Pseudo-ring testing of the FPGA memory using software Nios processor

Serghei Grițcov, Gherman Sorochin, Tatiana Sestacov
Technical University of Moldova
Chisinau, Moldova
gritscov@gmail.com

***Abstract* — In this paper we review FPGA of the Altera company and we present an example of a 'software' microcontroller based on Nios processor creating. Also an algorithm for memory self-testing was developed and implemented on C language. This algorithm provides self-testing of the Nios and FPGA memory. Also this algorithm is based on pseudo-ring testing methods, which allow to significantly reduce hardware resources for self-realization.**

***Key words* — Field Programmable Gate Array (FPGA), Nios, self-testing, psedo-ring method.**

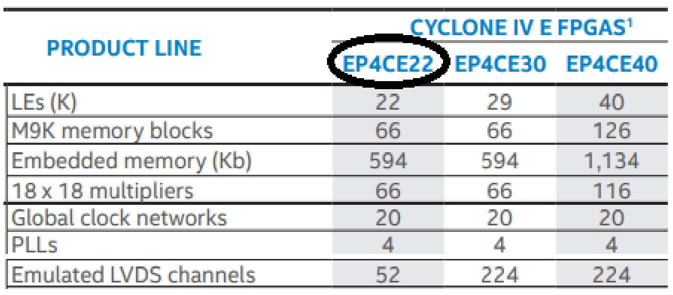
I. Introduction

Most modern electronic systems are based on digital data processing devices such as microcontrollers, DSP (Digital Signal Processor), FPGA (Field Programmable Gate Array), SoC (System-on-a-Chip) and others. If it is necessary to process a large data stream in a short time interval FPGA or SoC are used. Only FPGA or SoC are able to process parallel data on high speed. Many modern FPGAs and SoCs contain built-in MCUs or allow it to be implemented on software level. Using a MCU (microcontroller) with FPGA simplifies programming on FPGA reducing time of the projects development. The implementation of MCU and FPGA based on a single chip requires a lot of resources. Also it is required on many projects that FPGA and SoC should have minimum dimensions and reduced power consumption. This is possible with decreasing the lithography which leads to decreasing the thickness of the conductors and distance between the elements in the chip [1]. This feature leads to increasing of the defects and faults [2]. For the faults detection on FPGA and SoC a BIST (Built-in Self Test) is usually used [3]. Most of the BIST use classical and march testing methods which require significant hardware costs for their implementation [4].

In this article we will consider a pseudo-ring method for FPGA testing which provides much lower hardware costs for its implementation with respect to known techniques [5]. The second paragraph describes FPGA of the Altera company. Also an example of Nios processor generating is given. In the third paragraph pseudo-ring method of memory testing and its features are considered. In the fourth paragraph a few examples of FPGA with MCU memory self-test programs in the C language are given.

II. FPGA of the Altera company and a software MCU implementation

Altera like many other companies produces FPGA and SoC with hardware-integrated MCUs [6]. A significant part of the FPGA doesn't include a built-in MCU. When we implement a project it becomes necessary to have an MCU with FPGA in the project. For such cases, Altera has developed the ability to add a software processor. Let's consider the EP4CE22F17C6 FPGA of the Cyclone IV E family. The main characteristics of this FPGA are shown in Fig. 1.

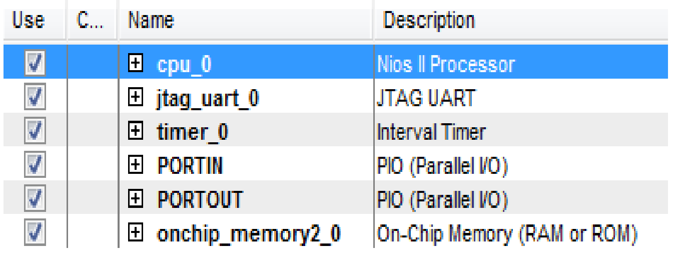


PRODUCT LINE	CYCLONE IV E FPGAS[1]		
	EP4CE22	EP4CE30	EP4CE40
LEs (K)	22	29	40
M9K memory blocks	66	66	126
Embedded memory (Kb)	594	594	1,134
18 x 18 multipliers	66	66	116
Global clock networks	20	20	20
PLLs	4	4	4
Emulated LVDS channels	52	224	224

Fig. 1. Main characteristic of the EP4CE22F17C6 [7].

The main characteristics for this FPGA are 22000 of the logic blocks and 594 Kbits of the embedded memory. It is quite enough for a project based on a FPGA implementation and generating a software MCU.

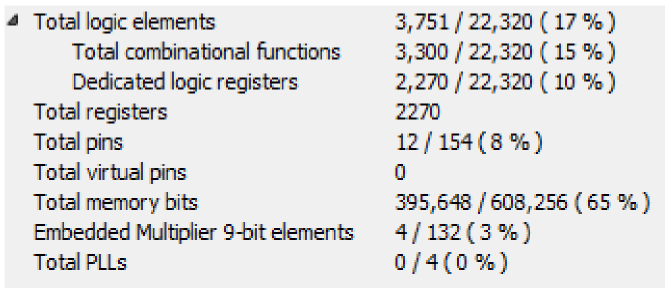
We'll generate a MCU based on this FPGA. For such implementation we can use Quartus II which provides a system of the SOPC Builder library modules. We'll connect 32-bit Nios processor to the MCU, a timer, a port for input and output of data and memory for instructions and data. The resulting architecture is shown in Fig. 2.



Use	C...	Name	Description
✓		⊞ cpu_0	Nios II Processor
✓		⊞ jtag_uart_0	JTAG UART
✓		⊞ timer_0	Interval Timer
✓		⊞ PORTIN	PIO (Parallel I/O)
✓		⊞ PORTOUT	PIO (Parallel I/O)
✓		⊞ onchip_memory2_0	On-Chip Memory (RAM or ROM)

Fig. 2. A software MCU architecture.

We'll compile the project. As the result the resources used for MCU implementation are shown in Fig. 3.



Total logic elements	3,751 / 22,320 (17 %)
Total combinational functions	3,300 / 22,320 (15 %)
Dedicated logic registers	2,270 / 22,320 (10 %)
Total registers	2270
Total pins	12 / 154 (8 %)
Total virtual pins	0
Total memory bits	395,648 / 608,256 (65 %)
Embedded Multiplier 9-bit elements	4 / 132 (3 %)
Total PLLs	0 / 4 (0 %)

Fig. 3. Resources used for MCU implementation.

As we can see from Fig. 3 MCU takes 17% of logical elements and 65% of memory. Thus any project implemented on this FPGA will consist of two parts: a data processing system on the FPGA and a software MCU.

III. Pseudo-ring testing

After the project is implemented on the FPGA with MCU it is necessary to provide the possibility of a self-test of the entire device. From the sources [8] it follows that the most effective from the point of view of hardware costs for self-realization is the pseudo-ring method of testing. Let's consider this testing method more detailed.

The basics of the pseudo-ring testing method includes 3 parameters: 1) the definition of the LFSR (Linear Feedback Shift Register) structure, 2) the setting of the initial state of the LFSR and 3) determination of the moving direction of the LFSR [2]. LFSR is a test sequence generator combined with a result analyzer. The structure of LFSR is given by the structure of an irreducible polynomial [2]. An LFSR can be 'virtual'. It is means that LFSR is implemented on the memory resources [9]. An example of this implementation of the pseudo-ring testing is shown in Fig. 4.

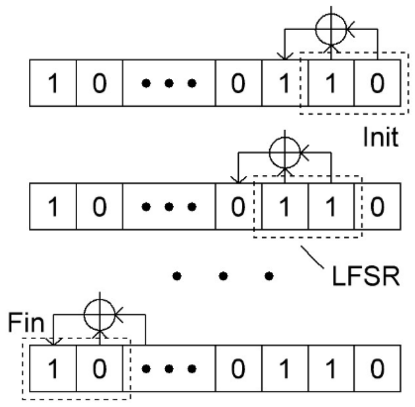


Fig. 4. The idea of the pseudo-ring testing.

The initial state 'Init' is written to the LFSR. The next LFSR value is calculated and the 'virtual' LFSR is shifted at one position to the left. Thus LFSR passes through all the memory cells overwriting their state. If the number of memory cells coincides with the period of an irreducible polynomial that specifies the structure of the current LFSR then the initial and final states of the LFSR must match. In this case, no faults were detected [9]. The algorithmic complexity of this test is 3n: $\Updownarrow\{r_i, r_{i+1}, w_{i+2}(r_i \oplus r_{i+1})\}$.

It is necessary to take into account such parameter as the faults frequency. Conform data which is presented in [10] the most frequent memory faults are SAF (Stack-at Faults). Their frequency is more than 50%. To detect them we can use pseudo-ring testing based on reducible polynomials [11]. An example of LFSR for this test method is shown in Fig. 5.

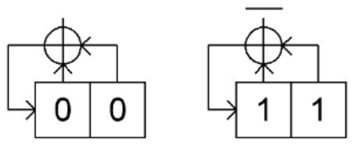


Fig. 5. LFSR based on reducible polynomials.

In this case testing will be realized in two steps as is shown in Fig. 6.

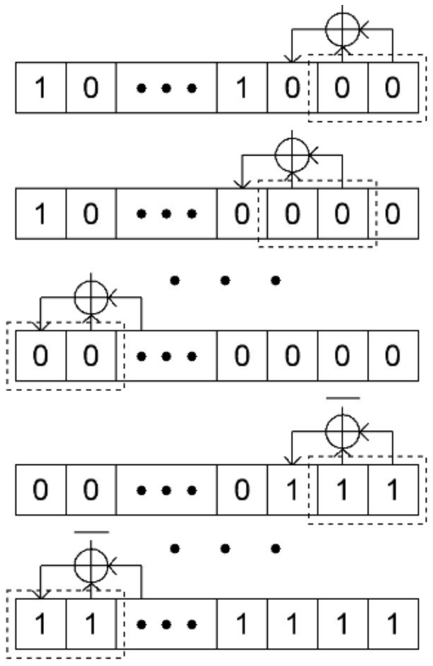


Fig. 6. Pseudo-ring testing based on reducible polynomials.

The difference between classical pseudo-ring testing and this method is that there are only two initial states: reset LFSR to '0' and set all LFSR bits to '1'. The first test detects all faults of type SAF1 (single constant fault '1'), and the second – SAF0. In the second case, after the XOR operation the result is inverted which allows us to assign the value of '1' to all memory bits.

Let's consider pseudo-ring testing implementation based on reducible polynomials.

IV. FPGA self-testing based on pseudo-ring testing

Here is an example of the self-testing program implemented on C language for software FPGA MCU. Testing is divided into two parts: testing the MCU's own memory and testing the memory modules used in the project on the FPGA. The first type of memory is internal to the MCU and the second type of memory is external. The algorithm for memory self-testing is shown in Fig. 7.

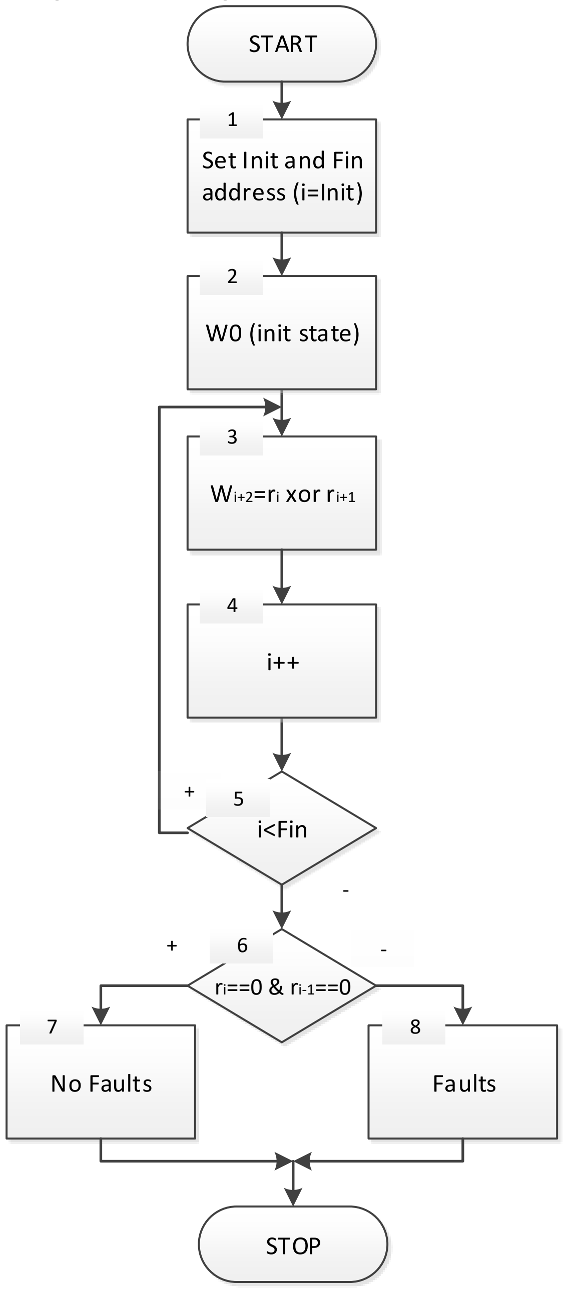


Fig. 7. Self-testing algorithm for the software MCU.

In Fig. 7 block 1 indicates that we must enter the start and end memory addresses. Also the value of the initial memory address is written to the counter 'i'. Block 2 allows us to write 'Init' value in the LFSR. For this case '0' is written to all LFSR bits. In block 3 the next LFSR value is calculated which is written to the next memory cell. In block 4 value of the counter 'i' is incremented by 1. In block 5 it is determined if the LFSR has passed through all the cells of the tested memory. After the LFSR pass through all the tested memory cells it is checked if 'Init' is equal to 'Fin' value. If the equality is fulfilled then we can assume that there is no SAF1 in tested memory.

Let's consider an example of this algorithm implementation for detecting faults SAF0 and SAF1 for MCU internal memory.

```
    unsigned int data     = 0x00000000;
    unsigned int startAddr  = 0x00019000;
    unsigned int endAddr  = 0x00019FFF;
    unsigned int offset  = 0;

    IOWR_32DIRECT(startAddr, offset, data);
    offset++;
    IOWR_32DIRECT(startAddr, offset, data);
    while (offset < endAddr)
    {
        offset++;
        reg1 = IORD_32DIRECT(startAddr, offset-2);
        reg2 = IORD_32DIRECT(startAddr, offset-1);
        IOWR_32DIRECT(startAddr, offset, (reg1 ^ reg2));
// ~(reg1 ^ reg2) for data=0xFFFFFFFF;
    }
    if((IORD_32DIRECT(startAddr, offset-1) != data) &&
(IORD_32DIRECT(startAddr, offset) != data)){
        error = 1;
    }
```

In this code 'data' means the initial LFSR state, 'offset' means the offset of the memory cell address for writing or reading. The 'while' loop executes the testing process: $\Updownarrow\{r_i, r_{i+1}, w_{i+2}(r_i \oplus r_{i+1})\}$. After the loop we check if any faults were found. This code is for SAF1 detecting. For SAF0 detection we need to replace: '(reg1 ^ reg2)' with '~ (reg1 ^ reg2)' in the last line of the loop and the value of 'data' from '0x00000000' to '0xFFFFFFFF'.

If external memory testing is performed the data is transmitted to the I/O port. Also it is necessary to configure control lines and external memory addresses.

```
    static void write(unsigned int addr, unsigned int data)
    {
        IOWR_ALTERA_AVALON_PIO_DATA(WRITE_
EN, 0);
        IOWR_ALTERA_AVALON_PIO_DATA(ADDRES
S, addr);
        IOWR_ALTERA_AVALON_PIO_DATA(CONTR
OL, 0);
        delay(10);
        IOWR_ALTERA_AVALON_PIO_DATA(DATA,
data);
        IOWR_ALTERA_AVALON_PIO_DATA(CONTR
OL, 1);
        delay(10);
        IOWR_ALTERA_AVALON_PIO_DATA(WRITE_
EN, 1);
    }
    static unsigned int read(unsigned int addr)
    {
            IOWR_ALTERA_AVALON_PIO_DATA(READ_E
N, 0);
            IOWR_ALTERA_AVALON_PIO_DATA(ADDRES
S, addr);
            IOWR_ALTERA_AVALON_PIO_DATA(CONTR
OL, 0);
            delay(10);
            IOWR_ALTERA_AVALON_PIO_DATA(CONTR
OL, 1);
            data =
IORD_ALTERA_AVALON_PIO_DATA(DATA);
            delay(10);
            IOWR_ALTERA_AVALON_PIO_DATA(READ_E
N, 1);
    }

    write((startAddr+offset), data);
    offset++;
    write((startAddr+offset), data);
    while (offset < endAddr)
    {
            offset++;
            reg1 = read(startAddr+offset-2);
            reg2 = read(startAddr+offset-1);
            write((startAddr+offset), (reg1 ^ reg2));
    }
    if((read(startAddr+offset-1) != data) &&
(read(startAddr+offset) != data)){
            error = 1;
    }
```

In this code 'write' function means writing in one memory cell of the external RAM. The 'read' function is intended for reading data from one memory cell. 'CONTROL_EN', 'WRITE_EN' and 'READ_EN' are memory control outputs. The rest of the code repeats the code described in the first example: initializing LFSR performing pseudo-ring testing and comparing the final state of the LFSR with the initial.

These examples allow us to perform a self-test of the both the built-in MCU memory and external memory which can be used in the project on the FPGA. These examples can be connected to any projects on the MCU Nios as additional functions.

Thus the implementation of memory self-testing based on pseudo-ring methods can be implemented on a few lines of the code which saves time for project development.

V. Conclusions

In this paper we considered FPGA of the Altera and the software-generated MCU based on Nios processor. Requirement of an additional MCU in modern projects on the FPGA leads to increasing of the size and cost of the device. Using of the 'software' MCU based on FPGA resources avoids these costs. As shown in the paper 'software' MCU took 17% of the logical elements and 65% of the FPGA EP4CE22 memory which is quite acceptable if a project is implementing with using logical part of this FPGA. Also the paper presents the algorithm and code of programs for MCU based on the Nios processor. This algorithm is based on pseudo-ring methods of the memory self-testing and allows detecting any SAFs whose frequency of occurrence is usually not less than 50%.

References

[1] Powell, T.J., Wu-Tung Cheng, Rayhawk, J., Samman, O., Policke, P., Lai, S., Bist for deep submicron asic memories with high performance application, Texas Instruments Inc., IEEE Int. Test Conf., 2003, pp. 386-392.

[2] S. Grițcov, A. Ghincul, Gh. Bodean „Autotestarea pseudoinelară a microcontrolerelor nanosatelitului SATUM", ICTEI, Chișinău, mai 2012, p. 260-267.

[3] B. F. Dutton, Ch. E. Stroud, Soft Core Embedded Processor Based Built-In Self-Test of FPGAs. Defect and Fault Tolerance in VLSI Systems, 7-9 Oct. 2009, p. 9.

[4] J. Sunwoo, Ch. Stroud, Built-In Self-Test of Configurable Cores in SoCs Using Embedded Processor Dynamic Reconfiguration, Proc. Int. System-on-Chip Design Conf., October 2005, pp. 174-177.

[5] D. Bodean, Gh. Bodean, Wajeb Gharibi. Pseudo-Ring Testing Schemes and Algorithms of RAM Built-In and Embedded Self-Testing. 14 th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, Cottbus, Germany, April 13-15, 2011, p. 4.

[6] FPGA with embedded processor, 17.03.2018, https://www.altera.com/products/boards_and_kits/embedded-processors-development-kits-and-cards.html.

[7] Cyclone series of the FPGA, 17.03.2018, https://www.altera.com/products/fpga/cyclone-series/cyclone-iv/overview.html.

[8] G. Bodean, "PRT: Pseudo-Ring Testing – A Method for SelfTesting RAM", IEEE-TTTC Int. Conf. On Automation, Quality and Testing, Robotics: AQTR 2002 (THETA 13), Tome 1, Cluj-Napoca, Romania, May 2002, pp. 295-300.

[9] Grițcov S., STRUCTURE CHART OF PSEUDO-RING TESTING AND EVALUATION OF ITS ALGORITHMIC AND HARDWARE COMPLEXITY, ICTEI-2015, Chisinău, pp. 77-78.

[10] S. Yarmolik, A. Zankovici, A. Ivaniuk. Маршевые тесты для самотестирования ОЗУ. Minsk: BSUIR, 2009, p. 270.

[11] S. Grițcov, П-ТЕСТИРОВАНИЕ С ПРИМЕНЕНИЕМ LFSR НА ОСНОВЕ ПРИВОДИМЫХ ПОЛИНОМОВ, ICTEI-2015, Chisinau, pp. 130-131.