Software and Hardware Cooperative Implementation of the Rafflesia Optimization Algorithm

Zonglin Fu[1], Jeng-Shyang Pan[1,*], Yundong Guo[1], and Václav Snášel[2]

[1]College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao 266590, China
zonglin@sdust.edu.cn, jengshyangpan@gmail.com, yundong88@zju.edu.cn
[2]Faculty of Electrical Engineering and Computer Science, VŠB-Technical University of Ostrava, 70032 Ostrava, Czech Republic
vaclav.snasel@vsb.cz
Corresponding Author: Jeng-Shyang Pan (jengshyangpan@gmail.com)

Abstract. With the development of edge technology in the fields of transportation, wireless sensor networks, and the internet of things, more and more intelligent optimization algorithms are implemented in hardware structures. The Rafflesia optimization algorithm (ROA) is a novel intelligent optimization algorithm proposed recently. To observe its performance on hardware, this paper implements the ROA algorithm in a cooperative way of software and hardware, referred to as the FROA algorithm. To convenient testing and accelerated computing, the initialization module, fitness module and update module of the FROA algorithm are deployed on the advanced RISC machine (ARM) platform and the field programmable gate array (FPGA) platform respectively. In the experimental part, we test the FROA algorithm on nine different benchmark functions. The results are compared with those implemented in software. The experimental results demonstrate the effectiveness and superiority of the FROA algorithm.

Keywords: Advanced RISC machine, Rafflesia optimization algorithm, Field programmable gate array.

1 Introduction

Intelligent optimization algorithms provide good solutions for optimization problems in many fields such as wireless sensor networks [1–3], information hiding [4], transportation[5–7], and so on. Typically, intelligent optimization algorithms are deployed and executed in software [8–11]. However, with the development of science and technology such as the internet of things and automation, more and more algorithms are deployed on edge devices and hardware systems [12–14]. Hardware platforms generally have faster processing speeds. It can speed up the operation of the algorithm. At present, some intelligent optimization algorithms have been implemented on the hardware platform, such as the particle

swarm optimization (PSO) algorithm [15–17], the bat algorithm (BA) [18, 19], the genetic algorithm (GA) [20, 21] and so on. In the hardware implementation of these algorithms, most of them are implemented in pure hardware structure [22, 23]. Although pure hardware design can maximize the execution speed of the algorithm, it has a serious drawback. Once the pure hardware design of the algorithm is complete, it cannot be modified. Therefore, it can only deal with one optimization problem, missing the generality of the algorithm [24]. Another hardware implementation of intelligent optimization algorithms is software-hardware co-design. Although it is less parallel than a pure hardware design, it can increase the speed while maintaining the flexibility of the program [25, 26].

The ROA algorithm is a novel intelligent optimization algorithm that we propose in another paper [27]. Its software results on the benchmark test set demonstrate the optimized performance of the algorithm. In order to further understand its effect on hardware, we implemented the ROA algorithm in the way of hardware and software co-design in this paper, named the FROA algorithm. According to the characteristics of the ROA algorithm, the update module with more computation is deployed on the field programmable gate array (FPGA). Some performance of FPGA platform can improve the execution efficiency of the algorithm, such as the parallelism of the platform itself and strong computing power. The initialization module and fitness module are deployed to the advanced RISC machine (ARM). This facilitates the replacement of test functions. The development kit used in this paper is the vivado design suite [28]. It contains three development tools: Vivado high-level synthesis (HLS) [29], Vivado, and Xilinx software development kit (SDK). In the experimental part, the FROA algorithm is tested on nine benchmark functions [30, 31] and compared with the software results. The comparison results demonstrate the superiority of the FROA algorithm in terms of convergence value and execution time.

The rest of the paper is structured as follows: Section 2 introduces the ROA algorithm in detail. Section 3 expounds the details of the software-hardware co-implementation of the FROA algorithm. Section 4 verifies the effectiveness of the FROA algorithm through experiments. Section 5 summarizes the paper.

2 Rafflesia Optimization Algorithm

The ROA algorithm is a recently proposed intelligent optimization algorithm. It is inspired by the growth properties of the Rafflesia plant. The ROA algorithm consists of three phases: pollination (attracting insects), fruiting (devouring insects), and sowing.

2.1 Pollination phase

This phase corresponds to the exploitation stage of the algorithm. Rafflesia plants emit a scent as they grow to attract some scent-chasing insects. These

insects can pollinate Rafflesia. The population of insects flying towards the target Rafflesia is not fixed. Individuals with poor fitness will be replaced by new individuals. Their numbers are one-third the size of the population. Their positions are updated using strategy 1. The positions of the remaining two-thirds of individuals are updated using strategy 2.

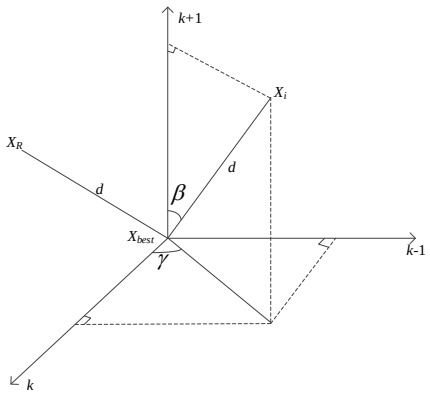

Fig. 1: Example diagram in strategy 1

Strategy 1 We map the individual dimensions k ($k = 1, 2, \cdots, D$) into three-dimensional space to solve. This three-dimensional space consists of $k-1$, k, and $k+1$, as shown in Figure 1. X_{best} represents the best individual. $X_i (i = 1, 2, \cdots, \frac{NP}{3})$ represents the newly added individual. NP is the population size. We assume that the distance from X_i to X_{best} is equal to the distance from random individual X_R to X_{best}. Based on the above conditions, the position update equation of the newly added individual is:

$$X_{i_k} = X_{best_k} + d \times sin\beta_k cos\gamma_k \tag{1}$$

Among them, β_k is the angle between $\overrightarrow{X_{best}X_i}$ and $k+1$ dimension. Its value range is $(0, \frac{\pi}{2})$. $\overrightarrow{X_{best}X_i}$ is a vector composed of X_i and X_{best}. γ_k represents the angle between the k dimension and the projection of $\overrightarrow{X_{best}X_i}$ on the plane formed by k dimension and $k+1$ dimension. Its value range is $(0, \pi)$. d is the distance between X_i and X_{best}.

$$d = \sqrt{\sum_{k=1}^{D} (X_{R_k} - X_{best_k})^2} \tag{2}$$

After that, individuals with poor fitness are replaced by new individuals.

$$X_{worst_i} = X_i \tag{3}$$

Strategy 2 In the process of insect flapping flight, the flight speed is usually represented by translational speed and rotational speed. Among them, the calculation equation of translation velocity is:

$$v_1 = \frac{\omega_0}{2}\sqrt{A^2 sin^2(\omega_0 t + \theta) + B^2 cos^2(\omega_1 t + \theta)} \tag{4}$$

The update equation for the rotational speed of the insect is:

$$v_2 = v_2 \omega_0 cos(\omega_0 t + \theta + \varphi) \tag{5}$$

Where A is the amplitude of the insect wings, with the value of 2.5. B is the lateral offset with the value of 0.1. ω_0 is the period of the flapping frequency. ω_1 is the frequency period of the lateral flapping frequency. The values of ω_0 and ω_1 are both 0.025. θ is the phase, the value is $(2, 2\pi)$. φ is the phase difference between translation and rotation, and the value is -0.78545. t is time and takes the value 1.

The speed of insect flapping flight is the superposition of translational speed and rotational speed, namely:

$$v = v_1 + v_2 \tag{6}$$

The position update of the insect individual is influenced by the optimal individual and the previous state. Its update equation is:

$$X_i = X_i + C \times v \times t + (X_{best} - X_i) \times (1 - C) \times rand \tag{7}$$

Where X_i $(i = 1, 2, ..., \frac{2NP}{3})$ represents the current update individual. C is the impact factor, and its value interval is [-1,1]. The $(1 - C) \times rand$ in the third term reduces the influence of the optimal individual on the current individual and prevents the premature convergence of the algorithm.

2.2 Fruiting phase

Rafflesia flowers that have been pollinated will bear fruit. At the same time, Rafflesia's unique flower chamber structure traps some pollinating insects. According to this characteristic, the ROA algorithm reduces an individual every certain number of iterations. The total number of reductions is about one-third of the initial population size.

2.3 Sowing phase

This phase corresponds to the exploration stage of the algorithm. Ripe fruits contain many seeds. They are scattered randomly all over the place in different ways. Their position update equation is:

$$X_{i_k} = X_{best_k} + rd \times exp(\frac{iter}{Max\_iter} - 1) \times sign(rand - 0.5) \tag{8}$$

Where X_i $(i = 1, 2, \cdots, NP)$ is the current update individual. $iter$ and Max_iter represent the current and the maximum iteration number, respectively. $exp(\frac{iter}{Max\_iter} - 1)$ is the influence factor that varies with the number of iterations.

$$rd = rand \times (ub - lb) + lb \tag{9}$$

Where $rand$ is a random number between $(0, 1)$. ub is the maximum boundary. lb is the minimum boundary.

3 Implementation of FROA

This section describes the detailed design and implementation of the FROA algorithm. The implementation of the FROA algorithm consists of two parts: FPGA part and ARM part. The FPGA is responsible for the update module of the algorithm to achieve its hardware acceleration. ARM is responsible for the initialization module and fitness module to increase the flexibility of the overall structure. The programming language in both FPGA and ARM is C++ language. The random number of the FPGA part is generated using a 32-bit linear feedback shift register (LFSR) [32, 33]. The feedback function is $f(x) = x^{32} + x^{22} + x^2 + x + 1$. The random numbers on the ARM side are generated using the rand() function in the C++ library.

First, the program of the FROA algorithm is processed in Vivado HLS. The Vivado HLS can convert C/C++ code to VHDL/Verilog code. Its use shortens the FPGA development cycle. In the Vavido HLS, the data types we use are mainly floating-point data. We use the advanced eXtensible interface (AXI) bus to transfer data between the FPGA part and the ARM part. The Vivado HLS allows users to add optimization directives to functions and loops. Therefore, we add the "pipeline" instruction to the loop statement in the FROA algorithm. It can make the code in the loop body execute in parallel to reduce the delay of the program. The rules for adding "pipeline" directives in loops are as follows: (1) The "pipeline" instruction is directly added to the single-layer loop. (2) If the current loop is a double-layer loop and there are no statements between the inner and outer layers, the "pipeline" instruction is added to the inner loop. Otherwise, the "pipeline" instruction is added to the outer loop. After adding optimization instructions, the Vivado HLS synthesizes and verifies the program. The verified program is packaged in the IP core.

The IP core is exported from the Vivado HLS. After that, it is loaded into Vivado's Block Design. Figure 2 shows the layout of the IP core in Block Design. The core modules of the layout diagram are the ZYNQ core and the IP core. Among them, the IP core encapsulates the main structure of the algorithm. It corresponds to the program on the FPGA side. The ZYNQ core executes programs on the ARM side and controls the overall scheduling of the algorithm. The AXI bus is used to transfer data between the ZYNQ core and the IP core. First, the data in the ZYNQ core is outgoing from the $M\_AXI\_HPM0\_LPD$ interface. After the lightweight AXI_Lite bus transmission, the data arrives at the IP core. At the same time, the IP core receives the command from the

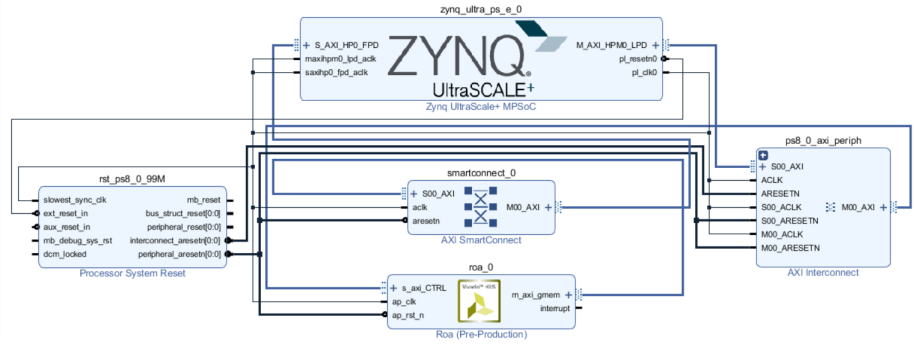

Fig. 2: The layout of the FROA algorithm in Block Design.

ZYNQ to start working. The computing task is completed when the state of the IP core becomes done. After that, the updated data is transmitted from the *m_axi_gmem* interface of the IP core. After the *AXI_master* bus transmission, the data arrives at the ZYNQ core. The ZYNQ re-evaluates the updated data before passing it to the IP core. This is repeated until the stopping condition of the algorithm is reached.

After that, the correctness of the layout and routing diagram of the algorithm is verified. After successful verification, the design source files are generated as external product and bitstream files. They are loaded into the Xilinx SDK along with driver files etc. Finally, the platform is deployed on the development board to complete the overall operation of the program.

4 Experimental Results

In this section, we prove the performance of the FROA algorithm through experiments. The model of the development board we are using is the Xilinx UltraScale MPsoc AXU3EG. We use nine different types of benchmark functions to evaluate the fitness of the population. The 2-dimensional (2D) benchmark functions are: drop-wave function, shubert function, three-hump camel function, levy n.13 function and goldstein-price function. The 10-dimensional (10D) benchmark functions are: sphere function, rastrigin function, sum squares function and styblinski-tang function. The number of evaluations of the algorithm is set to 200*NP. NP is the population size, and the initial value is 30.

4.1 Comparison of results with and without instruction optimization

To understand the effect of the algorithm after adding the "pipeline" instruction, we conduct experiments in terms of convergence value, latency, execution time, and resource occupancy. Table 1 records the resource occupancy and latency in

2D and 10D. Table 2 shows the convergence values and execution times of the FROA algorithm on nine functions.

Table 1: The resource occupancy and latency of the FROA algorithm.

	Total	2D		10D	
		N	Y	N	Y
Latency	-	11025	6762	39413	12586
BRAM	432	8	9	9	9
DSP48E	360	76	57	76	57
FF	141120	9867	10324	9149	9581
LUT	70560	20037	18192	19445	17526

In Table 1, "Latency" represents the delay from input to output of the algorithm. "Total" represents the total number of resources on the board. "Y" and "N" respectively indicate whether a pipeline instruction is added to the algorithm. As can be seen from the table, the optimized algorithm has a significant improvement in latency. In particular, the "latency" of the program optimized on 10D is 2.13 times lower than the original. In addition, the optimized algorithm occupies less DSP48E and LUT resources. This is because the parallel execution of the program makes it unnecessary to repeat some multiplication and logical judgments in the FROA algorithm. But the resource occupancy of the FF has increased. It should be noted that the increase or decrease of a certain resource is not absolute, it is determined by the features of the algorithm and the HLS instructions added by the user.

Table 2: The convergence value and running time of the FROA algorithm.

	Target value	Y		N	
		value	time	value	time
Drop-wave	-1	-0.97449	8	-0.97449	11
Shubert	-186.7309	-186.605	15	-186.605	18
Three-hump camel	0	2.99E-02	11	2.99E-02	14.1
Levy n.13	0	1.72E-06	9	1.72E-06	12
Goldstein-price	0	19.19991	7	19.19991	10
Sphere	0	6.26E-04	16	6.26E-04	36
Rastrigin	0	24.27915	24.7	24.27915	44.8
Sum squares	0	4.00E-02	15	4.00E-02	35
Styblinski-tang	-391.6599	-344.98	40	-344.98	60

In Table 2, "value" represents the convergence value of the algorithm. "time" represents the execution time of the algorithm in milliseconds. "Target value" is

the target optimal value of the benchmark function. The optimization instruction do not change the code and data of the algorithm. Therefore, no matter whether the algorithm adds the "pipeline" instruction or not, its convergence value will not change. In addition, we can know from the table that the execution time of the FROA algorithm with the "pipeline" instruction is reduced. In particular, the optimized algorithm has a more obvious acceleration effect on high-dimensional functions.

4.2 Comparison with software results

The software platform is ARM with 666MHz running memory. Table 3 records the comparison results of the FROA algorithm and the software-implemented ROA algorithm. The convergence value of the FROA algorithm is better than the software result of the ROA algorithm on the eight benchmark functions. Only on the goldstein function, the convergence value of the FROA algorithm is worse than that of the software-implemented ROA algorithm. Additionally, the execution time of the FROA algorithm is faster than the software results in all test functions. Especially for the sphere and sum squares functions, the execution speed of the FROA algorithm is about three times that of the software-implemented ROA algorithm.

Table 3: Comparison with software results in convergence value and running time.

	Target value	FROA		ROA	
		value	time	value	time
Drop-wave	-1	-0.97449	8	-0.9212	19
Shubert	-186.7309	-186.605	15	-147.597	27.4
Three-hump camel	0	2.99E-02	11	1.20E-01	13.3
Levy n.13	0	1.72E-06	9	1.22E-02	17
Goldstein-price	0	19.19991	7	3.031202	14
Sphere	0	6.26E-04	16	13.72556	47.4
Rastrigin	0	24.27915	24.7	68.76995	61.7
Sum squares	0	4.00E-02	15	71.60998	47.3
Styblinski-tang	-391.6599	-344.98	40	-310.178	50

5 Conclusion

In this paper, the ROA algorithm is implemented by the design scheme of software and hardware cooperation, named the FROA algorithm. The computationally intensive update module is executed on the FPGA to achieve hardware acceleration. The initialization module and fitness module are executed on the

ARM to facilitate the replacement of test functions. The experimental results of the algorithm on nine benchmark functions demonstrate its effectiveness and superiority. To test the effect of the FROA algorithm in practical problems, we plan to apply it to problems such as path planning in the future.

References

1. Chai, Q.W., Chu, S.C., Pan, J.S., Zheng, W.M.: Applying adaptive and self assessment fish migration optimization on localization of wireless sensor network on 3-d te rrain. J. Inf. Hiding Multim. Signal Process. **11**(2), 90–102 (2020)
2. Chu, S.C., Du, Z.G., Pan, J.S.: Symbiotic organism search algorithm with multi-group quantum-behavior communication scheme applied in wireless sensor networks. Applied Sciences **10**(3), 930 (2020)
3. Pan, J.S., Dao, T.K., Pan, T.S., Nguyen, T.T., Chu, S.C., Roddick, J.F.: An improvement of flower pollination algorithm for node localization optimization in wsn. J. Inf. Hiding Multim. Signal Process. **8**(2), 486–499 (2017)
4. Chu, S.C., Huang, H.C., Shi, Y., Wu, S.Y., Shieh, C.S.: Genetic watermarking for zerotree-based applications. Circuits, Systems & Signal Processing **27**(2), 171–182 (2008)
5. Pan, J.S., Yang, Q., Shieh, C.S., Chu, S.C.: Tumbleweed optimization algorithm and its application in vehicle path planning in smart city. Journal of Internet Technology **23**(5), 927–945 (2022)
6. Zhang, F., Wu, T.Y., Wang, Y., Xiong, R., Ding, G., Mei, P., Liu, L.: Application of quantum genetic optimization of lvq neural network in smart city traffic network prediction. IEEE Access **8**, 104555–104564 (2020)
7. Cai, Z.M., Lu, J., Ling, Y.F., Li, T.J., Xu, L.: Gsgc: An improved path planning optimization method using guided sampling and gradual cutting. Journal of Network Intelligence **7**, 84–100 (2022)
8. Chai, Q.w., Chu, S.C., Pan, J.S., Hu, P., Zheng, W.m.: A parallel woa with two communication strategies applied in dv-hop localization method. EURASIP Journal on Wireless Communications and Networking **2020**(1), 1–10 (2020)
9. Xue, X., Yang, H., Zhang, J.: Using population-based incremental learning algorithm for matching class diagrams. Data Science and Pattern Recognition **3**(1), 1–8 (2019)
10. Chiang, S., Chu, S.C., Hsin, Y.C., Wang, M.H.: Genetic distance measure for k-modes algorithm. International Journal of Innovative Computing, Information and Control **2**(1), 33–40 (2006)
11. Xi, J., Chen, Y., Liu, X., Chen, X.: Whale optimization algorithm based on nonlinear adjustment and random walk strategy (2022)
12. Mustafa, E.M., Elshafey, M.A., Fouad, M.M.: Enhancing cnn-based image steganalysis on gpus. J. Inf. Hiding Multim. Signal Process. **11**(3), 138–150 (2020)
13. Kang, L., Chen, R.S., Chen, Y.C., Wang, C.C., Li, X., Wu, T.Y.: Using cache optimization method to reduce network traffic in communication systems based on cloud computing. IEEE Access **7**, 124397–124409 (2019)
14. Kumari, A., Kumar, V., Abbasi, M.Y., Kumari, S., Chaudhary, P., Chen, C.M.: Csef: cloud-based secure and efficient framework for smart medical system using ecc. IEEE Access **8**, 107838–107852 (2020)
15. Pan, T.S., Dao, T.K., Chu, S.C., et al.: Optimal base station locations in heterogeneous wireless sensor network based on hybrid particle swarm optimization with bat algorithm. Journal of Computers **25**(4), 14–25 (2015)

16. Cavuslu, M.A., Karakuzu, C., Karakaya, F.: Neural identification of dynamic systems on fpga with improved pso learning. Applied Soft Computing **12**(9), 2707–2718 (2012)
17. Kang, L., Chen, R.S., Xiong, N., Chen, Y.C., Hu, Y.X., Chen, C.M.: Selecting hyper-parameters of gaussian process regression based on non-inertial particle swarm optimization in internet of things. IEEE Access **7**, 59504–59513 (2019)
18. Dao, T.K., Pan, T.S., Chu, S.C., et al.: Evolved bat algorithm for solving the economic load dispatch problem. In: Genetic and Evolutionary Computing, pp. 109–119. Springer (2015)
19. Ameur, M.S.B., Sakly, A., Mtibaa, A.: A hardware optimization of bat algorithms implemented on fpga. In: 2015 16th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA). pp. 146–151. IEEE (2015)
20. Lei, T., Ming-Cheng, Z., Jing-Xia, W.: The hardware implementation of a genetic algorithm model with fpga. In: 2002 IEEE International Conference on Field-Programmable Technology, 2002.(FPT). Proceedings. pp. 374–377. IEEE (2002)
21. Torquato, M.F., Fernandes, M.A.: High-performance parallel implementation of genetic algorithm on fpga. Circuits, Systems, and Signal Processing **38**(9), 4014–4039 (2019)
22. Zou, X., Wang, L., Tang, Y., Liu, Y., Zhan, S., Tao, F.: Parallel design of intelligent optimization algorithm based on fpga. The International Journal of Advanced Manufacturing Technology **94**(9), 3399–3412 (2018)
23. Juang, C.F., Lu, C.M., Lo, C., Wang, C.Y.: Ant colony optimization algorithm for fuzzy controller design and its fpga implementation. IEEE Transactions on Industrial Electronics **55**(3), 1453–1462 (2008)
24. Becker, J., Hubner, M., Hettich, G., Constapel, R., Eisenmann, J., Luka, J.: Dynamic and partial fpga exploitation. Proceedings of the IEEE **95**(2), 438–452 (2007)
25. Jiang, Q., Guo, Y., Yang, Z., Wang, Z., Yang, D., Zhou, X.: Improving the performance of whale optimization algorithm through opencl-based fpga accelerator. Complexity **2020** (2020)
26. Wolf, W.H.: Hardware-software co-design of embedded systems. Proceedings of the IEEE **82**(7), 967–989 (1994)
27. Pan, J.S., Fu, Z., Hu, C.C., Tsai, P.W., Chu, S.C.: Rafflesia optimization algorithm applied in the logistics distribution centers location problem. Journal of Internet Technology pp. 1–17 (2022)
28. Feist, T.: Vivado design suite. White Paper **5**, 30 (2012)
29. Nane, R., Sima, V.M., Pilato, C., Choi, J., Fort, B., Canis, A., Chen, Y.T., Hsiao, H., Brown, S., Ferrandi, F., et al.: A survey and evaluation of fpga high-level synthesis tools. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **35**(10), 1591–1604 (2015)
30. Molga, M., Smutnicki, C.: Test functions for optimization needs. Test functions for optimization needs **101**, 48 (2005)
31. Ali, M.M., Khompatraporn, C., Zabinsky, Z.B.: A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. Journal of global optimization **31**(4), 635–672 (2005)
32. Wang, L.T., McCluskey, E.J.: Linear feedback shift register design using cyclic codes. IEEE Transactions on Computers **37**(10), 1302–1306 (1988)
33. Savir, J., McAnney, W.H.: A multiple seed linear feedback shift register. In: Proceedings. International Test Conference 1990. pp. 657–659. IEEE (1990)