# Fuzzy-Glitch: A Practical Ring Oscillator Based Clock Glitch Attack

Johannes Obermaier, Robert Specht
Fraunhofer Institute AISEC
Garching b. München, Germany
{johannes.obermaier, robert.specht}@aisec.fraunhofer.de

Georg Sigl
Institute for Security in Information Technology
Technical University of Munich, Germany
sigl@tum.de

***Abstract*—Clock glitches are useful in hardware security applications, where systems are tested for vulnerabilities emerging from fault attacks. Usually a precisely timed and controlled glitch signal is employed. However, this requires complex generators and deep knowledge about the system under attack. Therefore we present a novel approach on clock glitch fault attacks that replaces the single precise glitch by a fuzzy glitch signal. We propose a compact FPGA design for fuzzy clock glitch generation, that is based on mixing two adjustable ring oscillators of different frequencies. The combination of these oscillators creates a glitch containing random and high frequency signal components. We show on the basis of a practical implementation on a Spartan-3E, that the proposed method is able to generate the desired fuzzy clock glitch. We verified experimentally, that the fuzzy clock glitch succeeds in error injection on an STM32F030, an ARM CORTEX-M0 based microcontroller. Our results demonstrate that the fuzzy glitch is an adequate solution for fault injection.**

## I. Introduction

Embedded systems are employed in many applications, ranging from industrial systems over automotive control units up to end-user devices. They contain intellectual property, such as algorithms and code, may be license-locked, and often include cryptographic algorithms and secret key material. As these systems become more powerful and hold more valuable information, it also becomes more worthwhile for a malicious attacker to gain access to them. Thus such a system will be subject to various attacks. In order to establish protection and to estimate the probability of successful attacks, tools must be developed to execute and understand those attacks.

There are various groups of hardware attacks, one of them are fault attacks which aim at causing erroneous operation of the processor and provoke leakage of critical information [1]. Clock glitches are an easy and effective technique to induce faults into a system. In general, a glitch can be described as a short and normally unwanted change of the signal's logic level. This is depicted in Fig. 1 on the upper plot. In case of an attack, this short clock signal is induced intentionally, such that the device is operated at frequencies out of specification. This will lead to timing violations of registers and results in undefined behavior [2]. The effects can be divided into two groups, control flow errors or data changes. The results in diversion of control flow are, e.g. early exit from loops or crashes [3], [4] . Data changes are especially critical for cryptographic algorithms, like AES, because they lead to very powerful attacks [5]. Hence, securing devices against fault attacks is very important.

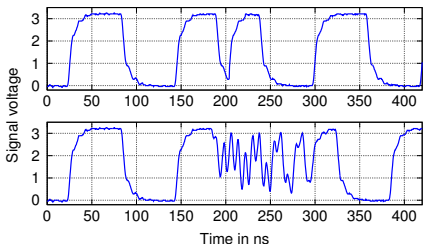


Fig. 1. Drawing of a usual (top) and a fuzzy (bottom) clock glitch

### A. Related Work

Numerous implementations of clock glitch attacks have been published that follow the same baseline for signal generation. In all cases known to the authors, the system generates a precisely timed glitch signal [6]–[9]. The method generates reproducible results, has proven to be useful for an in-depth analysis, and provides high success ratios [7], [9]. Thus there exist powerful methods to precisely attack well-known systems and algorithms by targeting down to a single instruction [4].

The method incorporates also some drawbacks. It works best if the system under attack is well-known and was analyzed beforehand—a precondition that cannot be fulfilled in all cases. Additionally, the glitch generation circuitry is built up around FPGA-specific hardware like Digital Clock Managers (DCM) [6]. Their number is limited and their capabilities depend on the specific device and vendor, which may be a limitation for small and simple devices. Moreover, the generated glitch is still a fairly deterministic digital signal without much randomness, thus a variation of the attack parameters, e.g., glitch frequency, rise, and fall time, has to be brought in externally. In most cases the evaluation of all possible parameter combinations is not feasible in reasonable time [10].

### B. Contributions

To overcome these issues, we propose an alternative concept on the generation of clock signal glitches that breaks with the commonly used approach to inject a precisely generated glitch. Instead, we present a novel method that generates a random, nondeterministic and fuzzy glitch signal, as depicted in Fig. 1.

ISBN 978-80-261-0642-5, 

Our approach aims at embedded systems, hardware as well as software, that have not been analyzed in detail yet, but shall be tested for susceptibility against glitch attacks. Thus we trade in attack reproducibility for the ability to reduce the parameter space and allow a simpler and quicker evaluation of an attack. Additionally, the requirements on the FPGA are lower.

Our presented contributions include:

- A novel concept to create a clock glitch signal by combining two adjustable ring oscillators of different frequency using a XOR gate.
- The use of a "fuzzy" random and nondeterministic glitch signal instead of a exactly timed glitch.
- A practical implementation of the method on a Spartan-3E FPGA platform using only standard slice logic.
- A demonstration of a successful fuzzy clock glitch attack on the ARM Cortex-M0 based microcontroller STM32F030.

### C. Structure

The paper starts with a general overview in Section I. The explanation of the technical approach and implementation follows in Section II. The paper continues with an evaluation of the implementation in Section III. The previously implemented system is utilized to carry out an exemplary attack on a microcontroller in Section IV. Finally, we sum up our results in Section V.

## II. Technical Approach

Our design aims at achieving a random, nondeterministic and fuzzy clock glitch signal. This requires three components: A frequency source for signal generation, a source of entropy for randomness, and means to generate a glitch. Common oscillators target high stability and signal quality, thus they are no viable source for random glitches. Instead, we resort to using ring oscillators (ROs) [11].

ROs are frequency generators that are implemented by simply interconnecting an odd number of inverters in a circular manner as shown in Fig. 2. This creates an unstable combinatorial loop that leads to oscillation. ROs are implemented using standard slice logic, thus these oscillators do not need DCMs or other special FPGA-specific blocks. Their frequency depends on the length of the inverter chain, because each inverter adds additional delay and consequently increases the signal period.

ROs have another unique feature, they are not only frequency generators but they also provide randomness to the system, thus they fulfill a dual purpose. The frequency and phase of the output signal is subjected to jitter [12]—a property that is also used in RO-based true random number generators [13]. This contributes

Fig. 2. A simplified ring oscillator with an odd number of inverters

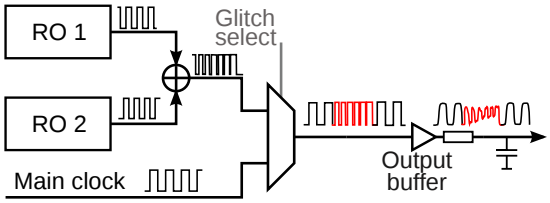


Fig. 3. Glitch insertion circuitry, output buffer, and RC load

an intrinsic variation of the glitch, thus the attack parameters are inherently altered over time.

The third required component for the system must provide means to generate a glitch, as a single RO does not create such a signal by itself. If two ROs of different frequencies $f_1$ and $f_2$ are combined using a XOR gate, the resulting signal is toggled on every edge of each RO, hence creating numerous glitches. This corresponds to mixing the frequency of both ROs, resulting in $|f_1 - f_2|$ and $|f_1 + f_2|$, a signal containing various high and low frequency spectral components. Harmonics contribute additional frequency components. The output impedance and load capacitance limits the bandwidth of the FPGA output and the resulting signal is distorted.

The glitch is injected only for a limited amount of time, since the system is able to switch between the unmodified clock signal and the glitch. This is facilitated by a multiplexer at the system's output.

The glitch generation and injection logic is depicted in Fig. 3. The glitch signal, that is generated by applying the XOR operation on two RO outputs, can be selected using the Glitch-select control input. Otherwise the system outputs an unmodified clock signal.

### A. Practical Implementation

The proposed system was implemented on a Nexys 2 board which is a Spartan-3E FPGA platform. The basic structure, depicted in Fig. 3, was implemented straightforward.

In contrast to this, the ROs require a reconfigurable design. Their behavior is difficult to estimate, since it is a free running oscillator that is not coupled to any clock. Its frequency depends on the speed grade of the device, the placement and routing of the inverters, and local internal variations of the FPGA [14]. Therefore it was necessary to develop a solution that allows more control over the RO frequency.

The oscillation period of an RO depends on the length of its combinatorial loop. Each inverter consumes time during switching and the routing generates an additional delay. Thus an increase in the number of inverters raises the loop length and consequently the oscillation period, thereby lowering the frequency. Hence the oscillation frequency can be controlled by scaling the length of the inverter chain.

The scalable RO module is depicted in Fig. 4. It consists of sub-modules, each containing two inverters and the corresponding demultiplexer and multiplexer. The number of additional inverters inserted by each sub-module is even, since this will preserve the odd parity

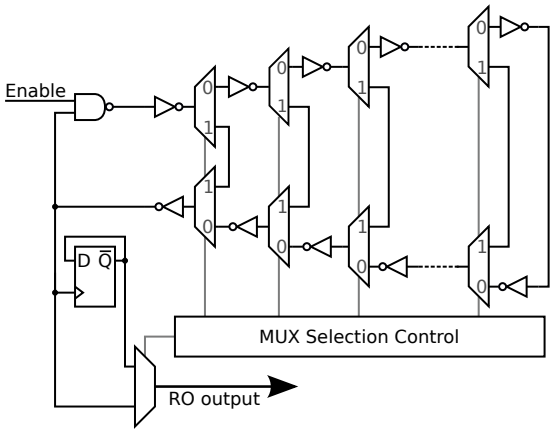


Fig. 4. The adjustable RO with its demultiplexers (top) and multiplexers (bottom), which allow adjusting the length of the inverter chain and therefore controlling the oscillation frequency.

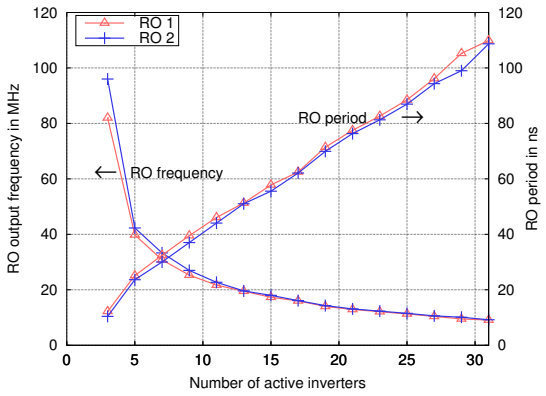


Fig. 5. RO output frequency and period vs. the inverter chain length

of the number of inverters. The MUX Selection Control block implements a one-hot encoding, that shortens the inverter chain by bypassing inactive inverters. Although the demultiplexers are optional, they shut down unused inverters, so no dangling logic exists.

The Enable input of the NAND gate allows to start and stop the oscillation of the RO. If the Enable signal is `0`, the NAND gate's output is constantly `1`, independent from the other input, thus preventing the RO to oscillate. When the Enable signal becomes `1`, the NAND gate becomes sensitive to the other input and implements an inverter. This additional inverter causes the number of inverters to become odd. That leads to instability of the loop and hence causes oscillation.

The toggle flipflop at the output is optional. It can be selected to further divide the output frequency by two. The frequency can be halved by doubling the length of the inverter chain or by adding the register. While the first method requires excessive resources, the register comes at low down to no additional slice usage.

This flexible RO implementation was developed, to allow a reconfiguration of the inverter chain while the system is operating. It enables quick adjust of parameters without having to execute the cumbersome synthesis process repeatedly.

The glitch circuitry is configured using a UART interface of the FPGA. The length of each RO can be set independently in steps of two between 3 and 255. The use of the output register can also be chosen independently from each other.

In the glitch insertion module, the duration of the clock glitch is configured in steps of 10 ns between 10 ns and up to several microseconds. The duration is defined as the time span, during which the original clock signal is replaced by the fuzzy glitch.

## III. Experimental Results

In this section the previous claims are verified. At first, the performance of the scalable RO is measured in terms of its frequency output and adjustability. Secondly, the overall system is evaluated and the generated glitch is assessed.

In order to analyze the two scalable ROs, named RO 1 and RO 2, the frequency of RO 1 is measured while RO 2 remains disabled. Afterwards the experiment is repeated vice versa. The result is expected to show a linear mapping of the inverter chain length onto the period of the output signal.

The results for both ROs are depicted in Fig. 5. The period rises approximately linearly with the number of active inverters in the chain. The period of the shortest configuration, an RO of three inverters, is slightly lower than expected. This minor deviation is attributed to the irregular structure of the first group of inverters, which has less multiplexers/demultiplexers per inverter, compared to the remaining circuit. Altogether, the expectation matches the measurement and is in accordance to the design. The design shows good adjustability, as increasing the length by two inverters will raise the period by 6.5 ns in average.

An interesting detail is the slightly increased frequency of RO 2 compared to RO 1. It is common for RO structures to exhibit frequency variation, despite they were implemented using the same VHDL design [14]. Such a characteristic is beneficial for fuzzy glitch generation, since the mixing of exactly matching signals would not create a suitable glitch signal.

After the verification of the scalable ROs, the fuzzy clock glitch signal is assessed. Therefore the ROs are configured to a length of three and five inverters, corresponding to 82 MHz and 42 MHz, respectively. The glitch duration is set to 100 ns. The original clock frequency is 8 MHz by default. An oscilloscope is attached to the FPGA output, the glitch is triggered, and recorded.

Fig. 6 shows the resulting signal. As long as the glitch signal is not active, the original 8 MHz clock remains

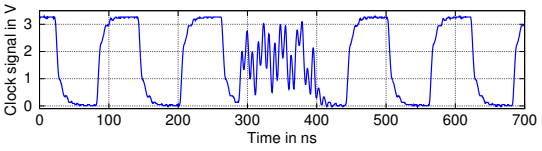


Fig. 6. A fuzzy glitch with a duration of 100 ns

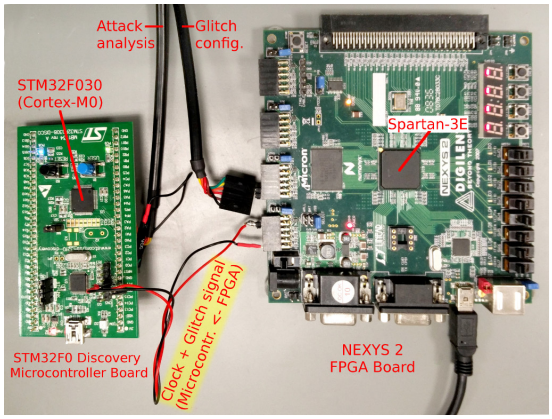


Fig. 7. Attack setup for the validation of the fuzzy glitch method

unaltered. Upon the trigger event at about 280 ns, the clock is replaced by the fuzzy glitch signal for about 100 ns. As expected, the signal is highly distorted, since the FPGA board cannot drive this high frequency signal sufficiently well. The signal has no regular shape, does not fully span between 0 V and 3.3 V, and has mainly analogue characteristics. Upon repeating the experiment, the glitch will be different, since the internal state of each RO diverges over time.

## IV. Exemplary Attack on Cortex-M0

The effectiveness of the implemented clock glitch setup was proven in practice. We set up a sample embedded system using a microcontroller. Subsequently the system's reaction on clock glitches was investigated, while the attack parameters were varied.

### A. Experimental Setup

The experimental setup is shown in Fig. 7. The Spartan-3E FPGA board, which generates the clock glitch signal, is on the right-hand side. The glitch parameters are configured using the FPGA's UART interface. The clock signal and glitch is transmitted to the system under attack, using the red/black twisted pair of wires in the center of the image.

The STM32F0 Discovery evaluation board, placed on the left-hand side, resembles a simple embedded system that is under attack. It contains an STM32F030 microcontroller that incorporates an ARM Cortex-M0 CPU. The effects of the glitch attack on the microcontroller can be examined using the microcontroller's UART interface (UART-M).

The microcontroller is configured to use an external 8 MHz clock source, controlled by the FPGA. The PLL is not enabled, thus the external clock drives the Cortex-M0 CPU directly.

### B. The Code under Attack

Usually, an embedded system performs several tasks in quick succession, but this impedes a thorough analysis of the glitch effects. The external visibility of errors, caused by the glitch, highly depends on the currently executed instructions and tasks. As a consequence, results would be hard to interpret and will be impossible to reproduce reliably.

Therefore a special implementation was tailored in order to overcome these issues. The code contains selected instructions, that are comprised in an infinite loop. The infinite-loop ensures, that the processor is executing the code of interest whenever a glitch arrives. The body of the loop includes instructions for computations, comparisons and branches—basic building blocks of common control flow statements like loops and conditional expressions. The firmware was implemented in such a way that a severe erroneous execution of any instruction of the loop will have a recognizable effect. This part of the firmware was manually implemented using assembly language [15] in order to maintain full control over the generated instructions. The corresponding ARM thumb assembly code is shown in Listing 1.

Listing 1. Excerpt from the microcontroller code under attack

```
movs r0, #0x01  ;   r0 = 0x01
mov r10, r0     ;  r10 = 0x01
loop1:
        movs r5, #0x00  ;  r5 =  0x00
        add r5, r10     ;  r5 += 0x01
        add r5, r10     ;  r5 += 0x01
        add r5, r10     ;  ...
        add r5, r10
        add r5, r10
        add r5, r10
        add r5, r10
        add r5, r10
        add r5, r10     ;  ...
        add r5, r10     ;  r5 += 0x01
        cmp r5, #0x0A   ;  r5 == 0x0A?
beq loop1  ;  if true, goto loop1
```

At first, the registers r0 and r10 are set to `0x01`. Inside the loop, r5 is reset to `0x00` in order to have a well-defined initial state. The following ten instructions are identical. Each one increments the register r5 by r10, thus r5 is incremented by `0x01`. After these ten instructions were executed, r5 obviously equals ten. This is verified in the next step, where r5 is compared to `0x0A`. If both values are equal, the branch-if-equal (beq) instruction will jump back to the beginning of the loop, to the label "loop1". This represents an infinite loop, since r5 will always equal ten after it has been incremented ten times.

The loop has a very high transparency for incorrect instruction execution due to glitches. For example, if the register r5 is not correctly reset to `0x00` or if one incrementation is missed or executed twice, the final value will not equal `0x0A`. The subsequent compare will fail, the branch will not be executed and thus the control flow is diverted out of the infinite loop. Similar effects may happen, if the compare itself fails or if the branch is mislead.

### C. Firmware Behavior and Design

The microcontroller requires an extended firmware, that allows a deep analysis of the glitch effects on the infinite loop. The implemented firmware is depicted in Fig. 8.

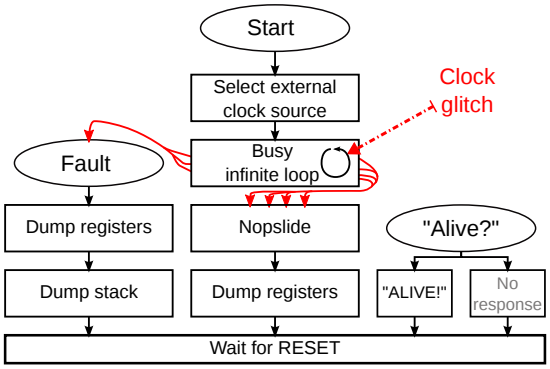


Fig. 8. The microcontroller firmware under attack including possible glitch-induced faults

After the first few practical tests, the different possible outcomes of the glitch became visible. They can be grouped into four categories:

1) **Success:** The infinite loop is left and code following the loop is executed.
2) **Hardfault:** A fault occurs and the system jumps into the corresponding fault handler.
3) **No effect:** No visible effect on the system, the system continues execution unaffectedly.
4) **Crash:** The system crashes and the CPU stops the execution of code.

In the **success** case, the system leaves the loop and drops into the nopslide that is placed directly after the infinite loop. A nopslide is a large region of code containing tens of subsequent NOP (no operation) instructions. Such a structure is necessary to correctly catch the perturbed control flow, even if multiple instructions are skipped on exiting the loop. After the nopslide, the microcontroller dumps its registers using the microcontroller's UART interface.

In the **hardfault** case, the glitch has caused the system to jump into the fault handler or an interrupt handler. These cases are caught by employing a default-handler, redirecting each such event to the dumping function. Upon a fault or interrupt, this function is called and creates a register dump. Additionally, a stack dump is extracted, showing the triggering instruction and gives additional information.

In the success and hardfault case, the microcontroller automatically sends a status dump upon exiting the infinite loop. If no status message was sent by the microcontroller, it was either unaffected by the glitch and is still running correctly (no effect) or it has crashed and ceased operation (crash). Both cases can be identified by sending an "Alive?" query packet to the microcontroller and looking for a reply.

In the **no effect** case, the microcontroller was unaffected by the glitch and continues operation. Thus it will respond to the "Alive?" packet and prove, that it is still fully functional.

In the **crash** case, where the system has ceased operation, no response will be generated for an "Alive?" packet. This shows, that the system is in the "lockup" state, where the processor has fully stopped code execution and requires a hard reset.

### D. Evaluation and Results

The clock glitch attack was executed on the embedded system and the effects were analyzed. The attack was configured to use ROs with a length of three and five inverters, the same setting as shown in Section III. The duration of glitch insertion was varied, starting with 20 ns. It was then increased in steps of 20 ns up to a maximum time of 800 ns, totaling in 40 different settings for duration. For each of the 40 settings, we repeated the glitch attack 800 times and analyzed the result after each repetition. One glitch attack corresponds to replacing the 8 MHz clock with the clock glitch signal for the specified duration.

The register and stack dumps show very different effects of the clock glitch. In most cases, the register r5 was affected, as the loop primarily operates on this register. It occurs, that the incrementation of r5 fails and the register is set to a number less than ten, e.g., `0x06`. Thus some instructions were skipped or the incrementation failed. Also the opposite was observed, where r5 was over-increased, e.g. to `0x14` or `0x0C`. This may have been caused by a skipped reset of r5, by multiple execution of a single instruction, or by miscalculation. In some cases, the r5 register contents were inconclusive, e.g., `0x08000606` or `0x00800000`. This is a hint on a major malfunction of the CPU that destroyed the register contents.

There were cases, where r5 equaled ten, but nevertheless, the system left the loop. An analysis of the status register shows, that the compare instruction erroneously reported r5 being different from ten. This incorrect result is given to the following branch-if-equal instruction, the branch is not taken, and the loop is left.

The branch instruction has also failed in some occasions. Despite r5 equals ten and the status register shows, that the compare of r5 with ten returned a positive result, the branch is not taken.

The nopslide after the infinite loop has proven to be useful. The code was instrumented and it became visible that the glitch sometimes also causes the first few instruction of the nopslide to be executed incorrectly. Nevertheless, the analysis of these cases was still successful, since an error in nopslide execution does not cause any effect.

In very few cases, around 10 % in average, the glitch causes a hard fault. The occurrence of any other interrupts was not observed. The stack and especially the return pointer was examined in this case. It became evident, that there is no certain instruction that causes the hard fault during a glitch. Instead, it happens randomly. The reason for the hard fault is unknown but can be speculated. One possibility may be an access to an invalid memory address, caused by a corruption of the program counter which holds the currently executed memory address. Another explanation is an error during instruction fetch, where an invalid instruction is given to the processor.

The glitch can also cause a crash of the system. This was observed in less then 5 % up to 45 % in all cases, highly depending on the glitch duration. A lockup

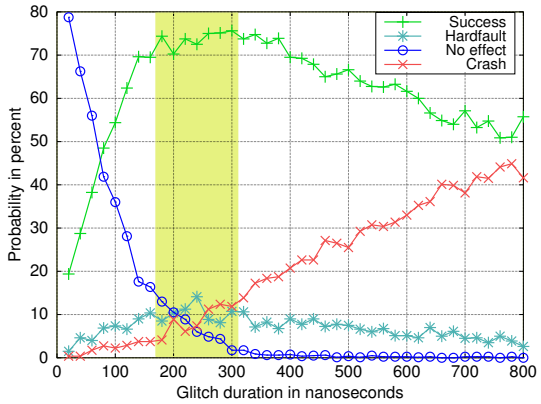


Fig. 9. Glitch-caused effects for different glitch durations

occurs, if the system is overstressed by the glitch, such that it ceases operation. Too many data was destroyed, what leads to an unrecoverable malfunction.

### E. Statistical Evaluation

A plot of glitch effectiveness vs. length is depicted in Fig. 9. The data is based on 800 measurements per glitch duration.

The plot shows, that for very short glitch durations, there is **no effect** on the system in about 80 %. Upon increasing the glitch duration the ratio of unsuccessful attempts begins to decline quickly. At 200 ns duration, the glitch causes an effect in 90 % of the cases.

As the glitch duration approaches 180 ns, the probability for **success** rises quickly. The optimal region for glitch duration spans from 180 ns up to 310 ns, where success is achieved in about 75 % of the attacks while maintaining little crash probability. This region of optimal glitch effectiveness is highlighted in Fig. 9. If the glitch duration is further increased, the success ratio begins to decline again.

The probability for a system **crash** increases approximately linearly from 12 % at 300 ns to up to 45 % at around 800 ns. This increase takes place on cost of the success ratio, thus operating in this region is not advised. The crashes are attributed to an increased number of multiple errors due to the longer glitch, which effectively leads to a crash of the system.

The probability for a **hard fault** stays below 15 % in most of the cases and has minor relevance for the attack.

The plot shows, that despite the glitch is generated by a random process, its effect on the microcontroller is still under control and can be modified by varying the glitch duration. The results of the measurement are conclusive and match the expectation. Neither very short nor excessively long fuzzy glitch durations are optimal. The region of optimal glitch effectiveness spans between 180 ns up to 310 ns.

## V. Conclusion

In this work we presented the novel class of fuzzy clock signal glitches. They are created by mixing the signals of two free-running adjustable ring oscillators which contribute randomness to the attack. We demonstrated successfully, that the proposed design can be implemented on an FPGA in practice. An experiment was set up, where an STM32F030, an ARM Cortex-M0 based microcontroller, was successfully attacked using fuzzy clock glitches generated by our design. The attack's effects were analyzed in detail by employing a custom mircontroller firmware. The results show, that our system reaches up to 75 % glitch effectiveness in the optimal case and remains over 50 % over a wide range of parameters, depending on the glitch duration. All in all, these results prove, that this alternative approach on clock glitch attacks is feasible and valid.

## Acknowledgment

The authors would like to thank Robert Hesselbarth for providing the RO VHDL primitive and Qiyi Li for supporting the practical implementation.

## References

[1] M. Tunstall, D. Mukhopadhyay, and S. Ali, "Differential fault analysis of the advanced encryption standard using a single fault," in *IFIP International Workshop on Information Security Theory and Practices*. Springer, 2011, pp. 224–233.

[2] M. Agoyan, J.-M. Dutertre, D. Naccache, B. Robisson, and A. Tria, "When clocks fail: On critical paths and clock faults," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2010, pp. 182–193.

[3] H. Choukri and M. Tunstall, "Round reduction using faults," *FDTC*, vol. 5, pp. 13–24, 2005.

[4] J. Balasch, B. Gierlichs, and I. Verbauwhede, "An in-depth and black-box characterization of the effects of clock glitches on 8-bit mcus," in *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2011 Workshop on*. IEEE, 2011, pp. 105–114.

[5] D. Saha, D. Mukhopadhyay, and D. Roychowdhury, "A diagonal fault attack on the advanced encryption standard."

[6] M. Matsubayashi, A. Satoh, and J. Ishii, "Clock glitch generator on sakura-g for fault injection attack against a cryptographic circuit," in *Consumer Electronics, 2016 IEEE 5th Global Conference on*. IEEE, 2016, pp. 1–4.

[7] T. Korak and M. Hoefler, "On the effects of clock and power supply tampering on two microcontroller platforms," in *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2014 Workshop on*. IEEE, 2014, pp. 8–17.

[8] T. Korak, M. Hutter, B. Ege, and L. Batina, "Clock glitch attacks in the presence of heating," in *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2014 Workshop on*. IEEE, 2014, pp. 104–114.

[9] S. Endo, T. Sugawara, N. Homma, T. Aoki, and A. Satoh, "An on-chip glitchy-clock generator for testing fault injection attacks," *Journal of Cryptographic Engineering*, vol. 1, no. 4, p. 265, 2011. [Online]. Available: http://dx.doi.org/10.1007/s13389-011-0022-y

[10] S. Picek, L. Batina, P. Buzing, and D. Jakobovic, *Fault Injection with a New Flavor: Memetic Algorithms Make a Difference*. Cham: Springer International Publishing, 2015, pp. 159–173. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-21476-4_11

[11] E. Eilley, "Ring oscillator," Jan. 2 1990, uS Patent 4,891,609. [Online]. Available: https://www.google.com/patents/US4891609

[12] A. A. Abidi, "Phase noise and jitter in cmos ring oscillators," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 8, pp. 1803–1816, 2006.

[13] V. Fischer, F. Bernard, N. Bochard, and M. Varchola, "Enhancing security of ring oscillator-based trng implemented in fpga," in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*. IEEE, 2008, pp. 245–250.

[14] A. Maiti, J. Casarona, L. McHale, and P. Schaumont, "A large scale characterization of ro-puf," in *Hardware-Oriented Security and Trust (HOST), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 94–99.

[15] STMicroelectronics, "Stm32f0xxx cortex-m0 programming manual," Apr. 4 2012. [Online]. Available: https://www.st.com/resource/en/programming_manual/dm00051352.pdf