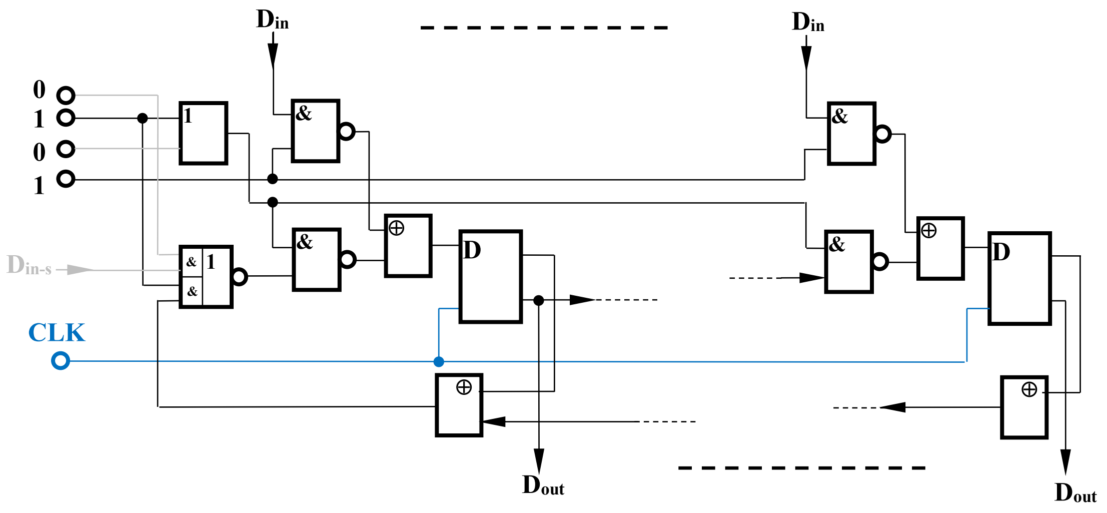


Figure 15.12: BILBO as more inputs signature analyzer

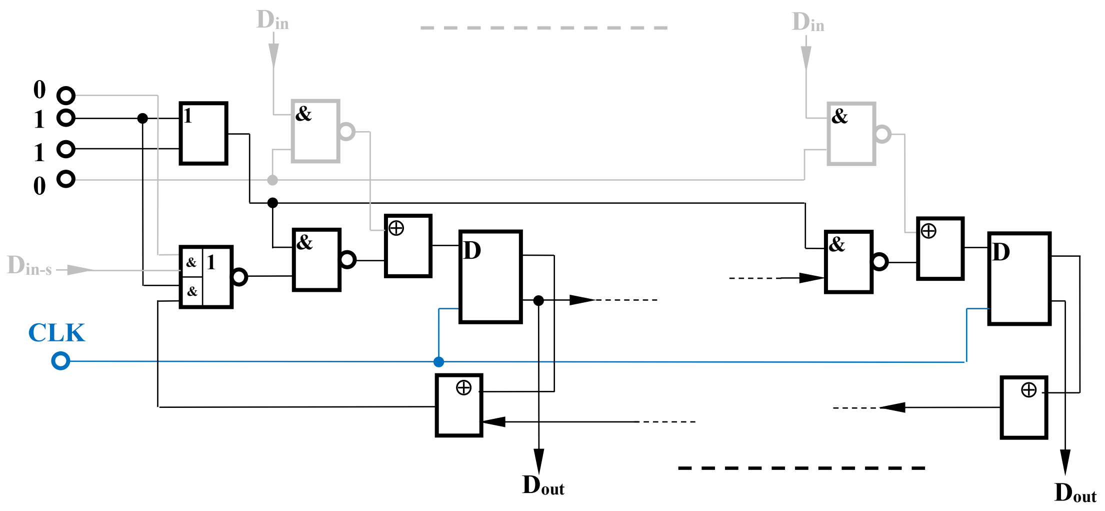


Figure 15.13: BILBO as generator test vectors

Note:

In the previous figures, the passive parts of the BILBO register are grayed out.

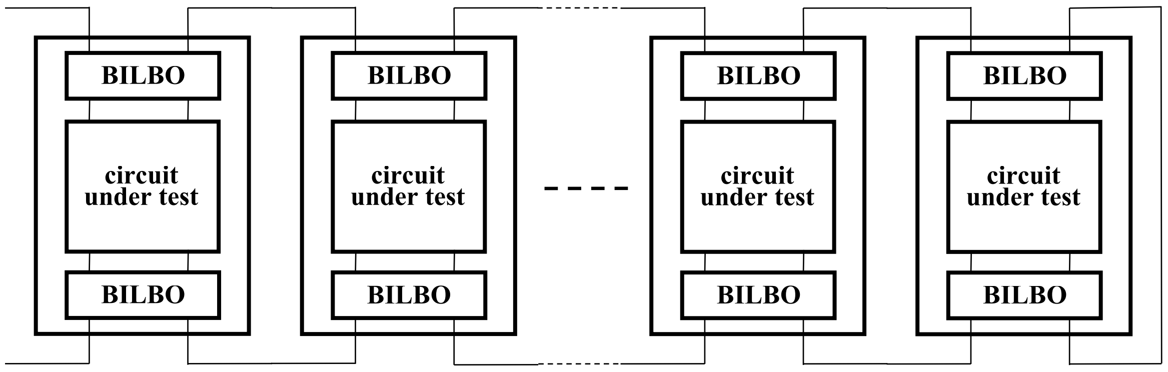


Figure 15.14: Implementation of the BILBO into digital system

The BILBO circuit can be used for both external and internal testing of large digital circuits and systems. It can be implemented in technical equipment similar to BOUNDERS SCAN circuits – see Figure 15.14.

15.5 System HILDO - *Highly Integrated Logic Design Observer*

When designing built-in diagnostic tools, there is increasing pressure to reduce the cost of diagnostics. For these reasons, two separate diagnostic parts of the system are combined into one unit. The main feature of the **HILDO** method is that in a single LZPR string, the test generator and response compressor are hidden at the same time. This connection was designed to test VLSI circuits and was labeled **HILDO** - *Highly Integrated Logic Design Observer*.

The **HILDO** is combination of two separate diagnostic system components into a single unit. In a single chain LSFR hides generator and compressor test responses at the same time.

The **HILDO** Registry is basically a *multiple input signature analyzer*. In addition, in each cycle it generates one step of the test on its outputs, it also receives a test response on each of its inputs. This test response is first added to the existing content of module 2, then the entire content moves one space to the right. Thus, the linear feedback register does not go through its usual cycle, because the received responses to a given diagnostic register actually insert a new start state in each cycle.

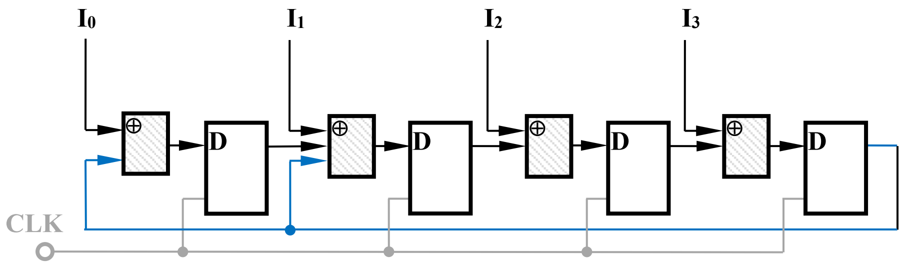


Figure 15.15: Principal scheme of the multiple input signature analyzer

Note:

In the figure - see Figure 15.15 - the bindings of the polynomial are marked in blue.

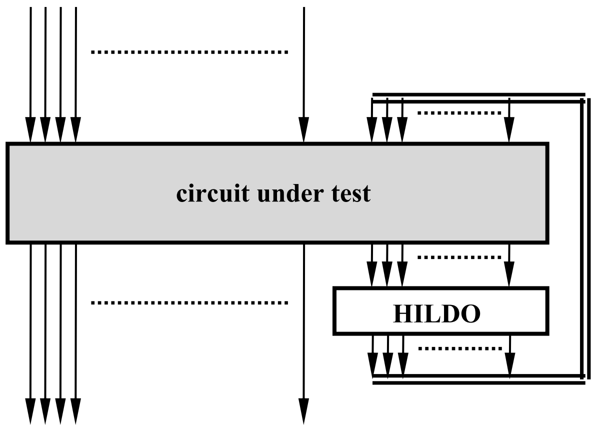


Figure 15.16: Schematic diagram of the register HILDO

List of figures

List of tables