LOOKING. FORWARD >>

IEEE COMPUTER SOCIETY

The IEEE Computer Society's Student Magazine

Summer 2003 **Vol. 11, No. 2**

Message from the Editor-in-Chief

Mohsen Shaaban & Cengiz Gunay

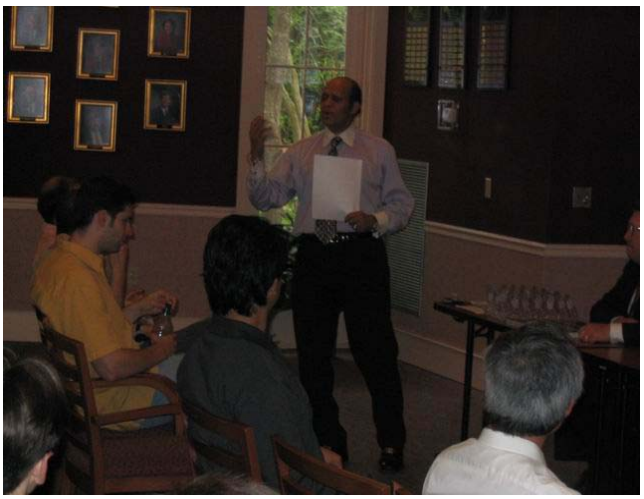
Ph.D. Students
The Center of Advanced Computer Studies
University of Louisiana at Lafayette
mmm5554@cacs.louisiana.edu
cxg9789@cacs.louisiana.edu

Dear Colleagues,

We proudly present the Spring 2003 issue of the Looking Forward E-zine. This issue is to reflect the work done by the IEEE Student Group at The Center for Advanced Computer Studies (CACS) at the University of Louisiana at Lafayette (UL Lafayette) [www.cacs.louisiana.edu].

CACS is a research-oriented department of computer science and computer engineering, where the primary missions of the center are to conduct research and provide graduate-level education.

One of the yearly events of CACS is the student paper contest, organized by the IEEE Student Chapter. The winners of the contest are announced at the annual graduation ceremony of the department.



CACS Director Dr. Magdy A. Bayoumi initiating the graduation ceremony.

This ceremony includes honoring the Bachelor's, Master's and Ph.D. graduates of Spring 2003 semester.

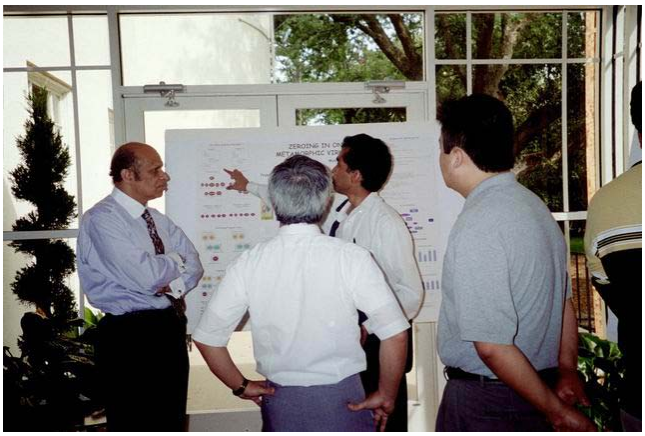
Speech from a graduating student at the ceremony.

Students and faculty attending the graduation ceremony.

Local business and technology leaders are invited as guest speakers, thereby giving a chance for all students to meet them and learn from their experiences.

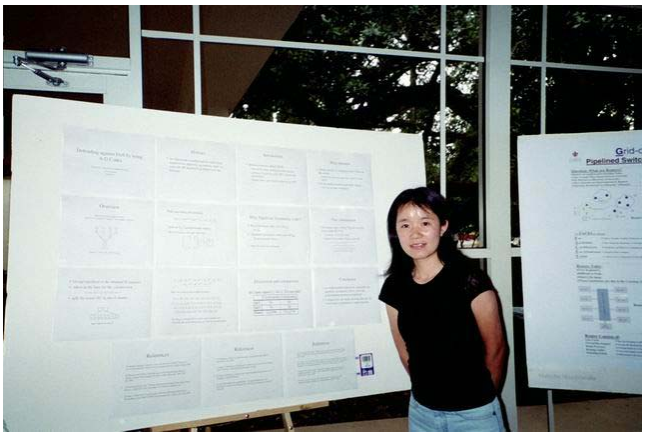
Presenting Lafayette parish industry leaders.

This year the IEEE student chapter organized a poster session for the paper contestants, which gave them a chance to showcase their work.



CACS faculty listening to a student presenting his work at the poster session.

Award prizes were given to all the contestants, including high prizes for the first three places.



First prize winner at our year 2003 student paper contest.

The ceremony also featured talented students playing music and therefore creating a relaxing atmosphere for students and faculty to come together. We, as the IEEE Student Chapter, prepared T-shirts and sold them for fund raising during the ceremony.

Talented CACS students playing music during the event.

IEEE Student Chapter officer selling T-shirts for fund raising.

The papers presented below targets the following fields: Algorithms, Operating Systems, Computer Architectures, Wireless Communications, and VLSI. All these fields are current research areas in CACS.

We present selected articles from this year's student paper competition in this issue. We hope that you find them useful.

Yours sincerely,

Guest Editors,
Mohsen Shaaban and Cengiz Gunay.

Table of Contents:

Peiyi Zhao, Tarek Darwish, and Magdy Bayoumi 5
Low power Conditional-Execution Pulsed Flip-Flop

Chun-yan Bai and Gui-liang Feng 8
Defending against DoS by using A-G Codes

Sumeer Goel and Magdy Bayoumi 14
Improved Hybrid-Latch Flip-Flop for low-power VLSI systems

Moinuddin Mohammed and Arun Lakhotia 19
A method to detect metamorphic computer viruses

Low Power Conditional-Execution Pulsed Flip-Flop

Peiyi Zhao, Tarek Darwish, and Magdy Bayoumi

The Center for Advanced Computer Studies ,University of Louisiana at Lafayette
{pxz6874, tkd5171, mab}@cacs.louisiana.edu

Abstract

We propose conditional execution technique to reduce the redundant switching activity of the internal nodes in flip-flops. Double-edge clocking is utilized to further reduce the power consumption. With a data switching activity of 37.5%, the new conditional execution pulsed Flip-Flop (CEPFF) can achieve 12% improvement in terms of PDP.

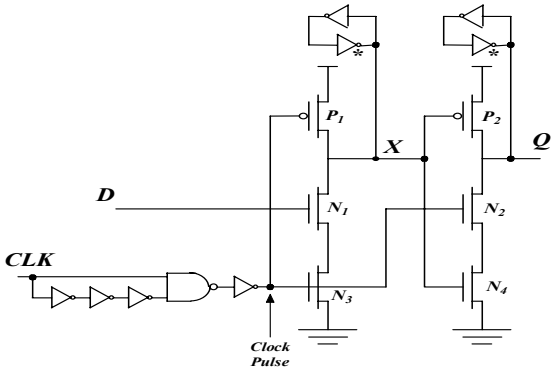
1. Introduction

With chip frequency doubling every two years and feature size shrinking, power consumption becomes one of the main concern for high performance digital systems. Flip flops are used intensively across the whole chip and consumes considerable amount of power.

Different flip-flops can be found in the literature [1-22] include master-slave topology and pulse-triggered topology, etc. For high speed applications, pulse-triggered flip-flops are more suitable for their small DQ delay properties. Pulse triggered flip-flops could be classified into two types: implicit pulse triggered flip-flops (ip-FF) such as HLFF [7], SDFF [8], ip-DCO [9] and explicit pulse triggered flip-flops (ep-FF) such as ep-DCO [9], and the flip-flops in [10] and [11]. One common property among most of these flip-flops is the utilization of dynamic structure to achieve superior performance. However there are large amounts of internal redundant switching activity that cause a lot of wasted dynamic power dissipation [21]. In this paper, we achieve lower power consumption by utilizing conditional execution technique which helps in reducing the switching activity, and double-edge triggering which maintains data bandwidth with lower clock frequency. This paper is organized as follows: Section 2 describes one pulse-triggered flip-flop. Section 3 presents the new flip-flop utilizing the conditional execution power reduction technique. Section 4 simulates these flip-flops. Finally, we conclude in section 5.

2. Explicit Pulsed Flip-Flops

One example of the explicit pulsed flip-flops is the ep-DCO flip-flop [9], Fig. 1. In the precharging phase, the node *X* is pulled up HIGH via P1. At clock pulse rising edge, the transistors N2 and N3 turn on, and the flip-flop will be in the evaluation phase. If the input data *D* is LOW, *X* will stay at the HIGH state and transistor N4 will be on which will discharge the output *Q* to LOW. If *D* is HIGH, *X* will be discharged by transistors N1 andN3. As a result, transistor P2 turns on and pulls Q to the HIGH state. When clock is low, N3 and N2 are off, the



discharging paths are disabled, and the flip-flop is in hold mode.

Figure 1: Explicit-Pulsed triggered Flip-flop, ep-DCO

Careful analysis of the above semidynamic flip-flop reveals a significant amount of power being consumed by charging and discharging the internal node *X* even when the input *D* is stable HIGH and these internal activities do not produce useful operation. Glitches appear at the output that would cause noise problem. To tackle this problem, we propose conditional execution flip-flop.

3. Proposed Conditional Execution Flip-Flop

Conditional Execution Technique is proposed in this paper: an NMOS transistor controlled by *Qb* is inserted in the discharge path of the stage with the high switching activity.

The proposed explicit pulsed Conditional Execution Flip-Flop is shown in Fig. 2. The latching part, which is made of two stages, is activated only during a small window of time (called sampling window) the width of which is specified by the topology of the pulse generator. The double edge pulse generator [9] was utilized to further reduce the power consumed on the clock tree and the clocked transistors in the pulse generator. The flip-flop associated with this double edge pulse generator will have the same data throughput as that of the flip-flop associated with a single edge pulse generator at only half the frequency of the single edge flip-flop. The power saved in the clock distribution network is not included when we compare the power consumption. Although the

input load is increased, significant savings in the overall power is expected.

The first stage of the latching part is responsible for capturing the LOW→HIGH transition of the input. Assume a LOW at *D* was latched in a previous cycle causing the output *Q* and *Qb* to be LOW and HIGH respectively. If later on, *D* undergoes a LOW→HIGH transition and it is captured in the sampling window, the internal node *X* is discharged, since its discharge path is on via *CLK*, *D* and *Qb*. As a result, the output node will be charged HIGH through PMOS transistor P2 and the outputs states for (*Q*, *Qb*) change from (LOW, HIGH) to (HIGH, LOW). Subsequently, if *D* stays HIGH for a long time, node *X* will only precharge to HIGH once and stays precharged afterwards since its discharge path is disabled by *Qb*=LOW. This precharging occurs when *CLK* goes LOW after the first transparent window which captured the LOW→HIGH transition on *D*. Unlike the flip-flops mentioned in section 2, this flip-flop will have no extra switching activity at the internal node X and thus no extra power will be dissipated by the new flip-flop.

Stage two captures the HIGH→LOW input transition. Continuing with the above scenario, if *D* undergoes a HIGH→LOW and it is captured by the sampling window, Y will be HIGH, and N4 will be on, the discharge path of the second stage is enabled causing the outputs (*Q*,*Qb*) to change from (HIGH,LOW) to (LOW,HIGH). If *D* stays LOW afterwards, stage one will be disabled, and stage two will always be enabled maintaining the outputs' states (LOW, HIGH). Since node *X* is not charged and discharged every clock cycle when *D* stays HIGH, no glitches associated with the clock edge appear on the output node *Q*, and thus eliminating the power consumed by the spurious glitches.

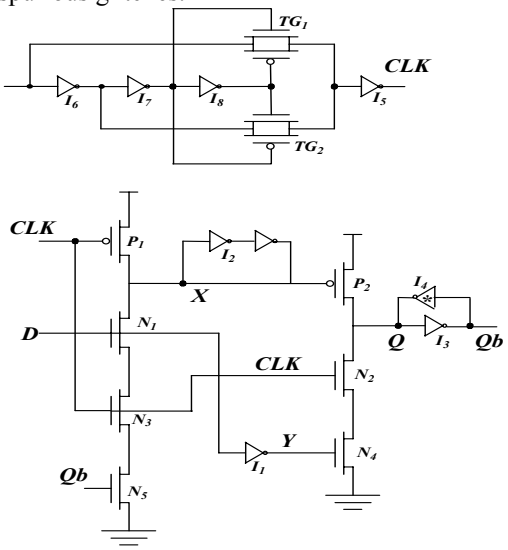


Figure 2: Proposed Conditional Execution pulsed Double-Edge Triggered Flip-Flop

One of the drawbacks of the conditional execution technique is adding one transistor to the NMOS stack of the first stage of the above flip-flop. But as *X* need only drive one PMOS transistor in the flip-flop, the external capacity load of the *X* node is lowered; hence alleviating the negative effect of the increasing stack on delay.

4. Simulation Results

The simulations were done in 1.8-V, 0.18-µm CMOS technology at room temperature using HSPICE. The value of the capacitance load at *Q* is selected to simulate a fan out of fourteen standard sized inverters (FO14) [17]. The setup used in our simulations is shown in Fig. 3, we have supplied *D* with 16-cycle pseudorandom input data with activity 37.5%. A Clock frequency of 250 MHz is used for single edge triggered ep-DCO flip-flop, whereas a 125 MHz frequency is used for double edge triggered CEPFF. Circuits were optimized for minimum power delay product, PDP. The minimum D-to-Q delay [22] is obtained by sweeping the 0→1 and 1→0 data transition times with respect to the clock edge and the minimum data to output delay corresponding to optimum setup time is recorded. This optimization methodology is mainly from[9].

Table 1 shows the minimum D-to-Q propagation delays, average power consumption, and power-delay-product (PDP) for several flip-flops at target D-to-Q delay about 140ps. PDP is reduced in the case of CEPFF by 12% comparing with ep-DCO. CEPFF can have the pulse generator shared by other flip flops to distribute the pulse generator overhead.

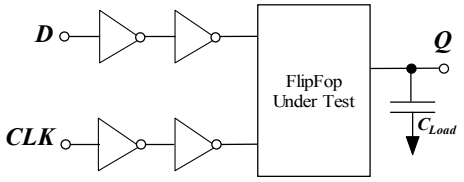


Figure 3: Setup for simulations

Table 1: Comparison of number of transistors, minimum DQ delay, average power and PDP.

	No. of Transistors	Minimum DQ (ps)	Average Power(uw)	PDP (fF)
ep-DCO	26	140	24.2	3.38
CEPFF	30	142	21.0	2.96
HLFF	20	140.7	23.62	3.32

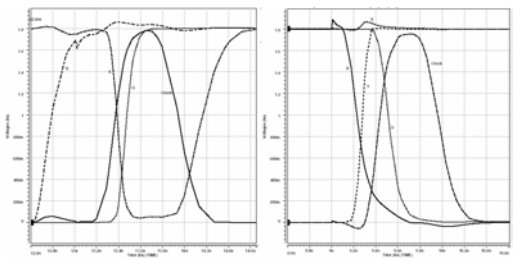


Figure 4: Waveforms of the proposed CEPFF

Figures 4 show the waveforms of the proposed CEPFF. The above proposed CEPFF is used to build a chip of 8-bit counter and has been fabricated by 0.5um technology from MOSIS. Part of the chip layout is shown on Fig. 5. It is clear that Conditional execution technology could be applied to ip-DCO [9], SAFF [24], SDFF [8], etc.

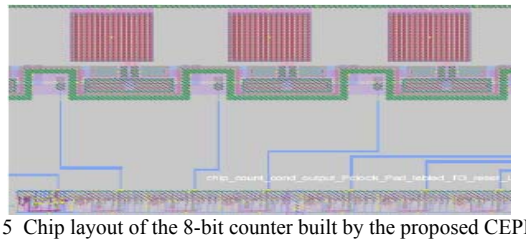


Fig. 5 Chip layout of the 8-bit counter built by the proposed CEPFF

5. Conclusion

In this study, the conditional execution pulsed flip-flop (CEPFF) is proposed to reduce the switching activity of internal node in semidynamic flip-flop. With a data switching activity of 37.5%, the new flip-flops can have 12% improvement in PDP, and eliminate glitches that associated with the clock coupling, so it is suitable for low power, high speed designs.

6. Acknowledgments

The authors acknowledge the support of the U.S. Department of Energy (DoE), EETAPP program, DE97ER12220 and the Governor's Information Technology Initiative. Thanks Mr. Tschanz, Intel for his valuable help.

7. References

[1] H. Kawaguchi, and T. Sakurai, "A reduced clock-swing flip-flop (RCSFF) for 63% power reduction," *IEEE Journal Solid-State Circuits*, Vol. 33, No.5, pp. 807—811, May 1998.

[2] A. Chandrakasan, W. Bowhill, and, F. Fox, *Design of High-Performance Microprocessor Circuits*, IEEE press 2001, New York, 1st edition.

[3] G. Gerosa, "A 2.2W, 80 MHz superscalar RISC microprocessor," vol. 29, no. 12, pp. 1440-1454, IEEE *J. Solid-State Circuits*, 1994.

[4] U. Ko and P. Balsara, "High-Performance Energy-Efficient D-Flip-Flop Circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 8, No.1, pp. 94-98, Feb 2000.

[5] J. Yuan and C. Svensson, "High-Speed CMOS Circuit Technique," *IEEE Journal of Solid-State Circuits*, Vol.24, No.1, pp.62-70, Feb. 1989.

[6] B. Nikolic, V. G. Oklobzija, V. Stojanovic, W. Jia, J.K. Chiu and M.M. Leung, "Improved Sense-Amplifier-Based Flipflop: Design and Measurements," *IEEE J. Solid-State Circuits*, vol. 35, No.6, pp. 876-883, June 2000.

[7] H. Partovi et al., "Flow-through latch and edge-triggered flip-flop hybrid elements," *Digest ISSCC*, February 1996, pp. 138—139

[8] F. Klass, "Semi-dynamic and dynamic flip-flops with embedded logic," in *Symp. on VLSI Circuits, Dig. of Tech. Papers*, June 1998, pp. 108--109.

[9] J. Tschanz, S. Narendra, Z.P. Chen, S. Borkar, M. Sachdev, and V. De, "Comparative delay and energy of single edge-triggered & dual edge-triggered pulsed flip-flops for high-performance microprocessors," *ISPLED'01*, Aug. 2001, Huntington Beach, California, pp. 207- 212.

[10] S. Hesley, V. Andrade, et al., "A 7th-generation X86 Microprocessor," 1999 *IEEE International Solid State Circuits Conference Digest of Technical Papers*, 1999, pp. 92-93

[11] C. F. Webb, C. J. Anderson, et al., "A 400-MHz S/390 Microprocessor," *IEEE Journal of Solid State Circuits*, vol. 32, no. 11, pp. 1665-1675, Nov. 1997.

[12] N. Nedovic and V.G. Oklobdzija, "Hybrid Latch Flip-Flop with Improved Power Efficiency," *Proceedings of the Symposium on Integrated Circuits and Systems Design,* SBCCI2000, Manaus, Brazil, September 18-22, 2000, pp. 211-215

[13] N. Nedovic, M. Aleksic, and V.G. Oklobdzija, "Conditional Pre-Charge Techniques for Power-Efficient Dual-Edge Clocking," *Proceedings of the International Symposium on Low-Power Electronics and Design*, Monterey, California, August 12-14, 2002. pp. 56 –59

[14] Y. Zhang, H. Yang, and H. Wang, "Low clock-swing conditional-precharge flip-flop for more than 30% power reduction," Apr. 2000, Vol.36, No.9, pp.785 -786, *Electronics Letters*.

[15] B.Kong, S.Kim, and Y.Jun, "Conditional-capture flip-flop for statistical power reduction," *IEEE Journal of Solid-State Circuits*, Vol.36, No.8, pp. 1263 –1271, Aug. 2001.

[16] N. Nedovic, M. Aleksic, and V.G. Oklobdzija, "Conditional Techniques for Small Power Consumption Flip-Flops," *Proceedings of the 8th IEEE International Conference on Electronics, Circuits and Systems*, Malta, September 2-5, 2001, pp. 803-806

[17] V.G. Oklobdzija, "Clocking in multi-GHz environment," *23rd International Conference on Microelectronics*, *2002. MIEL 2002,* Volume: 2, 2002, Vol. 2, pp. 561-568

[18] N. Nedovic, M. Aleksic, and V.G. Oklobdzija, "Conditional Pre-Charge Techniques for Power-Efficient Dual-Edge Clocking," Proceedings of the International Symposium on Low-Power Electronics and Design Monterey, California, August 12-14, 2002. pp. 56 –59

[19] P. Zhao, T. Darwish, M. Bayoumi, "Low Power and High Speed Explicit-Pulsed Flip-Flops," *45th IEEE International Midwest Symposium on Circuits and Systems Conference*, Tulsa, Oklahoma, August 4-7, 2002.

[20] Q.wu, M.Pedram, X.Wu, " Clock-gating and its application to low power design of sequential circuits," *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, Vol. 47, No.3, 415- 420, 2000.

[21] R. Ramanarayanan, N. Vijaykrishnan and M.J.Irwin, " Characterizing Dynamic and Leakage Behavior in Flip-Flops." 15th Annual IEEE International ASIC/SOC Conference, September 2002.

[22] V. Stojanovic, and V. Oklobdzija, "Comparative Analysis of Master-Slave Latches and Flip-Flops for High-Performance and Low Power System," *IEEE Journal of Solid State Circuits*, Vol.34, No.4, pp. 536-548, 1999.

Defending against DoS by using A-G Codes

Chun-yan Bai and Gui-liang Feng
Center for Advanced Computer Studies
University of Louisiana at Lafayette, Lafayette, LA 70504, USA.
Email: cxb7146@cacs.louisiana.edu and glf@cacs.louisiana.edu

Abstract

In this paper, IP traceback problem over the Internet was solved by using the algebraic-geometric codes. Comparisons between the algebraic-geometric codes based construction and the Reed-Solomon codes based construction show the feasibility of the AG code based construction.

1. Introduction

One of the major problems on the Internet today is the denial of service (DoS) attack against machines and networks. Most DoS attacks are characterized by a flood of packets with spoofed addresses. Finding the source of these spoofed packets, which we call the *IP traceback problem*, is amongst the hardest security problems to address.

Most prior attempts to solve the IP traceback problem can be described by tracing attacks back towards their origins, in hopes of stopping attackers at the source [6-8]. This approach tries to find out the path information of attacking packets in near real-time, thereby controlling the attacks at far routers. We will review the related work in Section 2.

In this paper, we attempt to find a mathematical expression for algebraic-geometric codes solution to the polynomial reconstruction problem, which is the key step for the IP traceback problem over the Internet. Also, how to reduce the overhead in the IP header is proposed and analyzed in this paper. The result in this paper shows that our scheme can not only be implemented for today's routers, but also can be extended for future use whenever the router IP address need to be enlarged to 48 bits.

2. Related work

There have been several efforts to fight against the DoS attack[2-8]. All these efforts are different in terms of the management cost, additional network load, overhead on the router, ability to trace multiple simultaneous attacks, ability of tracing attacks after they have completed, and whether they are preventive or reactive. Considering the actual situation of today's Internet, we will focus on tracing back the attacking packets.

Tracing packets back to their physical source can be done manually by contacting an ISP and having it test each link to determine if a large number of packets are traversing the link destined for the victim network [3]. This method requires significant cooperation and attention from all the intervening ISPs(Internet Service Provider), which

has proven to be a problem over the Internet. Burch and Cheswick proposed a method of controlled flooding [3], in which the victim floods in a tree-like manner during the attack in order to check the correlation of the flooding with the attack. Hence the victim is able to gather information about the sources of the attacks. But this approach can apply only to on-going attacks. In [4], a scheme is suggested which logs packets at key routers and then use data mining techniques to determine the path that the packets traversed. In [5], the router-generated ICMP traceback messages are used to find the source of attacking packets. Both the schemes have the drawbacks of potentially enormous resource requirements and a large scale inter-provider database integration problem.

Tracing attacking packets can also be achieved by marking packets with IP addresses probabilistically or deterministically [6,7]. In [6], Savage et al. proposed a clever path encoding scheme (FMS) which lets each router along the way probabilistically mark packets with path information during packet forwarding. The victim can reconstruct the complete paths after receiving a modest number of packets that contain the marking. This approach has a low overhead for routers and the network. Also, this approach allows a victim to identify the network paths traversed by the attack traffic without requiring interactive support from ISPs. In [7], two IP marking techniques are presented which are Advanced Marking Scheme and the Authenticated Marking Scheme. These two techniques allow the victim to traceback the approximate origin of spoofed packets with the same low network and router overhead as FMS in [6]. These two approaches are more efficient and accurate for the attacker path reconstruction under DDos. However both the work from [6] and [7] have two disadvantages: the combinatorial explosion during the edge identification step and the few feasible parameterizations.

In contrast to the exact traceback problem, which determines the exact attack path and the associated attack origin for each attacker, the approximate traceback problem defined in [6] finds a candidate attack path for each attacker that contains the true attack path as a suffix. We will continue to address the approximate traceback problem because it is possible that the exact attack origin used for solving the exact traceback problem may never be identified.

3. Overview

The directed acyclic graph (DAG) rooted at V in Figure 1 depicts our example network. The root V represents the victim that is attacked and leaf nodes $A=\{A_1,A_2,A_3,A_4\}$ stand for attackers. Round nodes $R = \{R_1, R_2, R_3,\ldots,R_8\}$ denote routers on the way from attackers A_i,i=1,2,3,4 to the victim V. An attack path from A_i is an ordered list of routers between A_i and V that the attack packet has traversed. For example, the two dotted lines in Figure 1 indicate two attack paths (R_5,R_3,R_2,R_1) and (R_7, R_4, R_2, R_1). The path (R_3,R_5, R_6, R_4, R_2, R_1) will be a valid candidate for the approximate attacking path from attacker A_2 to the victim V.

In [8], a new solution to the problem of approximate traceback is presented. The scheme reframes the traceback problem as a finite field polynomial

reconstruction problem and uses techniques from algebraic coding theory to provide robust methods of transmission and reconstruction. Next, we will summarize the idea of the so-called full-path encoding scheme discussed in [8].

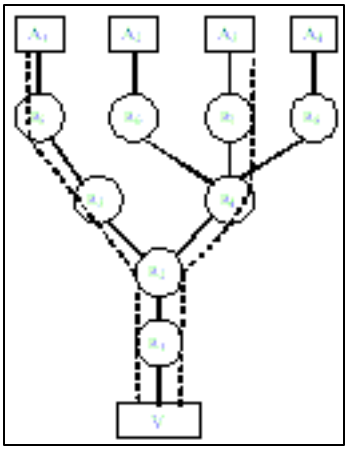


Fig.1 Example network layout

Let f(x) be a polynomial of degree d over the finite field GF(p), we can recover f(x) given f(x) evaluated at (d+1) unique points. Let $A_1,A_2,...,A_n$ be the 32-bit IP addresses of the routers on path P and x_j be the ID for the j^{th} packet. Let

$$f_P(x_j) = A_1 x_j^{n-1} + A_2 x_j^{n-2} + \ldots + A_n,$$

which will be evaluated as the packet x_j travels along the path P, accumulating the result of the computation in a running total along the way. When enough packets from the same path reach the destination, f_P can be reconstructed by interpolation. That is, the defined polynomial can be reconstructed by solving the following matrix equation (1) over GF(p). The right hand side of the equation is a Reed-Solomon codeword. As long as all the x_j's are distinct, the matrix is a Vandermonde matrix (and thus has full rank) that is solvable in $O(n^2)$ field operations.

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{bmatrix} \begin{bmatrix} A_n \\ A_{n-1} \\ \vdots \\ A_1 \end{bmatrix} = \begin{bmatrix} f_P(x_1) \\ f_P(x_2) \\ \vdots \\ f_P(x_n) \end{bmatrix} \quad (1)$$

4. IP traceback based on AG codes

4.1 Background on AG codes

Algebraic-geometric codes are known to be more efficient than Reed-solomn codes in many parameter ranges. They also offer more flexibility in the choice of code parameters. However, in order to understand AG codes, one needs an extensive background in algebraic-geometry. A simpler approach, referred to as improved algebraic-geometric codes makes AG code more accessible [9].

Let us consider the Hermitian curve $x^8+y^7+y=0$ over the finite field $GF(7^2)=\{0,1,\alpha,\alpha^2,\ldots,\alpha^{47}\}$. This curve has a total of 7^3 roots [9]. Improved geometric Goppa codes can be constructed from algebraic-geometric curves based on a well-behaving sequence H [9]. Next, we will show how to construct an improved geometric code from the Hermitian curve discussed above.

Consider the Hermitian curve $x^8+y^7+y=0$ over $GF(7^2)$ again and let $w(x)=7$ and $w(y)=8$, where $w(x)$ is the *weight* of x. A well-behaving sequence H is found to be:

$$H = \{1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3, x^4, x^3y, x^2y^2, ..., xy^5, y^6, x^7, x^6y, x^5y^2, x^4y^3, x^3y^4, x^2y^5, xy^6, y^7, x^7y, x^6y^2, x^5y^3, x^4y^4, x^3y^5, x^2y^6, xy^7, y^8, ...\}.$$

After taking a detailed look at the H sequence, we find that starting from the item x^7, we can divide the rest of the H sequence into groups. Each group includes 8 items

$$y^i \cdot \{x^7, x^6y, x^5y^2, x^4y^3, x^3y^4, x^2y^5, xy^6, y^7\}$$

for *i=0,1,2,…*.That is, each group is formed by multiplying each element of the previous group by *y*. This procedure continues until all the *|H|=343* elements in the sequence are obtained.

With this well-behaving sequence H, an improved algebraic-geometric code of length *n=343* can be defined for different designed minimum distance, e.g. *(343,338,4)* code or *(343, 334, 6)* code following the Construction 2.1 in [9].

4.2 IP traceback based on AG code

Let $l(f_P(x_j))$ be the length of $f_P(x_j)$. From [8] we know that we can trade off bits for packets by splitting a router's IP address into *c* chunks. Inspired by the work in [8] and enriched by the fact that the codeword length of algebraic-geometric codes over GF(q) are not limited to q, we can use algebraic-geometric codes to construct the *FullPath* polynomial. Consequently, we can further reduce the bits needed to encode the *FullPath* by using algebraic-geometric codes instead of Reed-Solomon codes. Next we will demonstrate the construction with details.

Let us once again consider the AG code (length 343) defined by the Hermitian curve $x^8+y^7+y=0$ over GF(7^2). Noticing the group repetition in the obtained H sequence, we take

$$x^7, x^6y, x^5y^2, x^4y^3, x^3y^4, x^2y^5, xy^6, y^7$$

as the base for our polynomial construction. Because the base contains 8 elements, we split the router ID A_j into 8 chunks as shown in Figure 2.

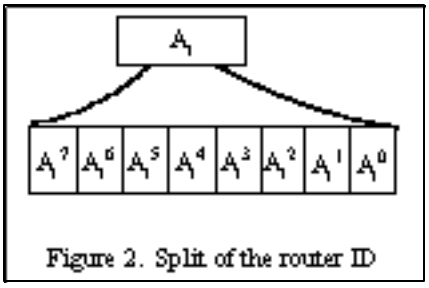


Figure 2. Split of the router ID

Also, we divide the packet ID x_j into two parts (x_j, y_j). Representing each x_j or y_j needs only half number of bits compared to the representation of the original x_j. Thus, the *FullPath* polynomial will be represented as:

$$f_P(x_j, y_j) = (A_n^0 x_j^7 + A_n^1 x_j^6 y_j + A_n^2 x_j^5 y_j^2 + ... A_n^7 y_j^7) + y_j(A_{n-1}^0 x_j^7 + A_{n-1}^1 x_j^6 y_j + A_{n-1}^2 x_j^5 y_j^2 + ... + A_{n-1}^7 y_j^7) + ... + y_j^i(A_{n-i}^0 x_j^7 + A_{n-i}^1 x_j^6 y_j + ... + A_{n-i}^7 y_j^7) + ... + y_j^{n-1}(A_1^0 x_j^7 + A_1^1 x_j^6 y_j + ... + A_1^7 y_j^7).$$

Similar to the Reed-Solomon code based construction in [8], the *FullPath* information $f_P(x_j, y_j)$, together with the packet ID x_j will be passed on to the next router. An additional message

y_jID, which is used to indicate which of the 7 y_js is corresponding to the given x_j over GF(7^2), is also required to transferred to the next router. Each router A_i calculates

$$A_{n-i}^0 x_j^7 + A_{n-i}^1 x_j^6 y_j + \ldots + A_{n-i}^7 y_j^7,$$

then multiplies it with y_j^i and adds the value to the accumulator. As long as enough (8*n in this case) packets are received, the polynomial $f_P(x_j, y_j)$ can be reconstructed by interpolating the running total value. Thus the router ID can be retrieved.

5. Discussion and Comparison

Suppose $l(A_k)=32$ bits, which is reasonable for today's actual IP address, and $l(x_j)=8$ bits in our analysis. If we consider the Hermitian curve constructed above ($x^8+y^7+y=0$ over GF(7^2)), then the number of chunks we split the router IP address into is 8. Thus

$$l(A_k^i) = \left\lceil \frac{32}{8} \right\rceil = 4$$

bits and

$$l(x_j) = l(y_j) = \left\lfloor \log_2^{7^2} \right\rfloor = 6 \text{ bits},$$

where $l(x_j)$ and $l(y_j)$ are the number of bits used for representing elements over the finite field GF(7^2). Consequently, $l(A_k^i) \le \left\lfloor \log_2^{7^2} \right\rfloor$ guarantees that the multiplication operations among x_j, y_j and $l(A_k^i)$ can be performed over GF(7^2). From the inequality $R*(7+1)<7^3$, which is used to make the scheme accommodate the required number R of routers along the way, we have $R \le 42$. That is, the constructed AG code based IP traceback scheme can accommodate at most 42 routers along the way from the attacker to the victim, which is enough to fight back most of the Denial-of-Service attacks. Totally, this scheme requires

$$l(f_P(x_j, y_j)) + l(x_j + y_j ID) = 6 + (4+3) = 13$$

bits per packet in the packet header, where $l(f_P(x_j, y_j)) = \max\{l(A_k^j), l(x_j)\}$ $=\max\{4,6\}=6$ bits. The second item comes from the fact that we divide the 8 bits packet ID into two parts (x_j,y_j), pass only the first part 4 bits $l(x_j)$ and the corresponding y_j index to the next router. The 4 bits $l(x_j)$ will be extended to 6 bits for the polynomial calculation by attaching 2 zeros as the most significant two bits. y_jID=3 because each value of x_j corresponds to 7 values of y_js according to the Hermitian curve $x^8+y^7+y=0$, which needs 3 bits to represent.

Now, let us go back to the RS code based scheme. In order to accommodate at least 42 routers, the best way is to split the router IP address into 4 chunks. Then $l(A_k^i) = \left\lceil \frac{32}{4} \right\rceil = 8$ bits. Thus $l(A_k^i) \le l(x_j)=8$ guarantees that the multiplication operation $(x_j * A_j^i)$ in the $f_P(x)$ polynomial calculation can be performed over the finite field GF(q). From $R*4 \le 2^8$, we have $R \le 64$. Although the RS code based scheme can accommodate more routers, it needs $l(f_P(x_j)) + l(x_j) = 8 + 8 = 16$ bits, which is 3 bits more than the AG code based scheme. This shows an great improvement of the AG code based scheme over the RS code based scheme because free bits in the IP header which can be assigned for traceback purpose is extremely limited. Usually, 42 routers

are enough for resisting the Denial-of-Service attacks. The increase in the number of routers has no great significance in practice. Furthermore, there is one more benefit for the given AG code based scheme. We can extend the router ID from 32 bits to 48 bits without affecting the performance of the scheme. Since $l(x_j) = \frac{48}{8} = 6$ bits is still less than or equal to $l(x_j)=6$ bits, the scheme can work well without making any modification. This is a potential advantage for the AG code based scheme to be used in the future whenever the current 32 bits IP address is not enough.

Conclusion and future work

In this paper, we determined a mathematical expression required for an algebraic-geometric code to solve the polynomial reconstruction problem, i.e., the IP traceback problem over the Internet. The scheme is demonstrated by utilizing and illustrating several useful features of the improved algebraic-geometric codes. Comparisons are made between the algebraic-geometric code based construction and the Reed-Solomon code based construction.

However, this is still an initial step in applying AG codes to the IP traceback problem. Further attempts by using the theoretical analysis are necessary to make the scheme complete from both the practical and theoretical point of views. Furthermore, a strategy which is robust in the presence of incorrect data or data from multiple paths is also urgently sought.

References

1. "Computer emergency response team, cert advisory ca-2000-01:Denial-of-service developments," http://www.cert.org/advisories/CA-000-01.html, 2000.
2. P.Ferguson and D.Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing", RFC 2267, Jan.1998.
3. H.Burch and B.Cheswick, "Tracing Anonymous Packets to Their Approximate Source," 2000 LIXA XIV, December 3-8, 2000, New Orleans, LA.
4. R.Stone, "CenterTrack: An IP Overlay Network for Tracking DoS Floods," Proceedings of the 2000 USENIX security symposium, Denver, CO, July 2000
5. S.M.Bellovin, "ICMP Traceback Messages", Internet Draft:draft-bellovin-itrace-00.txt, Mar.2000.
6. S.Savage, D.Wetherall, A.KArlin and T.Anderson, "Practical Network Support for IP Traceback", SIGCOMM'00, Stockholm, Sweden.
7. D.Song and A.Perrig, "Advanced and Authenticated Marking Schemes for IP Traceback, Technical Report UCB?CSD-00-1107, University of California, Berkeley, June 2000.
8. D.Dean, M.Franklin and A.Stubblefield, "An algebraic approach to IP Traceback," Network and Distributed System Security Symposium, NDSS'01, Feb.2001.
9. G.L. Feng and T.R.N. Rao, "Improved Geometric Goppa Codes Part I: Basic Theory,", IEEE Transactions on Information Theory, vol.41, No.6, 1995.
10. G.L. Feng, V.K. Wei, T.R.N. Rao, and K.K. Tzeng, Simplified Understanding and Efficient Decoding of a Class of Algebraic-Geometric Codes. IEEE Transaction on IT, vol.40, No.4, Jul. 1994.
11. V.Guruswami and M.Sudan, "Improved Decoding of Reed-Solomon and Algebraic-Geometric Codes," IEEE Transactions on IT, vol.45, p1757-1767, 1999.
12. J.H.van Lint and G.van der Geer, Introduction to coding theory and Algebraic geometry, Birkhauser Verlag publisher, Boston, 1988.

Improved Hybrid-Latch Flip-Flop for low-power VLSI systems

Sumeer Goel and Magdy Bayoumi
VLSI Research Lab, Center for Advanced Computer Studies
University of Louisiana at Lafayette

Abstract

This paper presents an enhanced hybrid-latch flip-flop (E-HLFF) that achieves low-power consumption as compared to the hybrid-latch flip-flop (HLFF) without trading-off speed of operation. The technique used to reduce power consumption is to prevent alternate charging and discharging of internal nodes at every clock cycle when no useful work is done. This is achieved by restructuring the dynamic stage of the HLFF and re-ordering the transistors in the critical path. Simulation results show that there is major reduction in power consumption (8% - 40%) and an 18.9% improvement in power-delay product. The proposed design displays high performance at variable load, supply voltage and operating speeds.

1. Introduction

Research shows that timing elements such as flip-flops and latches constitute of almost 30-40% of the integrated system. Reducing power consumption in this fraction can lead to significant power savings in the total system. These timing elements are driven by clock signal(s) for controlling their operation and their performance depend heavily on it. As clock speed is increased, flip-flops are plagued with high power dissipation and clock skew effects. Also, they should be able to satisfy various timing constraints [1] like setup and hold time and data to output latency while keeping power consumption to the minimum possible. A detailed analysis for the selection of an appropriate master-slave flip-flop for high-performance and low power is presented in [2]. The effect of electrical load on delay and power consumption for flip-flop characterization to avoid sub-optimal selection has been shown to be vital in [3]. In [4], they present an analysis of low-energy flip-flops. These studies show that the most commonly used design techniques are conventional master-slave latch-pairs and pulse-triggered latches. Most of the recently reported flip-flops using these design techniques deal with the possibility of a trade off between speed and power consumption. In this paper, we present an enhanced version of the hybrid-latch flip-flop (HLFF) to achieve low-power operation without any reduction in speed of operation. The paper is organized as follows: Section 2 discusses the present flip-flop designs and their disadvantages. In section 3, we discuss in detail the working of the HLFF and its weak points. In section 4 we present our proposed design and section 5 presents the analysis criterion used to analyze the proposed design. Simulation results follow next. Section 7 concludes the paper.

2. Existing Flip-Flop Design

The most commonly used flip-flops for implementing high-performance digital systems are the HLFF [5], sense amplifier-based flip-flop (SAFF) [6] and transmission-gate flip-flop (TGFF) [7]. All of them possess qualities like small D-to-Q delay, capability to absorb clock skew and embedding logic functions into themselves to reduce pipeline stage. The main disadvantage of these designs is that they are inefficient in power consumption. This is attributed to the unwanted internal transitions i.e. charging or discharging of internal nodes even when there is no useful work being done. These unnecessary transitions lead to considerable increase in power expenditure of the flip-flops. The internal power dissipation may be more pronounced for a specific input pattern. This may be attributed either to the working principle or its structure. Some of these flip-flops like HLFF have master-slave configuration in which the master is dynamic and slave is static. The internal transitions in such a case are the precharging and discharging or the internal nodes in the dynamic structure for every clock cycle. In TGFF, considerable power consumption occurs due to heavy clock load and its feedback path. Even when the input switching activity is low, it accounts for a large portion of power consumption. SAFF has a differential structure; every clock cycle, there is a transition in one end of the structure regardless of the input. Each of the above design is suitable for certain application depending upon the availability of power and the speed requirements. A trade-off has to be made considering the best option.

3. Hybrid-Latch Flip-Flop

The HLFF has two stages, first stage is a dynamic master stage and second stage is a static slave stage. The clock (clk), delayed clock (clkd) and data input (D) are fed to the PMOS-transistors P1, P4 and P3 respectively (see Fig. 1). The node X is pulled high when either of these is a low. Essentially, the first stage is a three input NAND-gate. In the second stage, clk and clkd together contribute in latching onto X and its complement is

passed to the output Y. Consequently, the string of NMOS-transistors N4, N5 and N6 act as an inverter.

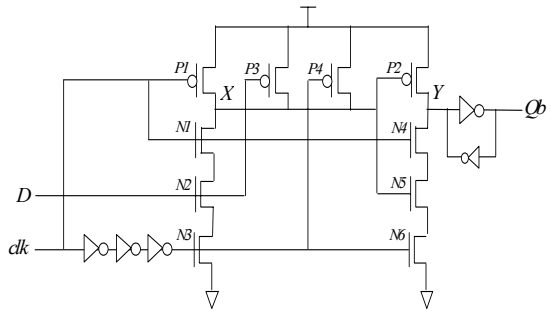


Figure 1. Hybrid-Latch Flip-Flop (HLFF).

Now, consider the case when clk is low and D remains at high i.e. same as previous cycle. So at the output, there is a low and at node Y there is a high. The transistor P1 is ON and precharges to a high making the P2 OFF and N5 ON in the slave stage. The internal node X is at high. Due to this, the input is completely isolated from the output. Now, when the rising-edge of the clock signal appears, it makes P1 OFF and N4 ON. The clkd signal connected to N6 keeps it ON for some time. During this time, the pull-down path to ground in the first stage is completed thereby pulling down node X to low. This causes N5 to break the pull down path in the second stage latching the correct output. If the input D remains high, then there is an alternate charging and discharging of the internal node X. This causes unnecessary power-dissipation when no real work is being done.

Also note that the NMOS-transistor N4, which is the clocked NMOS-transistor at the second stage, causes down shoot at the output at each clock cycle if the output stays high. This happens due to a completed path between ground and node Y i.e. N4, N5 and N6 are all ON simultaneously for a short period of time. As a result, a glitch in the output is produced which can be distinctly noticed in the output waveforms (see Fig. 3). These glitches not only can cause functional failure at a later cascaded stage but also contribute to the redundant power dissipation at every clock cycle.

4. Enhanced HLFF (E-HLFF)

HLFF has a semi-dynamic structure where the first stage i.e. the master stage is dynamic and the second stage i.e. the slave stage is static. As mentioned earlier, in the dynamic stage, there is an alternate precharge and discharge occurring every clock cycle. This happens regardless of an output transition causing unnecessary power loss. This is taken care of by the removing the PMOS transistors that switch every clock cycle i.e. P1 and rearranging the first stage (see Fig. 2).

Now, consider the case when clk is low and D remains high. So at the output, there is a low and at node Y there is a high. The PMOS-transistor P1 is now OFF because of the rearranged inputs. Depending upon the previous cycle, the internal node X maybe at any logic level. Before the rising edge is encountered, the input is completely isolated from the output. Now, when the rising-edge of the clock signal appears, it makes N1 ON. The delayed clock signal connected to N3 keeps it ON for some time and N2 is already ON due to high D input. The pull-down path to ground in the first stage is completed thereby pulling down node X to ground if it was high and maintaining low if it was low previously. Node X makes P2 ON delivering a high at node Y latching the correct output. During this cycle, there is no unnecessary power dissipation. Even if the input remains constant, there is no alternate charging and discharging of the internal node X. This prevents the E-HLFF to make unnecessary power-dissipation when no real work is being done.

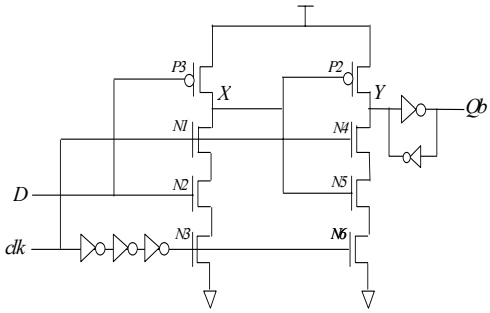


Figure 2. Enhanced HLFF (E-HLFF).

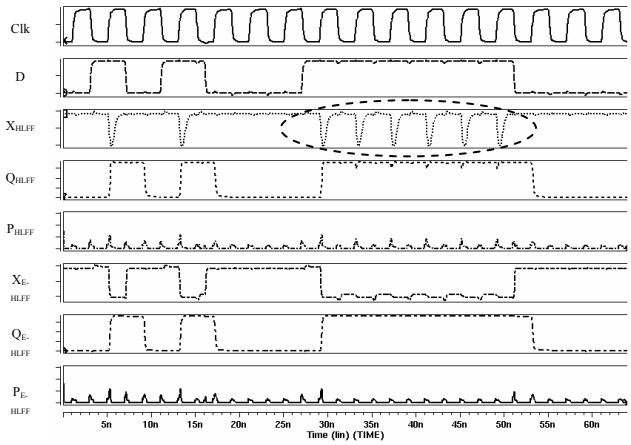


Figure 3. Output waveforms for HLFF and E-HLFF.

Also note that now the NMOS-transistor N5 does not become ON because the node X is not precharging at every clock pulse. Although N4 is ON at rising edge and N6 is ON because of the delayed clock, a completed path between ground and node Y is not possible. The resulting output is glitch free with considerable saving in power consumption. Note that rearranging the input stage does not change either the clock load or the input load. There is no loss in operating speed after these modifications.

These results can be graphically verified by the waveforms generated for each design (see Fig. 3**)**. For HLFF, when input remains high, there is internal switching at node X_{HLFF} (circled region in figure). Also note the presence of glitches in the output Q_{HLFF}. For the same input pattern, there are no redundant transitions at node $X_{E\text{-}HLFF}$ and no output glitches. The cumulative effect can be noted by comparing the spikes in P_{HLFF} & $P_{E\text{-}HLFF}$. Every power spike at the rising edge is bigger than the spike in the E-HLFF. These waveforms also highlight the noise-tolerant [9] behavior of this design. The noise spikes present at the clock and D inputs do not affect the output levels at any time. As compared to the HLFF, the E-HLFF displays more noise-tolerance.

5. Analysis Criterion

5.1. Power Considerations

Power consumption of a circuit depends strongly on the structure and statistics of the applied inputs. As shown earlier, the structural changes made in HLFF effect the power consumption. To make a fair comparison, we conduct power measurements using data patterns comprising the worst, average and best cases for switching activity. We use three different input patterns. Assuming uniform data distribution, the first pattern is a 32 cycle pseudorandom input pattern comprising of '11111111001100110000000010101010' [2]. '1111...' and '0000...' represent a switching activity of 0, '0011...' represents a switching activity of 0.5 and '1010...' represents a maximum switching activity of 1. The second pattern used is a 96-clock cycle input pattern. The first 16 cycles input is low and for next 16 cycles, input is high. Then on, the input changes every single cycle, every two cycles, every four cycles and every 8 cycles, each for 16 cycles [8]. The last sequence is a random input pattern. These input patterns are shown in Figure 4.

5.2. Timing Considerations

In this paper, we investigate the timing characteristics of the flip-flop designs by measuring the clock-to-output delay (D_{CQ}) and the data-to-output delay (D_{DQ}). Both, D_{CQ} and D_{DQ} are calculated for the low-to-high (0-1) and high-to-low (1-0) transitions and the bigger value is kept. For optimum measurement of the D_{DQ} delay, we find the minimum power consumption at a target D_{DQ} delay. The transistors are sized using TILOS algorithm [10] and parasitic information is included in the netlist. There can be many resultant transistor sizes achieving the target D_{DQ} but one with least power consumption is picked. This is repeated for a wide range of D_{DQ} values starting from 200ps going down to 120ps.

5.3. Load Considerations

Flip-flops are abundantly used in the critical paths of a system thereby making their performance vital towards system performance. However, to avoid sub-optimal selection of a flip-flop design for a specific application, studying the effect of absolute load on the performance is important [3]. Here, we study the effect of variable load on the power consumption and D_{CQ} for the HLFF and E-HLFF. The designs are optimized at a given load for minimum power-delay product and this is repeated for a wide range of loads (2.0fF to 16fF).

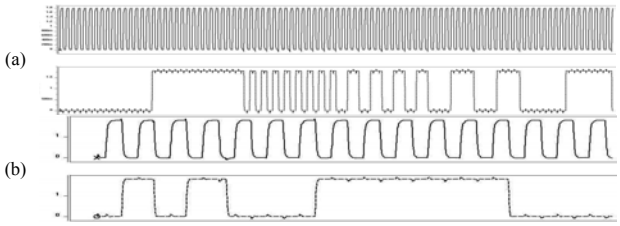


Figure 4(a). 96 cycle test input. (b) Random test input.

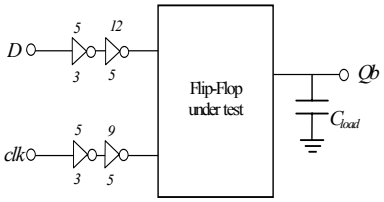


Figure 5. Simulation testbench.

6. Simulation Setup and Results

All the circuits are implemented using MAGIC 7.1 layout tool, extracted using TSMC 0.18-µ technology and simulated using HSPICE. All simulations were carried out with 1.8v V_{dd} at a temperature of 25°C. To simulate real environment, we use input buffers for both the clock and data inputs. The size of these buffers is so chosen that there is sufficient signal distortion expected in an actual circuit. The simulation testbench used is shown in Fig. 5. Note that the power consumption results are inclusive of the power consumed by the input buffers. Since a comparison is being made, it is fair to have this overhead added to both the compared configurations. A constant output load of 5.6fF (equivalent to 14 inverters in 0.18 µ technology) is used for power and delay measurements.

The simulation results for power-consumption, D_{CQ} and power-delay product are compiled in Table 1. At an average, for wave 1 and 2, there is a 19% power saving in the E-HLFF as compared to HLFF. This reduction is achieved solely due to the prevention of the alternate precharging and discharging. The E-HLFF demonstrates a saving of 39.8% when input sequence is '1111...'. This is due to the internal switching at node X for every clock

cycle. E-HLFF shows an improvement of 8.5% when the input is '0000...'. It is low because there is no internal switching activity in either design. The power saving comes from the transistors removed from the HLFF. Under maximum activity, the E-HLFF shows 8.1% power saving. At a switching activity of 0.5, the E-HLFF exhibits a saving of 22.6%. For the whole input pattern, there is an average saving of 21%. These results are shown in Fig. 6.

Table 1: Simulation results at 1.8v V_{dd} and 5.6fF load.

	HLFF		E-HLFF	
	Wave 1	**Wave 2**	**Wave 1**	**Wave 2**
P (μW)	21.09	21.84	17.05	18.07
D_{CQ} (ns)	2.331	1.288	2.317	1.271
PDP (nJ)	49.16	28.13	39.51	22.97

Wave 1: 96 cycle input pattern. Wave 2: Random input pattern.

As observed from Table 1, for both input patterns, there is insignificant change in the delay values (less than 1%) in fact the E-HLFF shows slightly better delay characteristics than its counterpart. This negligible change in the delay is attributed to the removed transistors. Since these transistors did not constitute the critical path they do not affect the delay value by a large amount. Fig. 7 also shows the D_{DQ} vs power-consumption results. The curve for E-HLFF lies above (18% saving) the curve for HLFF showing that the E-HLFF is capable of delivering a constant power saving at different operating speeds. Owing to its structural properties, this behavior is predicted to be similar even at higher speeds and lower D_{DQ} delays ranging down to 20ps. For variable loads, the E-HLFF exhibits better performance both in terms of power and delay displaying a constant improvement for a wide range of load values. The behavior of E-HLFF over a wide range of operating voltages (0.6v – 3.3v) is determined in terms of power and delay (see Fig. 7). These results strengthen the observation that the E-HLFF is capable of delivering higher performance for a wide range of operating speeds, loads and supply voltages.

7. Conclusion

We have presented an enhanced design based on the HLFF. The enhancement was achieved by structural modification made to the standard design. These changes prevented the undesired internal switching at every clock cycle when no useful work was done. The E-HLFF shows an improvement of 8% - 40% in power-consumption without any noticeable loss in operating speed. There is an average improvement of 18.9% in the power delay product of the E-HLFF. These improvements are constant over a wide range of operating speeds, loads and supply voltages making the E-HLFF advantageous in many conditions depending upon the application.

8. Acknowledgements

The authors acknowledge the support of the U.S. Department of Energy (DoE), EETAPP program, DE97ER12220 and the Governor's Information Technology Initiative.

9. References

[1] N. Nedovic, M. Aleksic and V. Oklobdzija, "Timing characterization of dual-edge triggered flip-flops," in Proc. Intl. Conf. on Computer Design, Austin, Texas, 2001, pp 538–541.

[2] V. Stojanovic and V. Oklobdzija, "Comparative analysis of master-slave latches and flip-flops for high-performance and low-power systems," IEEE Jnl. of Solid-State Circuits, Vol: 34, Issue: 4, Apr1999, pp 536-548.

[3] S. Heo and K. Asanovic, "Load-sensitive flip-flop characterization," in Proc. IEEE Computer Society Workshop on VLSI, Orlando, Florida, 2001, pp 87-92.

[4] D. Markovic, B. Nikolic and R. Brodersen, "Analysis and design of low-energy flip-flops," in Proc. Intl. Symp. on Low Power Electronics and Design, Huntington Beach, California, 2001, pp 52-55.

[5] H. Partovi, et al., "Flow-through latch and edge-triggered flip-flop hybrid elements," in International Solid State Circuits Conference, Digest of Technical Papers, February 1996, pp 138-139.

[6] F. Klass, "Semi-dynamic and dynamic flip-flops with embedded logic," in Symposium on VLSI circuits, Digest of Technical Papers, June 1998, pp 108-109.

[7] G. Gerosa et al., "A 2.2 W 80Mhz superscalar RISC microprocessor," IEEE Journal of Solid State Circuits, vol. 29, Dec. 1994, pp. 1440 – 1452.

[8] T. Darwish and M. Bayoumi, "Reducing the switching activity of modified saff flip-flop for low power applications," Proc. 14th Intl. Conf. on Microelectronics, Beirut, Lebanon , 2002, pp 96-99.

[9] S. Goel, T. Darwish and M. Bayoumi, "A novel technique for noise-tolerance in dynamic circuits," Proc. IEEE Comp. Society Annual Symp. on VLSI, Tampa, Florida, 2003, pp 203-206.

[10] J. Fishburn, A. Dunlop, "TILOS: A posynomial programming approach to transistor sizing", Proc. IEEE Int. Conf. Computer-Aided Design, 1985, pp 326-328.

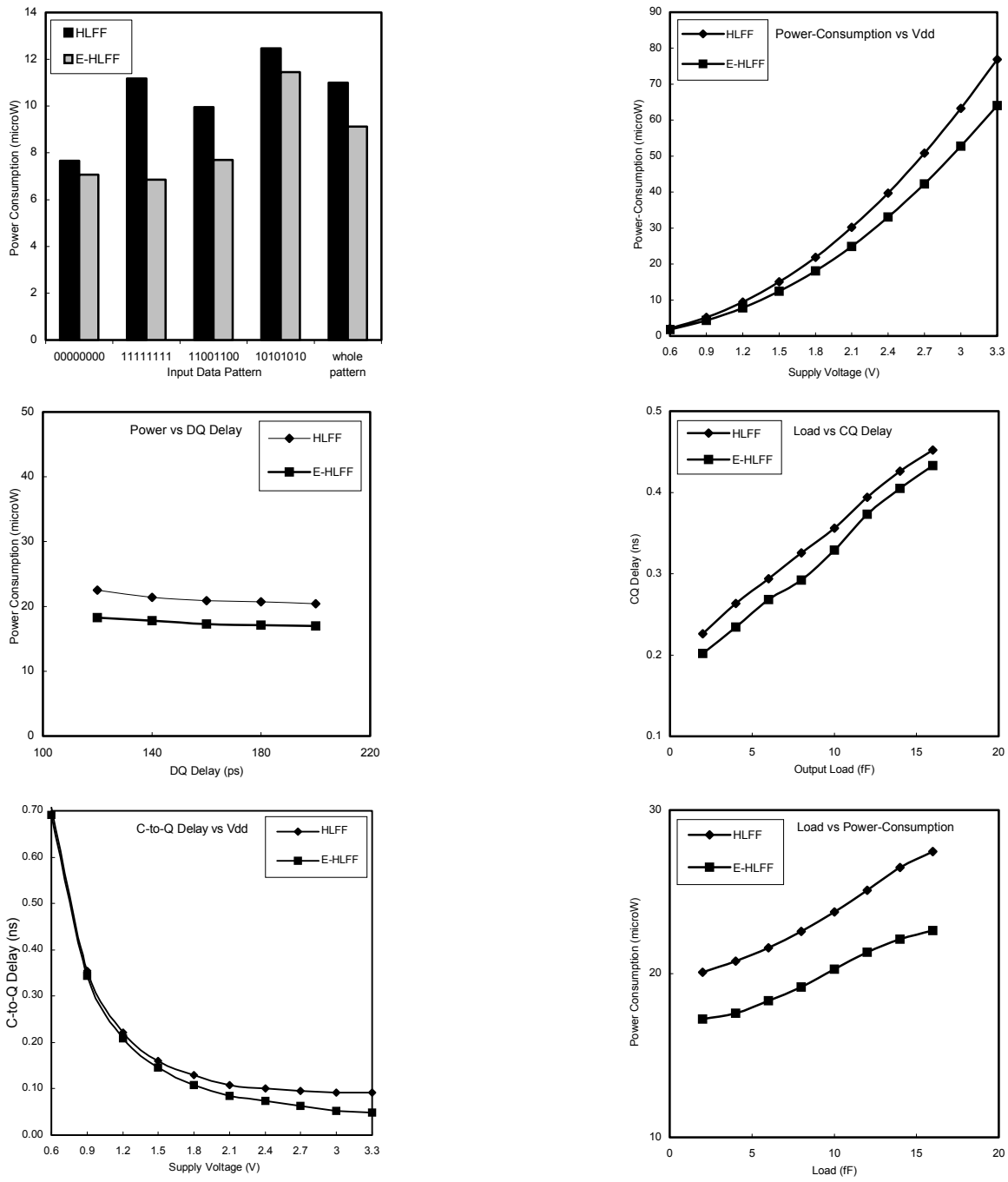


Figure 6. Simulation results comparing the HLFF with the E-HLFF for different input patterns, output loads, supply voltages and operating speeds.

A method to detect metamorphic computer viruses

Moinuddin Mohammed and Arun Lakhotia
{mxm1110, arun} @cacs.louisiana,edu

Software Research Laboratory
Center for Advanced Computer Studies
University of Louisiana at Lafayette

ABSTRACT

Metamorphic computer viruses programmatically vary their instructions to create a different form for each infection. This is done using code evolution techniques, such as, introducing dead code, reordering statements, reshaping expressions, and changing variable names. Current anti-virus technologies use signature, a fixed sequence of bytes, from a sample of a virus, to detect its copies on a user's machine. This technique does not work very well with metamorphic viruses since two versions of a metamorphic virus may have very little in common. This paper presents a method to transform different variants of a metamorphic virus to the same form, called the zero form. Current technologies can be improved to detect metamorphic viruses by using the zero form of a virus, and not the original version, for extracting signature.

Keywords: Computer virus detection, metamorphic computer viruses, anti-virus technology, compiler optimizations, program transformations.

1 Introduction

Computer security is an important concern for any organization that uses computers, which in today's world leaves out very few organizations, if any. A compromise in computer security can cause severe losses in terms of sensitive information, money, time, and reputation of the organization. Most common and damaging security attacks are done using programs called computer viruses and worms. These are computer programs that can rapidly spread from one machine to another. They spread by exploiting some weakness in the existing programs on a computer, or weakness in the security policy of an organization, or by simply fooling the user into executing the programs. Damages caused by viruses and worms are estimated to be in billions of dollars. For example, CodeRed II worm is estimated to have caused damages in excess of $2.6 billion [15].

The number of virus and worm attacks is increasing at an alarming rate. The number of known viruses was about 70,000 in 2002, which is 700% more than the number of known viruses in 1997 [3, 12]. This enormous increase can be attributed to the increasing use of Internet. As the number of machines on the Internet increases, so does the number of target hosts that can be exploited. The most common exploit is to transmit a virus by email. In addition, hackers also exploit the Internet to connect to remote, compromised machines to initiate an attack. They also use compromised machines to give commands to viruses on other compromised machines. Using compromise machines help a hacker in hiding his/her identity.

Though viruses and worms are very complex computer programs, it is not very difficult to write a virus. The recipe for writing such programs is abundantly available on the Internet. There is no need to write these programs from scratch. The simplest method is to modify an existing virus to generate a new one. One does not need to be a programmer to write a virus either. There are many virus generation tools available on the Internet [14]. Using these tools creating a new virus is as easy as selecting its lethality from a menu of options and clicking "ok".

Current AV technologies use virus signature, a sequence of bytes extracted from a sample of a virus, to detect copies of that virus. Thus they can detect a virus if the virus signature extracted in the laboratory of the AV Company is found in a program on user's desktops.

It has been observed before that detecting whether a given program is a virus is an *undecidable* problem. A problem is undecidable if a computer (or a network of computers) cannot

solve it no matter how fast the computer(s) may be. AV technologies are thus limited by this theoretical result. While they can detect a specific virus that is known *a priori* to the technology, they cannot always detect whether an arbitrary program is a virus.

Virus writers exploit this inherent limitation of AV technologies. If a virus is written such that two instances of the virus do not have the same signature, then the virus can evade detection. This is precisely what metamorphic viruses do. A metamorphic virus can modify its own program as it spreads from one host to another [4, 7-10]. The child virus, the one on the newly infected host, may not have the same sequence of bytes as the parent virus. Hence the same signature cannot be used for detecting such viruses.

The experiments conducted by Christodorescu et al. [1] suggest that commercial anti-virus software fails to detect morphed virus variants, the virus variants obtained by changing the program text without changing the virus behavior. If anti-virus were to detect metamorphic viruses using signature scanning approach, they would need to maintain signatures for all possible variants of the metamorphic viruses. This approach is infeasible as the number of signatures to be maintained is too high. More sound methods need to be developed.

In this paper, we present a strategy for augmenting current AV technologies to detect metamorphic viruses. They key contribution is a sequence of transformations called *zeroing transformations* to nullify the effect of the code modifications performed by a metamorphic virus. *Zeroing transformations* are used to map any program to a *zero form*, a single-unique form for all variants of a program created using modifications applied by known metamorphic viruses. The name zeroing transformations is derived from number *zero*. Multiplication of any real number with zero always results in zero. Similarly, application of zeroing transformations on programs result in the zero forms of programs.

The zero form of a program may be used in the laboratory of an AV company to generate a zero signature. The zero signatures may be distributed to the AV scanners on user's desktops. The scanners may also convert a program to a zero form and then match the zero signatures. Since variants of a metamorphic virus will have the same zero form, this method improves the ability of AV technologies in detecting metamorphic viruses. Moreover zero signatures reduce the overhead of maintaining a separate signature for every variant of a virus.

The rest of the paper is organized as follows. Section 2, introduces metamorphic viruses and describes the transformations applied by currently known metamorphic viruses. Section 3 outlines our method for creating a zero form of a program. This method may be used in AV technologies to generate zero signatures. Section 4 presents the related work. Section 5 gives the conclusion and future work.

2 Metamorphic Viruses

A computer virus is a program that infects a host program with its malicious code [2]. The infected host program when executed further spreads the infection to other host programs. Metamorphic viruses are viruses that alter their instructions before spreading to a host. These viruses change their instructions without changing their behavior.

Figure 1 gives a diagrammatic representation of the working of a metamorphic virus. The varying shapes in the Figure 1 suggest different variants of the same virus. The transformations that the virus applies to change its program code (shape) without changing its behavior are called *morphing transformations*.

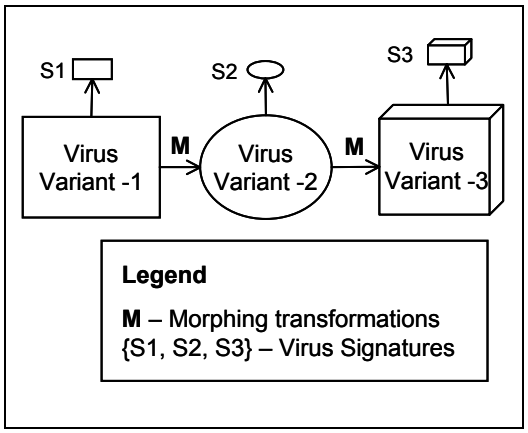


Figure 1: Metamorphic viruses

Definition *Variant of a virus*: A variant of a virus V is a virus V' where V, V' have same behavior but have some difference in the code.

Definition *Morphing Transformations*: Morphing transformations are the transformations that when applied to a virus yields a variant of that virus. These transformations change the shape of a virus, but do not change its behavior.

Definition *Morpher*: The part of virus program logic, responsible for generating different variants of the virus using morphing transformations, is referred to as morpher.

Definition *Metamorphic virus*: Metamorphic virus is a virus that carries a morpher with itself to generate a variant for each infection.

2.1 Morphing transformations

Common morphing transformations used by virus writers are: dead code insertion, variable renaming, statement reordering, expressions reshaping and break & join transformations. This section discusses these transformations.

2.1.1 Dead code insertion

Dead code is the part of program code that is either not executed in the program or has no effect on results of the program. Addition of such code to a program doesn't change its behavior.

Figure 2 shows an example of dead code insertion. Adding *dead-code-1, dead-code-2* and *dead-code-3* to V1 creates V2, a morphed variant of V1. Similarly, addition of *dead-code-4* and *dead-code-5* to V2 creates V3. All the three variants, V1, V2 and V3, have same behavior. If AV software uses the sequence of bytes corresponding to the instructions *xor edx, edx* and *div ecx* as the virus signature, the morphed variants V2, and V3 will get undetected as *dead-code-2* is inserted after *xor edx, edx*.

2.1.2 Variable Renaming

Variable renaming transformation changes variables' names by changing all the instances of a variable with a new name. Morphed variants created by variable renaming have the same behavior, as this transformation doesn't change the program behavior.

Figure 3 shows an example of variable renaming transformations. The example code segment shown in Figure 3 is in assembly language. Renaming variables corresponds to renaming registers in assembly language. Instructions in variants V1, V2 and V3 differ in their usage of registers. The register *edx* is renamed to *eax* from V1 to its morphed variant V2. If the signature for V1 has *edx* in its byte sequence, its morphed variants V2 and V3 will not be detected using that signature.

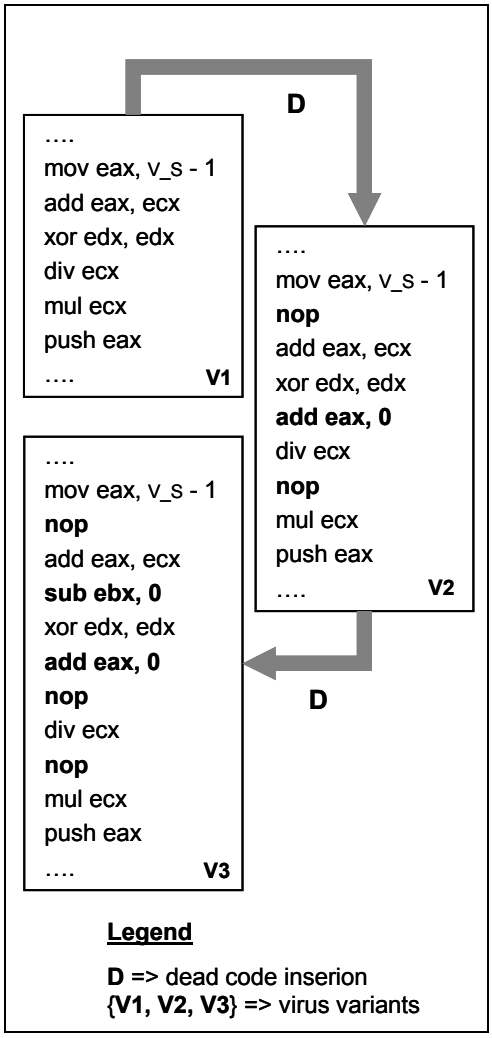


Figure 2: Dead code insertion

2.1.3 Break & Join Transformations

Break & Join transformations break a program into pieces, select a random order of these pieces, and use unconditional branch statements to connect these pieces such that the statements are executed in the same sequence as in the original program.

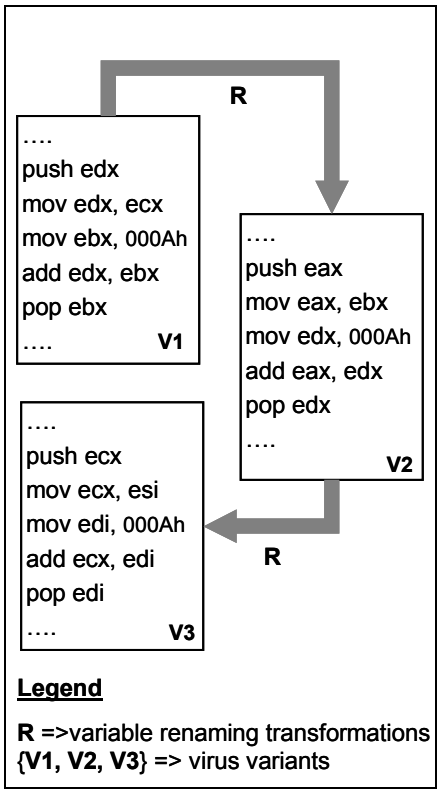


Figure 3: Variable renaming

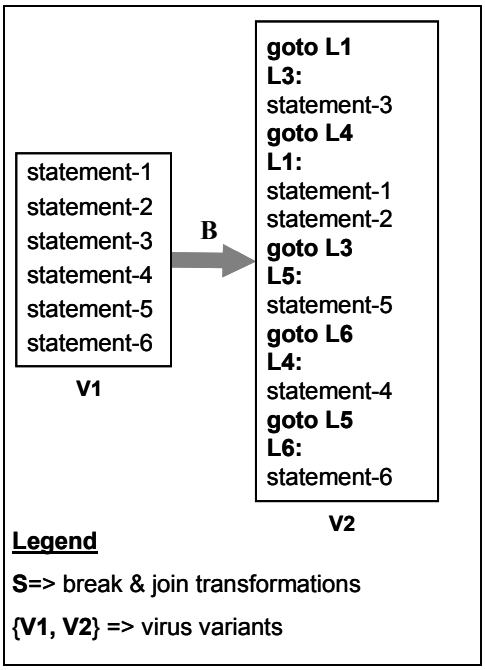


Figure 4: Break & Join Transformations

Figure 4 shows an example of break and join transformation. The order of statements in V2 is different from the order in which these statements appear in V1. Unconditional branch statements (GOTO statements) are used to connect these pieces so that the statements in V1 and V2 are executed in the same order.

2.1.4 Expression Reshaping

Generating random permutations of operands in expressions with commutative and associative operators reshapes expressions in programs. This results in a change in the structure of expression. Expression reshaping doesn't change the behavior of the program.

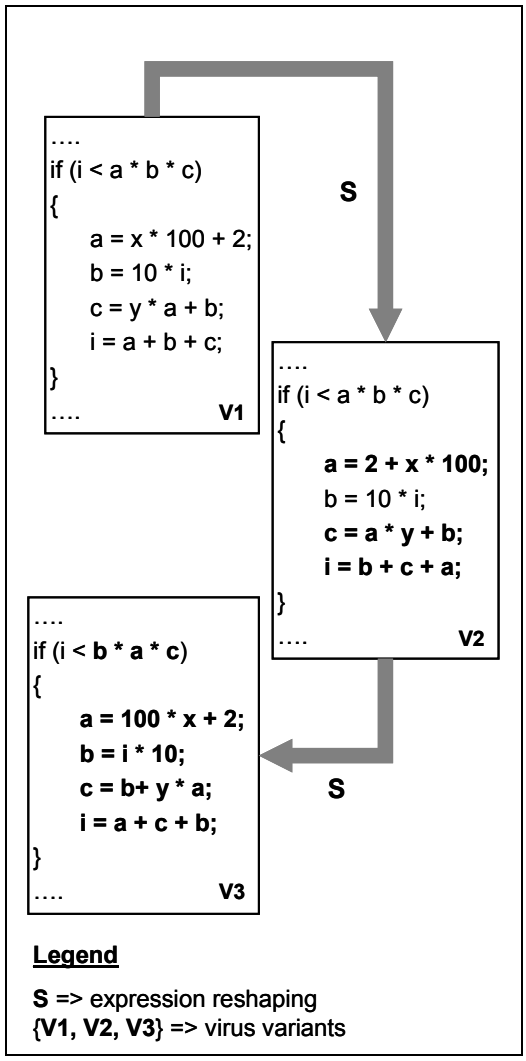


Figure 5: Expression Reshaping

Figure 5 shows an example of expression reshaping transformations. The expression *x*100+2* in V1 is reshaped to *2+x*100* in V2. Behavior of the variants V1, V2, and V3 remains same. If the virus signature of V1 includes the expression

*x*100+2*, V2 and V3 will not be detected by AV software.

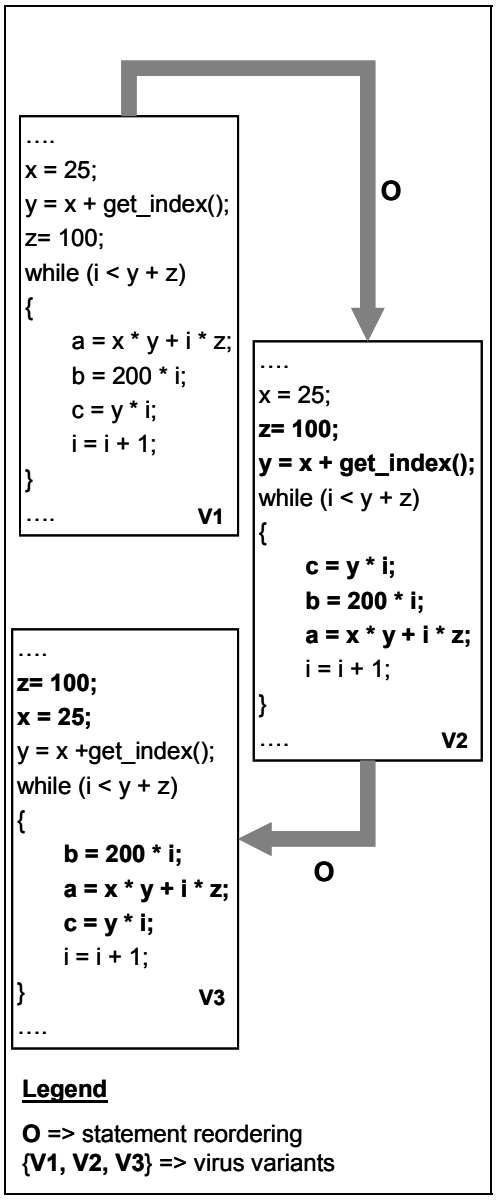


Figure 6: Statement Reordering

2.1.5 Statement Reordering

Statement reordering transformation reorders statements in a program such that the behavior of the program doesn't change. It is possible to reorder statements iff there are no dependences between the statements being reordered [6]. If the virus signature includes bytes corresponding to a statement from this set of reorderable statements, application of statement reordering transformation makes the original virus signature useless for morphed variants.

Figure 6 shows an example of statement reordering transformations. Statements *a=y*i*, *b=200*i,* and *a=x*y+i*z* can be reordered as there are no dependences between these statements. Selection of random permutations of such reorderable statements creates the morphed variants V2, and V3.

3 Detection Approach

We now propose *zeroing transformations*, a set of transformations to nullify the effect of morphing transformations. Zeroing transformations, when applied to a virus result in its zero form. The idea is to apply these transformations on any morphed variant of a virus to get the same form. Figure 7 illustrates the idea of applying zeroing transformations to create zero forms of the viruses. V1, V2, and V3 in Figure 7 are transformed to Vc. AV companies can use this method and store the virus signature extracted from Vc instead of maintaining separate virus signatures for V1, V2 and V3. To use these zero signatures for virus detection, the AV software will need to apply zeroing transformations on the program to be checked for existence of virus behavior. Zero forms of the programs can be searched for zero signatures of viruses.

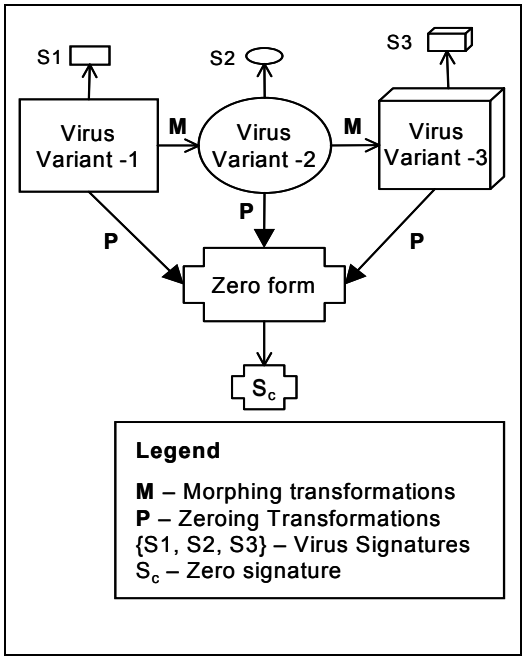


Figure 7: Detection Approach

Figure 8 shows the procedure for creating zero form of a program. A series of transformations are applied to the Input program. These transformations include dead-code elimination [5], constant propagation and removal of redundant computations [5], elimination of spurious unconditional branch statements, reshaping expressions to zero form, fixing an order for the statements that can be reordered and renaming variables to a zero form.

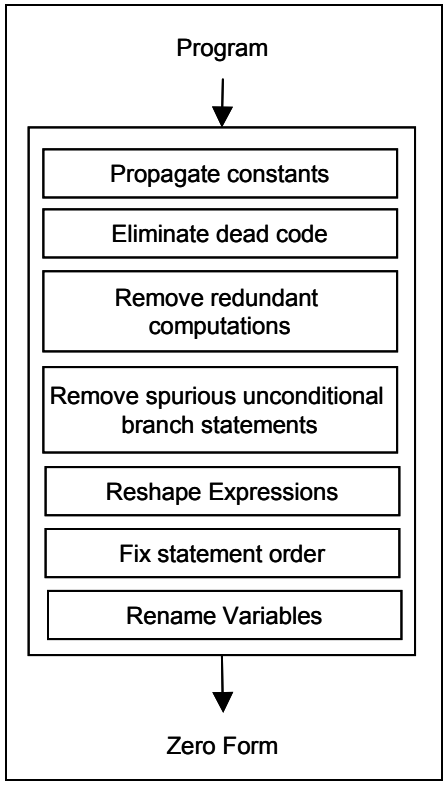


Figure 8: Zeroing Transformations

For fixing an order of the statements in the program [16], we calculate the sets of statements that can be reordered without changing the program behavior and order these sets using a lexicographic ordering based on the syntactic representation of the program statements, which is independent of variable names. As our method follows heuristics, the statement ordering generated by zeroing transformations may not always be the same for all morphed variants of a virus. But in practice, we observed that on an average 94% of the statements in the program could be given a unique order.

4 Related Work

The Bloodhound technology of Symantec Inc., uses heuristics for detecting malicious code [13]. Bloodhound uses two types of heuristic scanners: static and dynamic. The static heuristic scanner maintains a signature database. The signatures are associated with program code representing the different functional behaviors. The dynamic heuristic scanner uses CPU emulation to gather information about the interrupt calls the program is making. Based on this information it can identify the functional behavior of the program. Once different functional behaviors are identified using the static and dynamic heuristic scanners, they are fed to an expert system, which judges whether the program is malicious or not. Static heuristics fail to detect morphed variants of the viruses as morphed variants have different signatures. Dynamic heuristics consider only one possible execution of a program. A virus can avoid being detected by a dynamic scanner by introducing arbitrary loops.

Lo et al.'s MCF [11] uses program slicing and flow analysis for detecting computer viruses, worms, trojan-horses, and time/logic bombs. MCF identifies telltale signs that differentiate between malicious and benign programs. MCF slices a program with respect to these telltale signs to get a smaller program segment representing the malicious behavior. This smaller program segment is manually analyzed for the existence of virus behavior.

Szappanos [10] uses code normalization techniques to detect polymorphic viruses. Normalization techniques remove junk code & white spaces, and comments in programs before they generate virus signature. To deal with variable renaming, Szappanos suggests two methods – first, renaming variables by the order they appear in the program and second, renaming all the variables in a program with a same name. Former approach fails if the virus reorders its statements, and the later approach abstracts a lot of information and may lead to incorrect results. As our approach fixes the order of the statements in a program, the first approach suggested by Szappanos for renaming the variable can be used in combination with our method.

Our work relates to the work done by Christodorescu et al. [1] for detecting of malicious patterns in the executables. They use abstract patterns, patterns with typed variables and instruc-

tion sequences that use these typed variables, to represent the instructions in the program. A problem with abstract patterns is that it becomes difficult for AV companies to distribute virus signatures, as the virus can be reconstructed using these patterns. Their approach gives fewer false positives but the cost of creating and matching the abstract patters is high. They detect the virus variants created by performing dead code insertion, variable renaming, and break & join transformations. Our method, in addition to the above morphing transformations, can detect the computer viruses that apply statement reordering and expression reshaping transformations.

5 Conclusions and Future Work

We have described a method for detecting morphed variants of the viruses. Our approach can augment the current AV technologies such as traditional signature scanning approach, and other static and dynamic detection schemes. We map different variants of a virus to one zero form. The effectiveness of our method is determined by the effectiveness of zeroing transformations that map a program to a zero form. We take into account dead code insertion, statement reordering, variable renaming, expression reshaping, and break & join transformations.

As a future work, we like to do a more detailed investigation on zeroing transformations that map a program to its zero form.

6 References

[1] Mihai Christodorescu and Somesh Jha,"Static analysis of executables to detect malicious patterns," In *12th USENIX Security Symposium*, Washington, DC, 2003.

[2] Fredrick Cohen, "Computer Viruses--Theory and Experiments," *Computers and Security,* pp:22-35, 6(1), 1984.

[3] Jan Hruska, "Computer viruses prevention : a primer," *Sophos,* 2002 http://www.oucs.ox.ac.uk/viruses/documents/artdef.pdf.

[4] Myles Jordon, "Dealing with metamorphism," *Virus Bulletin,* pp:4-6, 2002.

[5] Robert Morgan, "Building an optimizing compiler," *Butterworth-Heinemann,* 1998.

[6] Fleming Nielson, Hanne Riis Nielson and Chris Hankin, "Principles of program analysis," *Springer,* 1999.

[7] Gabor Szappanos, "Polymorphic Macro Viruses, Part One," *Security Focus,* 2002 http://online.securityfocus.com/infocus/1635.

[8] Péter Ször and Peter Ferrie,"Hunting for Metamorphic," In *Procedings of the 11th International Virus Bulletin Conference*, pp:521-541, 2001.

[9] Vesselin Bontchev,"Macro and script virus polymorphism," In *Proceedings of the 12th International Virus Bulletin Conference*, pp:406-438, 2002.

[10] Gabor Szappanos,"Are there any polymorphic macro viruses at all? (… and what to do with them)," In *Procedings of the 12th International Virus Bulletin Conference*, pp:477-477, 2002.

[11] Raymond W. Lo, Karl N. Levitt and Ronald A. Olsson, "MCF: a Malicious Code Filter," *Computers & Security,* pp:541-566, 14(6), 1995.

[12] Lawrence M. Bridwell, "ICSA Labs 7th Annual Computer Virus Prevalence Survey 2001," *ICSA Labs,* 2001.

[13] Symantec, "Understanding Heuristics; Symantec's Bloodhound Technology," 1997.

[14] VX Heavens, "Virus Creation Tools," 2002 http://vx.netlux.org/dat/vct.shtml.

[15] David Moore, Colleen Shannon and Jeffery Brown,"Code-Red: a case study on the spread and victims of an Internet worm," In *2nd Internet Measurement Workshop*, 2002.

[16] Moinuddin Mohammed, "Zeroing in on metamorphic viruses," *Center for Advanced Computer Studies, University of Louisiana at Lafayette,* 2003.