"Glitch Logic" and Applications to Computing and Information Security

Adrian Stoica
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109, USA
adrian.stoica@jpl.nasa.gov

Srinivas Katkoori
CSE Department
University of South Florida
Tampa, FL 33620, USA
katkoori@cse.usf.edu

***Abstract*—This paper introduces a new method of information processing in digital systems, and discusses its potential benefits to computing and information security. The new method exploits glitches caused by delays in logic circuits for carrying and processing information. Glitch processing is hidden to conventional logic analyses and undetectable by traditional reverse engineering techniques. It enables the creation of new logic design methods that allow for an additional controllable "glitch logic" processing layer embedded into a conventional synchronous digital circuits as a hidden/covert information flow channel. The combination of synchronous logic with specific glitch logic design acting as an additional computing channel reduces the number of equivalent logic designs resulting from synthesis, thus implicitly reducing the possibility of modification and/or tampering with the design. The hidden information channel produced by the glitch logic can be used: 1) for covert computing/communication, 2) to prevent reverse engineering, tampering, and alteration of design, and 3) to act as a channel for information infiltration/exfiltration and propagation of viruses/spyware/Trojan horses.**

Keywords-anti-tamper; trusted design; protection to reverse engineering; information exfiltration; stego-design; hardware vulnerabilities.

I. Introduction

Threats to information security (IS) occur at different levels of the information infrastructure. Examples of traditional threats include network attacks exploiting vulnerabilities in the operating system and in the application software. A more recent threat has been identified in the potential exploitation of hardware vulnerabilities, in particular in the context where these vulnerabilities can be maliciously induced and are hard to detect.

The current systems acquisition process has been shown to be fragile in the sense of accepting counterfeit and otherwise tampered components and subsystems through the chain of defense subcontractors. This can ultimately lead to corrupt hardware/systems for defense systems or national infrastructure. By the same account, hardware vulnerabilities could be used to attack enemy infrastructures. Glitch logic is such a hardware-mediated vulnerability.

In this paper, we introduce a glitch logic system, in which the glitches occurring in digital systems are used for computation and to enhance system security. We describe how the proposed glitch logic can be used as a *hidden channel* and its application for security. We also demonstrate how glitch logic can be used as an anti-tamper technique.

The paper is organized as follows. Section 2 provides background on glitches in digital systems. Section 3 describes how glitch logic can be used as a hidden channel and for anti-tampering. Section 4 discusses the benefits of the proposed approach. Section 5 presents the conclusions.

II. Glitches in Digital Systems

Glitches are pulses of very small duration that are caused by different delays on alternative signal propagation paths. On any conventional digital information flow, one can always observe a stream of glitches. Glitches are not tightly controlled by design, but tolerated as harmless, being ignored in synchronous systems due to the fact that data is sought only at validation times on the edge of the clock. However, glitches are deterministic signals and can carry information. Glitches are still an untapped resource, since conventional logic design methods focused on masking them, and there are no synthesis methods that can control and take advantage of them. Glitch logic refers to a logic system that can detect, process, and transmit information embedded in glitch signals. Glitch logic differs from the main logic system and is not part of the main information flow in a digital signal processor. It may be however possible to superimpose the "glitch logic" on the "main logic" system without disturbing the main information flow. By generating a "glitch logic" superimposed/concurrent with the "main function" on the "level logic" we could create a concurrent information processing channel.

The conventional marking techniques used in logic systems hide the meaning of an information flow through "glitch logic" from unauthorized observers. The

monitoring, analysis, and interpretation is poorly supported by current tools, and glitch flow can be itself controlled by a stego-key that can turn on the glitch channel only when key is present. Also, a hazard-oriented circuit (producing glitches) is through its circuit design a mean to hide a key (manifested in the glitches it generates/processes) without having it stored in clear as is for example in a conventional memory. It is thus harder to probe for discovering the stored information compared to conventional techniques. The glitches can encode circuit-characteristic information, and makes this digital watermark/fingerprint of the circuit available for outside probing/testing, albeit in an obfuscated manner. All these characteristics make "glitch logic" an interesting candidate to a variety of applications ranging from protection to reverse engineering and alteration of design, to information infiltration/exfiltration into/from a digital processor.

The technology may also be used to create a "back door" to the system. For example, in case of digital integrated circuits designed/fabricated by un-trusted sources, or in the case of *any* FPGA that reconfigures during processing, glitch logic may already be built-in and offer a back door, a support platform on which malicious code is run, providing a vulnerability to the computing infrastructure.

Traditional digital technology uses circuits that operate with logic levels, most often stable voltage levels encoding a logic "*1*" with a high voltage level and logic "*0*" with a low voltage level. Thus, digital computing is basically computing with voltage levels and will be referred in the following as *level logic*. The mainstream paradigm of computing using logic levels proved successful and remained unchanged in decades, powered among other things by increased density of integration and switching speeds (between levels).

In digital circuits, concurrent signals may converge in logic gates after traveling paths with various delays, the resulting signal at the gate output may have short duration *glitches*. Good practice has recommendations for avoidance of these "*hazards,*" however there is no warranty that hazard-free logic implementation exists for an arbitrary Boolean function (Given an arbitrary Boolean function and specified set of (function hazard-free) input transitions, various techniques are developed to determine if any hazard-free multi-level logic implementation exists, e.g. [1]). Thus, superimposed with the conventional information flow, which is coded in binary logic levels, a stream of parasitic glitches usually travels along. In asynchronous circuits these glitches may be very harmful, however, in synchronous circuits (the large majority of today's systems) these are tolerated as harmless, as their effect is countered by the fact that a clock edge decides the moment when a logic level is attributed to a signal, and the timing of the clock edge is set such that by then the signal settled, i.e., the glitches (if any) have already appeared and died away.

The following example illustrates the same logic circuit designed in two classes of implementation: glitch-free level logic and glitch-logic. The circuit is a Binary to BCD converter when interpreted in terms of level logic. Various glitch logic implementations map other function as discussed later. Figure 1.a. presents the gate logic implementation with all the outputs Y4 to Y0 shown. Figure 1.b presents an alternative implementation, with glitches, for Y3 only.

Different circuit implementations of the same logic function have different glitches, also different path delays for same logic gate (e.g. implemented in different process/technology) lead to different glitches, as illustrated in Figure 2. The figure shows the response on output of Y3, for B3 – B0 decremented from '1111' to '0000 ', (a) shows a clock used to illustrate that glitches can be ignored in synchronous circuits, (b) shows B0 least significant bit (LSB) of binary to BCD converter, (c) shows Y3 for implementation of Figure 1.a, (d-f) shows Y3 for various values of t1-t9 gate delays: (d) with all gates with delay of 1 unit (say ns) (e) all delays are 1 , except t4=t5=t6 =2. (f) all delays are 1 except t4=t5=t6=t7=t8=2.

As Figure 2 shows, various circuit implementations for the same logic function, have glitches in different points. In a synchronous circuit only the value latched on the edge of the clock is meaningful. In this context, as illustrated by the figure above, all circuits provide the <u>same output logic level on the rising edge of the clock.</u> However, as the picture also shows, the instantaneous values of the signal in various circuits vary greatly.

The glitch-logic design of the described circuits was done by heuristics and trial and error. The development of precise methodology to design a glitch logic circuit to specifications is still an open problem.

III. Covert Glitch Channels and uses in Information Security

A. Covert/Hidden Channel and Glitch Logic Design

"Glitch logic" superimposed on the "level logic" creates a concurrent information processing channel. We could control this channel through proper design; we could extract the information in the channel by various means. For example, the simplest and most straightforward way to read would be sampling the output with a faster clock as illustrated in Figure 3. In this case the glitch logic level is detected and changes the information interpreted for that signal channel.

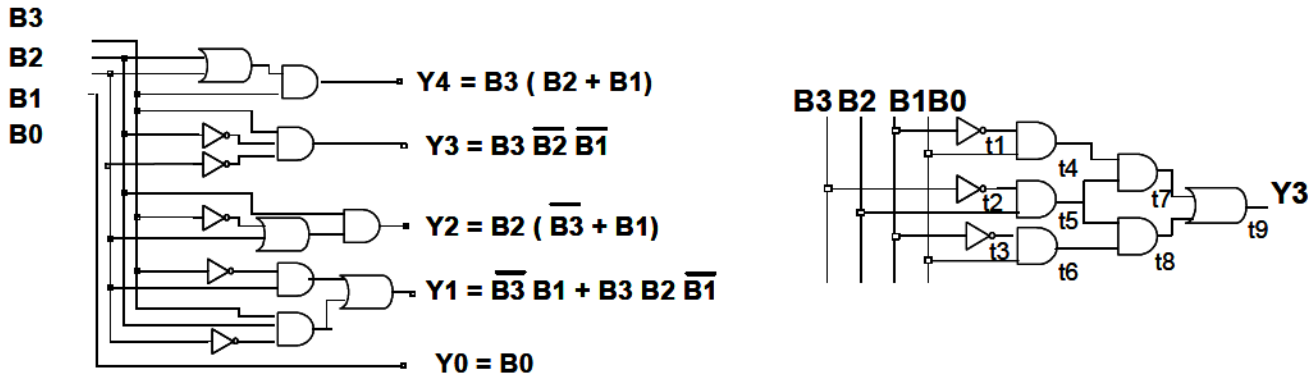


Figure 1. Binary to BCD converter: (a) left, a glitch-less implementation, where the clock edge was set to acquire the settled signal (b) right, one of many logic implementations with glitches for Y3. Different implementations have different glitches, which depend on delays t1 to t9.

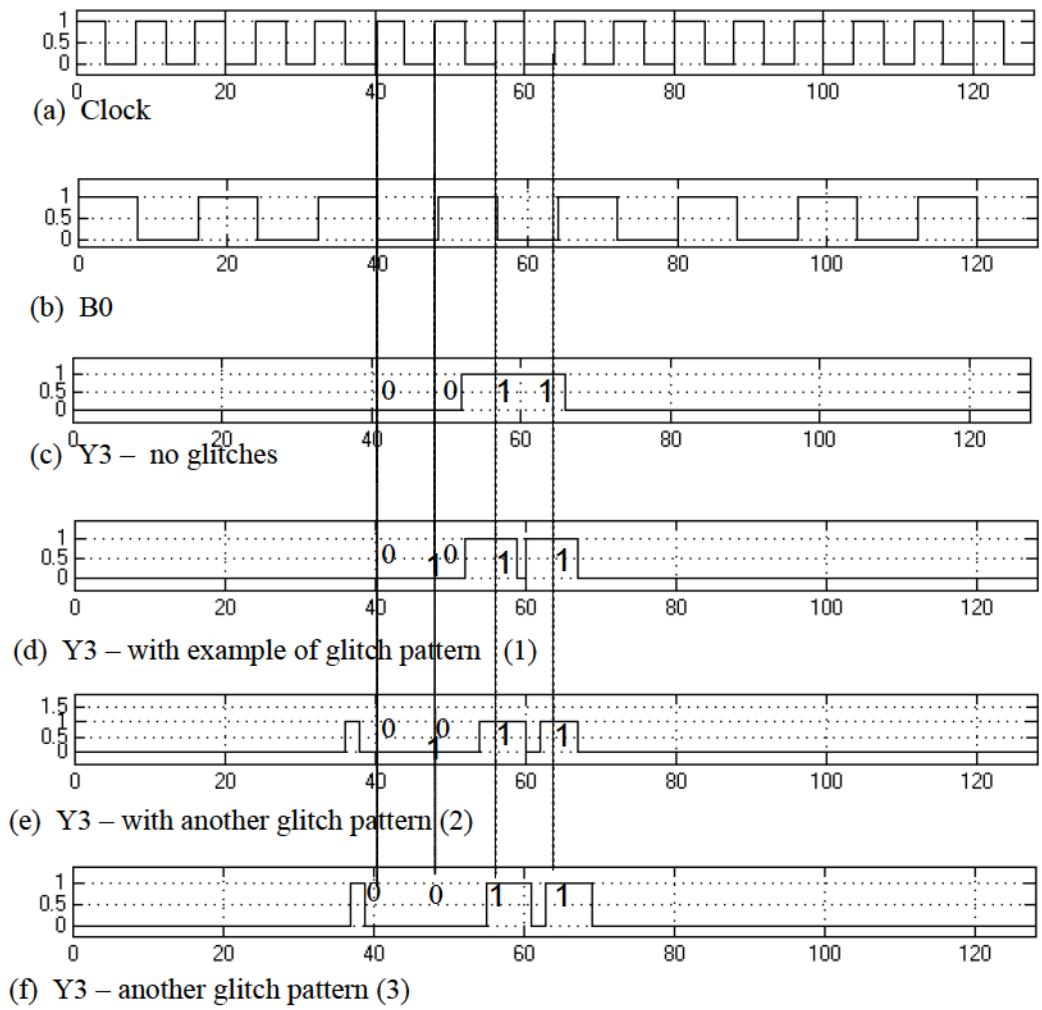
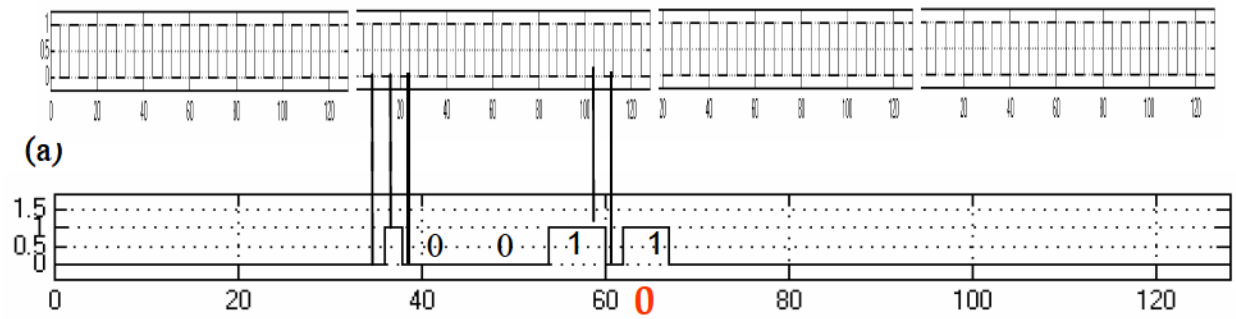


Figure 2: Output response for Y3, for B0 – B3 counting down from 1111 to 0000(b) B0 waveform, (c) Y3 waveform for the implementation of Figure 1.a, (d-f) Y3 waveforms for various values of t1-t9 gate delays: (d) with all gates with delay of 1, (e) all delays are 1, except t4=t5=t6 =2. (f) all delays are 1, except t4=t5=t6=t7=t8=2.



(b) **Figure 3.** Reading the glitch logic may simply be achieved by sampling with a faster clock that reads "interleaved" logic levels (here a '0' at time unit 60). (a) a 2ns clock, 4 times faster than in example in Figure 2 (b) output response for Y3 with a '0' at time unit 60.

Consider for example the glitch in Figure 3.b at ~ 60ns, where, for example, instead of reading a '1' one reads a '0' on BCD output Y3 line of Y3-Y0. Thus the sequence at 8, represented as '1 0 0 0', reads for a fraction of a second as '0 0 0 0', the glitch being equivalent to inserting another number in the series (for example, instead of the '8-8-8-8' sequence, this becomes '8-8-0-8'). To better illustrate a function on the glitch channel, we empirically designed by hand a modification of the binary to BCD encoder circuit, such that concurrent with the binary to BCD function on level-logic, an odd parity checker is implemented on the glitch-logic, on the bit set formed by two consecutive words on the bus (B2, B1). The odd parity function coming is coming on glitches on the Y4 line, and is triggered when B3 = '1'. This means that when B3 is '0' the circuit performs only the BCD conversion (without glitches on Y4) and when B3='1' the circuit is performing two functions: the binary to BCD conversion on level logic and the parity checker on glitch logic. The two functions, one on level logic and one on glitch logic operate concurrently, while also the glitch logic function can be enabled or disabled. The logic that enables it could be more complicated through more signals, equivalent to a key that starts the hidden computation.

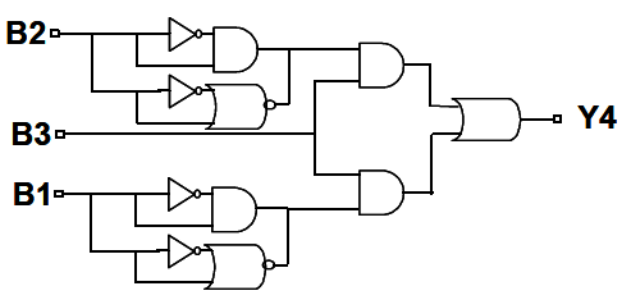


Figure 4. An alternative circuit implementation for Y4. It performs simultaneously 2 functions: MSB of the binary to BCD converter in "level logic": and odd parity check on successive word values on B2 and B1 lines (in glitch logic).

The circuit modification for Y4 is illustrated in Figure 4. The truth table of the parity function is illustrated in Table 1, where B2' and B1' are the inputs at time *t*, and B2 and B1 are the inputs at time *t+1*.

Table 1. Hidden channel function on glitch logic: a parity checker. The number of glitches encodes the function (one glitch for odd, zero or two for even). This function is concurrent with the Binary to BCD encoder, and is selectable by B3.

B2' (t) level	B1' (t) level	B2 (t+1) level	B1 (t+1) level	Y4 Glitch
0	0	0	0	0
0	0	0	1	⎍
0	0	1	0	⎍
0	0	1	1	⎍⎍
0	1	0	0	⎍
0	1	0	1	0
0	1	1	0	⎍⎍
0	1	1	1	⎍
1	0	0	0	⎍
1	0	0	1	⎍⎍
1	0	1	0	0
1	0	1	1	⎍
1	1	0	0	⎍⎍
1	1	0	1	⎍
1	1	1	0	⎍
1	1	1	1	0

For example, if current word on (B2, B1) is "00" and the new word is "10", a glitch will be produced on Y4 which indicates that the parity of the combined word (B2', B1', B2, B1) is odd. An external circuitry, not elaborated here, would decode the number of glitches, such that one glitch only is a logic one, else is a logic zero (for zero or two glitches).

New digital design methods are needed to exploit glitches, controlling their appearance, propagation, and combination (computing). Certainly, the designs should be in such a way that hazard-free design recommendations are intentionally infringed. This, unfortunately, is not a constructive solution, since many alternatives exist and choosing the ones that truly exploit glitches requires development of new design rules.

The logic of levels is the de-facto standard, and its capture, analysis by instruments and interpretation/reverse engineering by humans is easier; there are no current instruments/techniques to extract and analyze the significance of glitch logic propagation, which is another of open research.

B. Protection Against Reverse Engineering: Glitch Logic as Anti-Tamper (AT) Technique

The focus of conventional anti-tamper methods is deterring access to knowledge of the circuit topology or design. It is implicitly accepted that if a circuit design is known, then it can be reproduced, accessed, or its function denied. Practice demonstrates that any circuit design can be ultimately extracted, if sufficient resources are available. On the other hand, under the current economic model of outsourcing, with trusted foundries increasingly expensive, cheaper off-shore foundries may be used. These foundries receive the *actual circuit design.*

The danger of adversaries obtaining the knowledge from the design is very big, and conventional AT methods of preventing access to circuit design (and from there of function/algorithms) are of reduced use. *Technologies that offer protection of the function even if the circuit design is known would offer a great AT value.* We refer to this as cognitive anti-tamper methods. In particular, *glitches may carry the hidden information processing while the obvious digital function (logic levels) may be only a cover (possibly still performing useful computation, but not of sensitive information).* Thus glitch-logic design is a form of stego-design. A stego-key controls the correct glitch logic. In its absence other glitches and respectively other functions than the one we need to protect propagate through the system. Section 3.1 and example in Table 1 illustrated a case in which a 1-bit key inhibits the occurrence of the glitches. In practice the key would be multi-bit, and glitches can still propagate through, except their significance would not be the one to protect but to provide deceiving decoys.

Similar to other stego-designs, an attack of thus protected electronics would be greatly delayed because conventional analysis of these digital circuits would result in discovering a benign function and the circuit offers no hints of its access key or hidden function. Its discovery would require expensive simulations/stimulation to uncover subtle behaviors of the design. At first appearance, if a difference is observed from a "best implementation" it may appear as a "poor" (yet logically correct) design. Assuming attacker/adversary knows of the existence of a hidden design, their effort needed for the identification of the new hardware design and protection techniques will delay reverse engineering and exploitation, slowing him in compromising exposed critical technologies.

When simulated in absence of the stego-key, the circuit would not propagate the glitches, or would propagate different glitches than the ones obtained when key is used, and the function displayed by the circuit would be the benign level logic function. A reverse engineering attack of such circuits would imply simulations/stimulations with many combinations of input voltage signal levels and times/delay targeted to finding circuit functionality, but unless the secret stego-key is applied, functional response will only be cover functions. The search space on the combination of various timing and delays appears too large for brute force methods.

C. Protection Against Design Alteration

Glitch logic can act as a digital watermark/fingerprint, which can be probed externally and yet it is obfuscated. Each circuit can be designed to have a different "signature" (in terms of glitches). Thus, we can "prepare" the signature as intended to encode ownership information, but most importantly an 'IC-metric' (by analogy with bio-metric) characteristic of the circuit itself. This could be various check sums, information on the number of gates of a certain type, etc.

C.1 Hiding a Secret Key in the Circuit Design

A hazard-oriented circuit embeds, through its circuit design, a mean to hide a key (manifested in the glitches it generates/processes) without having it stored in clear, thus harder to probe for discovering the stored information compared to conventional techniques using memories.

C.2 Channel Interferences: Vulnerabilities, Taking Control, Information Infiltration/Exfiltration, Viruses, Spyware and Trojan Horses in Hardware

Glitch logic may offer an avenue of tapping into a processor, a back door, a platform on which runs malicious code, etc. Such vulnerability may already be present, including in 'secure' computing infrastructures. One could potentially inflict such channels at will to have a secret/covert access/control.

The covert channel expressed in glitch logic operates with conventional gates, requiring only new synthesis tools. It can be readily used on FPGA devices or on future ASIC.

In particular in FPGAs that continuously reconfigure at run-time, the software may decide to insert delay paths

that create specific glitches, while maintaining the correct functional level-logic. Then, at certain moments, again only under software control and without infringing the functional logic, a large delay (by artificially creating a long cascaded path for signal propagation) may actually create an interference/collision between glitch logic and level logic. In this case, one can exchange information between the two channels, either infiltrating wrong data, taking over data into the glitch channel, inducing a desired or an arbitrary error into the level-logic functionality. Practically this scenario uses the glitch channel either as a virus, spy-ware, Trojan horse etc, infiltrated into hardware by the software (reconfiguration software on FPGA or synthesis software in ASIC). Without an understanding of glitch technology it would hard to recognize the threat, even if already present in our digital systems.

IV. Systems Summary of Characteristics

Applications of glitch logic computing can be considered in two categories:

A. Active measures

- Usage as a covert computing or communication channel
- Preventing reverse engineering, tempering and alteration of a proprietary design
- Usage as a channel for information infiltration/exfiltration and propagation of viruses/spyware/Trojan horses

B. Passive measures

- Countermeasures against the use of glitch computing
- Mechanisms for detecting the presence of a glitch logic layer superimposed on a level logic layer
- Mechanisms for decoding glitch logic circuitry
- "Translation" of glitch logic functions into level logic functions

V. Conclusions

This paper proposed glitch logic, a new method of information processing, with potential impacts on digital computing and information security. We have shown that glitch logic can act as a hidden information flow channel in digital circuits. It can have applications as a covert computing/communication channel, as protection from reverse engineering/anti-tamper, as protection from alteration of design, in computer attacks/security, such as providing a channel for information infiltration/exfiltration and propagation of viruses/spyware/Trojan horses.

VI. Acknowledgment

This research was carried out at the Jet Propulsion Laboratory (JPL), California Institute of Technology, under a contract with the National Aeronautics and Space Administration. It was funded by a JPL Spontaneous Concepts Award part of Director's Research and Technology Development Fund.

VII. References

[1] Steven M. Nowick, Charles W. O 'Donnell, On the Existence of Hazard-Free Multi-Level Logic Ninth IEEE International Symposium on Asynchronous Circuits and Systems, (ASYNC'03) p. 109.

[2] A. Stoica, JPL New Technology Report, NTR # 44492, Method of using glitches for information hiding, communications, and processing. 2007