Fault Injection using Crowbars on Embedded Systems

Colin O'Flynn

Abstract—Causing a device to incorrectly execute an instruction or store faulty data is well-known strategy for attacking cryptographic implementations on embedded systems. One technique to generate such faults is to manipulate the supply voltage of the device. This paper introduces a novel technique to introduce those supply voltage manipulations onto existing digital systems, requiring minimal modifications to the device being attacked. This uses a *crowbar* to short the power supply for controlled periods of time. High-accuracy faults are demonstrated on the 8-bit AVR microcontroller, which can generate both single and multi-bit faults with high repeatability. Additionally this technique is demonstrated on a FPGA where it is capable of generating faults in both internal registers and the configuration fabric.

1 Introduction

Faults cause a computer program to behave in an unintended manner. For many systems this could have dire consequences, and protecting systems from such faults is an important area of research. Fault injection encompasses the techniques which are used to purposely cause faults to occur, for example as part of validating or testing a fault-tolerant or fault-detecting scheme [1]. Fault injection is also useful as a testing tool when designing for environments likely to cause single-bit failures, such as space applications or high-radiation environments [2].

Fault injection is also a powerful tool to break cryptographic algorithms. The previous example assumed the 'dire consequences' of a fault occurring were a result of the system performing an unexpected action. But a fault could be purposely injected to cause a system to behave abnormally, to an attackers advantage. It has been well known that a variety of fault injection methods can be used for this purpose [3], [4].

Previous work on fault injection has demonstrated methods of breaking cryptographic algorithms such as DES [5], AES [6], [7], [8], and RSA [9], [10], [11] by introduction of faults at specific parts of the algorithm. Of these, a practical demonstration of the proposed method is also given in [7], [8], [10], [11]. All of these demonstrations are performed on a custom board, specifically designed to inject faults into the embedded computer running the cryptographic algorithm. The reader is referred to [12] for a more detailed survey of attacks on AES and RSA.

Having a practical method of injecting faults into an embedded computer is of great importance to both these areas of research: understanding the vulnerability of systems to fault injection attacks, and validating design of fault-tolerant computing systems.

- *C. O'Flynn is with the Department of Electrical and Computer Engineering, Dalhousie University, Halifax, NS, Canada.*
  Z. Chen is currently with the School of Electronic Engineering, The University of Electronic Science and Technology of China, on leave from Dalhousie University, Halifax, Nova Scotia, Canada.
  E-mail: {z.chen,coflynn}@dal.ca

1.1 Our Contribution

Our work focuses on the practical aspect of injecting faults into a commercial off-the-shelf (COTS) embedded computer. Previous work has demonstrated the use of clock glitching or EM glitching on COTS embedded computers, such as attacking the Beaglebone Black using EM glitching as demonstrated in [13]. Our work instead uses power supply glitching to insert faults in COTS embedded computers. Clock glitching will not work on more complex devices (discussed in section 2.1), and EM glitching is very sensitive to setup and equipment (discussed in section 2.2). We introduce a novel method of reliably introducing faults using power-supply glitching, applicable to a wide range of platforms and devices.

The novel method of generating power supply glitches uses a *crowbar* circuit, which aggressively shorts the power supply of the device to generate faults. This introduces ringing in the power distribution network on the circuit board, which propagates into the on-chip power distribution network. Ringing in this on-chip network is known to cause faults in digital devices, as shown in [14].

It will be demonstrated that a power supply glitch can be used to glitch a specific instruction. Previously it was considered that clock glitching could achieve much better temporal accuracy than power supply glitching [12], but we will demonstrate that it is possible to achieve high temporal accuracy with power supply glitching on embedded systems.

This fault insertion is first characterized on a custom board using an AVR 8-bit microcontroller, then demonstrated on several COTS embedded computer boards: a Raspberry Pi running Linux, a Beaglebone Black running Linux, and an Android smart phone. The applicability of this method to fault injection against Field Programmable Gate Array (FPGA) targets will also be demonstrated using the SAKURA-G board.

2 Related Work

The related work on cryptographic attacks may broadly be broken into two categories: methods of attacking algorithms using injected faults, and methods of injecting the faults on physical devices. Papers may often cover both categories: a new attack using fault injection is proposed, and this method is tested on a physical device. An excellent summary of papers in both these categories is given in [12].

Three main methods of injecting faults are compared here: clock glitching, power glitching, and electromagnetic (EM) glitching. The reader is referred to [3], [4], [12] for other available methods. A summary of work relevant to this paper for each of those three injection techniques will be presented next.

2.1 Clock Glitching

Clock glitching involves inserting additional rising edges into the input clock of the device, with the objective of violating timing constraints in the target device. For this to function, the clock must be used directly by the internal core. This means clock glitching will not be effective against two large classes of devices: those using internal oscillators, and those that use a Phase Lock Loop (PLL) to derive a new clock from the external clock. The majority of high-performance devices fall into the latter category, as they will run the internal core at a much higher frequency than the external clock.

Clock glitching on an Atmel AVR microcontroller is throughly presented in [15], which uses the same microcontroller family as being used by our work. In addition an extensive case study of clock glitching has been presented in [16] that used perturbations in the device power supply to improve the effect of the clock glitches, but did not consider the effect of power supply glitching alone. In [16] two devices are targeted: an ARM Cortex-M0 implemented in a NXP LPC1114 device, and an Atmel ATxmega 256 device.

These two papers demonstrate that with fine-grained control of the glitch timing, various instruction and data movements can be faulted with fine-grained control over the fault result.

2.2 EM Glitching

A typical EM glitch injection setup involves a precision X-Y table that can position the probe over the surface of the target chip. It has been demonstrated that for a successful glitch injection a very high precision is required when placing the probe over the chip surface [13], [17]. In addition if Package-on-Package technology is used in the target chip, this can make glitching more difficult, as a memory die has been stacked over the processor die [13].

EM glitching can achieve very fine-grained control over the fault effect, for example attempting to fault operations of specific registers [13]. EM glitching is a very powerful attack, but has the downside of requiring a more complex physical environment.

2.3 Power Glitching

Power glitching involves manipulation of the power supply of the target devices to generate faults; a simple example is how lowering the supply voltage will again introduce timing errors due to increased propagation delay. This *underpowering* has proven to introduce faults in ARM-9 devices during cryptographic operations [11]. This does not however provide good temporal accuracy, making it difficult for the glitch to target specific instruction.

The ability to target specific instructions can be achieved by instead inserting a 'spike' in the power rail at a specific instance in time. Both positive and negative voltage spikes can be inserted into the external power rails, where the spikes have a narrow width and attempt to cause faults in specific instructions. Both positive and negative spikes on the external rails result in similar waveforms internally in the target device, as demonstrated in [14].

A comparison of voltage glitching on three targets is given in [18], including attacking a 'secure' device. The authors of [18] provide a search methodology for determining ideal parameters of a glitch, i.e. finding the glitch amplitude and width. The results presented in [18] are extremely useful in visualizing the sensitivity of a system to a voltage glitch.

A comprehensive discussion of voltage glitching against FPGA target has been presented in [19], where the authors compared voltage glitching to laser (optical) glitching. In that work voltage glitching is shown to be effective against FPGAs for fault injection, and voltages in the range of 45V – 80V were found to be most effective for their experimental setup.

3 Crowbar Glitching Mechanism

The glitch mechanism explored in this work is a simple 'crowbar' circuit. This circuit applies a short across the power rails of the device, the specific waveform generated depending on the target device power supplies.

The glitch is generated with an N-Channel MOSFET (IRF IRF7807), driven using the glitch generation circuitry from the open-source ChipWhisperer hardware [20]. The selected MOSFET is a higher-power logic-level MOSFET with $88A$ of peak pulse current capability and 0.014Ω $R_{DS(ON)}$. As is typical for such a MOSFET, the gate charge is sufficient that generating very narrow glitches requires more care in the design of the driver circuit [21].

If very narrow glitches are required, a lower-power MOSFET (such as IRF IRLML2502) can be used, as this device has lower gate charge requirements, and can be switched faster than the higher-power MOSFET. This particular MOSFET has a $R_{DS(ON)}$ of 0.035Ω, meaning it would be less effective against low-impedance power rails likely to be found on high-speed processor boards.

4 Target Devices

The glitching attack is demonstrated against five targets: four microcontroller/microprocessor devices, and one FPGA device. The first target is a simple 8-bit microcontroller, the next three are various types of ARM-based System-on-a-Chip (SoC) devices, and the final target is a Xilinx Spartan 6 FPGA. The SoC devices are selected to represent those found in a wide variety of embedded systems, from single-board Linux computers to standard smartphones.

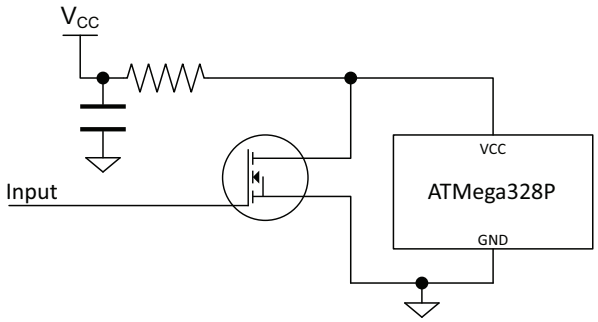

Fig. 1. The crowbar circuit using an N-Channel MOSFET is connected across the AVR power pins, and also allows power measurement across a shunt resistor.

4.1 AVR Microcontroller.

As the AVR microcontroller has been studied for clock glitching ([15]), and power glitching ([10], [18]), it serves as a useful benchmark for this work. Our work specifically uses the ATMega328P AVR microcontroller in DIP package running from a 7.3728 MHz crystal oscillator.

The glitching attack against the AVR uses the lower-power MOSFET (part number IRLML2502), connected as shown in Fig. 1. The series resistor serves two purposes: first, it allows the lower-power MOSFET to clamp the supply voltage towards zero, and second it allows simultaneous power-analysis, including triggering the fault based on patterns in the power consumption waveform.

4.2 Raspberry Pi (ARM11)

The *Raspberry Pi* is a low-cost single-board computer with an ARM11 based single-core processor, the BCM2835 from Broadcom, and runs at 700 MHz core frequency. This platform was loaded with Linux Debian with kernel 3.12.28.

For the power-glitching attack, we used the higher-power MOSFET IRF7807 connected across 220 nF decoupling capacitor C65, that capacitor being part of the VDD_{CORE} power distribution network. Additional details of the hardware setup are available as part of a tutorial[1].

4.3 Beaglebone Black (ARM Cortex-A8)

The *Beaglebone Black* is a low-cost development board with an ARM Cortex-A8 based single-core processor, the AM3358 from Texas Instruments (TI), and runs at 1 GHz core frequency. This is the most powerful platform tested in this paper, and runs Linux Debian with kernel 3.8.13-bone47.

This platform was selected in particular as it is also used as an EM glitching target by Hummel in [13]. Hummel extensively characterized the results of EM glitching over the surface of the main processor, and noted that careful positioning of the glitching coil was required to avoid simply rebooting the target.

Our power-glitching attack again used the higher-power MOSFET connected across 100 nF decoupling capacitor C63, part of the VDD_{MPU} rail. Note that attacks when the crowbar was connected across the VDD_{CORE} network were unsuccessful, only attacks against the VDD_{MPU} succeeded.

1. Details are posted as part of the ChipWhisperer Documentation, available at http://newae.com/sidechannel/cwdocs/tutorialglitchvcc.html

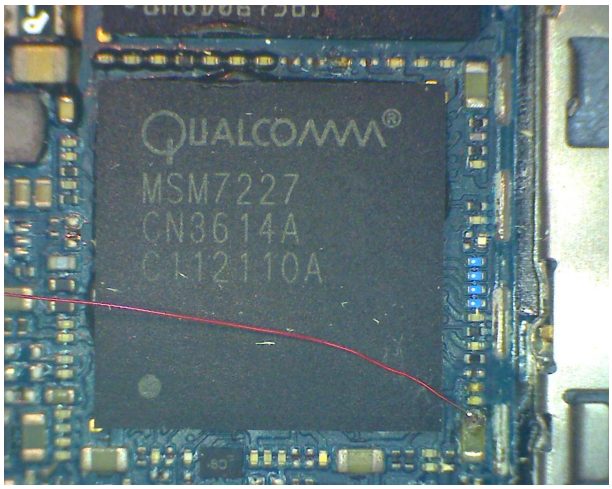

Fig. 2. A small length of magnet wire is used to connect a capacitor around the MSM7227 device to the crowbar MOSFET for glitch insertion. The ground connection comes from another point closer to where the MOSFET is mounted.

4.4 Android Smart Phone (ARM11)

A *HTC Wildfire S* smart phone was used with the stock image for this phone (Android 2.3.3). The main System-on-a-Chip (SoC) in this phone is a Qualcomm MSM7227. This is a highly integrated device with an ARM11 applications processor, applications DSP, ARM9 baseband processor, and baseband DSP. The user code will run in the ARM11 applications processor, which has a clock speed of 400 MHz.

The crowbar is attached across the capacitor shown in Fig. 2. This capacitor appears to be part of the power distribution network for the application processor core, based on comparison of the voltage at this point to the known core voltage of the device.

4.5 FPGA Board (SAKURA-G)

For this work the SAKURA-G board [22] is used as a platform for fault injection. This board contains a Spartan 6 LX75 FPGA (part number XC6SLX75-CSG484 in 2C speed grade) along with supporting circuitry. This board is designed for side-channel analysis so does not have capacitors mounted on the VCC_{INT} power rail, and contains a shunt resistor across this power rail. The lack of decoupling capacitors on this rail suggests the fault waveform should have little ringing when the crowbar is released.

The higher-power MOSFET (IRF7807) is used to short the VCC_{INT} power rail for the FPGA. As the SAKURA-G board contains a SMA connector on the VCC_{INT} rail, the MOSFET can be connected across this connector (J2).

5 Fault Insertion Results

Two types of faults were tested: in the first a simple code sample that should be highly sensitive to faults was tested, and in the second we explored faulting specific operations or data within algorithms. We refer to the first as a 'low-precision fault', as timing of the fault does not have a precise temporal trigger – the fault is being inserted at a

Listing 1. This code should result in `25000000-5000-5000` being printed for every successful loop.

```
int i,j,cnt;
while(1){
    for(i=0; i<5000; i++){
        for(j=0; j<5000; j++){
            cnt++;
        }
    }
    printf("%d-%d-%d\n", cnt, i, j);
}
```

random point during the clock cycle of the device. Low-precision fault insertion uses a fixed pulse width to activate the crowbar circuit.

For faulting specific operations or data, we use a 'high-precision fault', where specific temporal relationships between activity on the target device and fault injection time are maintained.

5.1 Low-Precision Faults on Microprocessors

For this low-precision work, the code being glitched is given in Listing 1. This was based on previously published glitching examples in [18]. The objective of the glitch is causing an incorrect count for the variable `cnt`. We do not explore the specific cause of the glitch (i.e. what instruction or data is being affected), only the resulting output was incorrectly calculated (i.e. a fault was inserted at some point).

This code is used on the four microcontroller / microprocessor targets. Discussion of low-precision faults on FPGA targets will be given in Section 5.2.

On the AVR target, Listing 1 is compiled directly onto 'bare metal' – there is no OS, only Listing 1 is running, with the `printf()` statements sending data over the serial port.

We have compiled Listing 1 as a regular user program on both the two Linux-based system and the Android system. The underlying OS will still be running background processes, and our objective is only to fault the user program. For the two Linux systems we interact the user program via a remote `ssh` terminal over the Ethernet connection, and with the Android system we interact using the touchscreen interface. The Android system uses a Java version of Listing 1.

This fault has been successfully applied against all four of the processor target devices described in Section 4. A successful fault is one where a single outer loop of the program from Listing 1 produces an incorrect result. The program must continue to run after the incorrect calculation without crashing. Details of the parameters for a successful fault are given in Table 1 for each target device.

As the fault is inserted at a random point in time, the only parameter to vary is the pulse width. The test program on the AVR has exclusive use of the core, as there is no OS, so a randomly inserted fault is almost certain to occur around a sensitive operation. On the Linux and Android system the underlying OS and other processes are also running, but due to the use of a infinite loop the test program will monopolize a single core, making it very likely a randomly

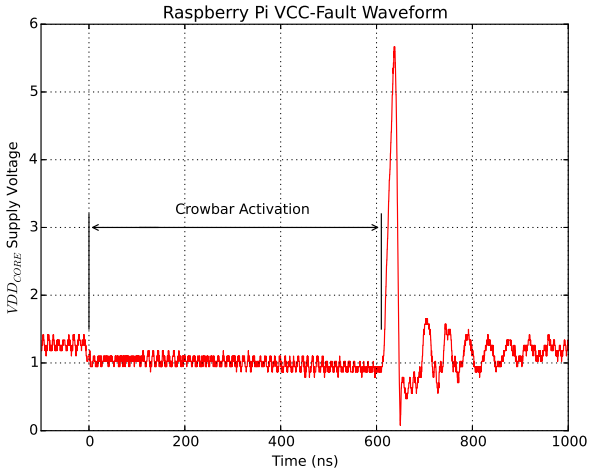

Fig. 3. The signal on the VCC_{CORE} rail for the Raspberry Pi during fault injection.

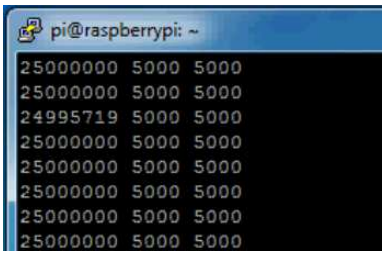

Fig. 4. An implementation of Listing 1 in C was used in a Linux application for testing purposes on the Raspberry Pi and Beaglebone Black. The output is monitored via a `ssh` connection, which is done to ensure the OS and network connection does not crash during the fault injection.

selected point in time will result in a fault inserted into our sensitive code rather than crashing the OS or a background process.

An example of the fault waveform for the Raspberry Pi device is given in Fig. 3. It can be seen the fault waveform involves both the power drooping while the crowbar is activated, along with substantial ringing once the crowbar is released.

The output of the software from Listing 1 running on the Raspberry Pi during a fault injection is shown in Fig. 4, and the Android Smartphone shown in Fig. 5.

This work does not characterize which aspects of this waveform are critical to fault generation, but instead simply parameterizes the fault based on length of time the crowbar is activated. The level and frequency of the ringing generated when the crowbar is released depends greatly on the power distribution network (PDN) design (including for example circuit board layout and number of decoupling capacitors), along with the location where the crowbar is connected across.

5.2 Low-Precision Faults on FPGAs

Fault injection on FPGAs has many uses, from simulating errors such as are expected from high-radiation environment [23] to attacking cryptographic implementations built

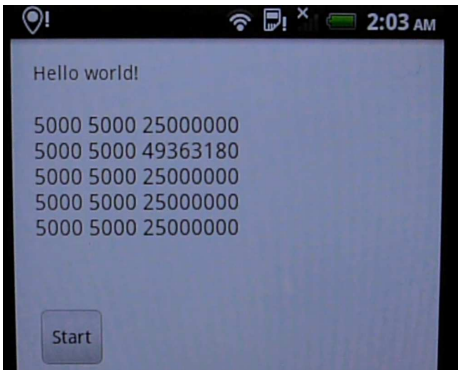

Fig. 5. An implementation of Listing 1 in Java was used in a simple Android application for testing purposes. The injected fault causes an incorrect count for a single loop iteration (49363180 instead of expected 25000000).

TABLE 1
Low-precision fault injection is used against all of these devices to cause the code from Listing 1 to calculate an incorrect result.

Target	Crowbar Activation Time
ATMega328P	135 nS
Raspberry Pi	635 nS
Beaglebone Black	485 nS
Android Smartphone	615 nS

on FPGA systems [24]. Work on the former has shown for example how to determine what specific type of errors occurred as a result of radiation-induced faults in an FPGA [25], and methods of simulating [26] or emulating [27] single-event upsets.

As mentioned, this work uses the SAKURA-G board [22] with a crowbar against the VCC_{INT} rail. As expected due to the lack of decoupling capacitors, the crowbar insertion has a very 'clean' waveform, as can be seen in Fig. 6. There is almost no ringing as a result of releasing the crowbar.

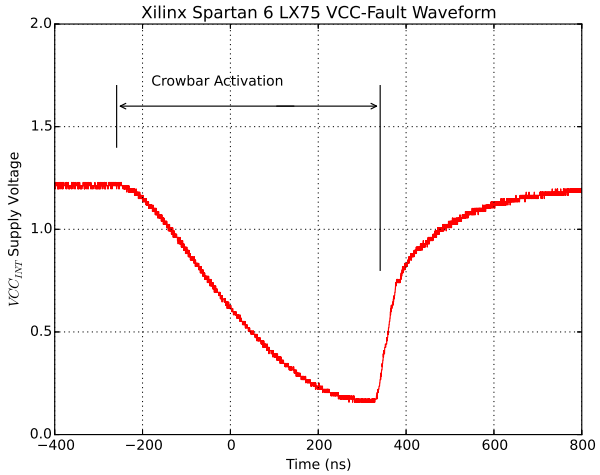

Fig. 6. The lack of decoupling capacitors on the SAKURA-G board along with inclusion of resistive shunt

5.2.1 FPGA Design

A basic design consisting of sixteen separate 32-bit registers is instantiated in the FPGA. Eight of these registers are loaded with all `1's` based on an external reset signal, and the other eight of these registers are loaded with all `0's` when that external signal is asserted.

The status of the registers are monitored by two external pins – this is able to detect one or more bits flipping from 0 to 1 (bit-set fault), or from 1 to 0 (bit-reset fault). An additional input signal temporarily overwrites the register value, used as a self-test to confirm the fault detection logic is still functioning.

As the configuration data of the FPGA itself is stored in SRAM and subject to corruption, the configuration data itself may become corrupted when inserting a fault [28]. A fault that is able to be cleared by asserting the external reset signal is considered a temporary fault (labeled a 'Design Register Fault' in the results from Table 2). If the design fails to function after the fault insertion even with an external reset, this is considered a 'Functional Failure'.

Determining that a 'functional failure' has occurred only means the configuration data specific to this design has been corrupted in such a way to prevent the design from working. It is also necessary to determine if other bits of the configuration data has been corrupted to properly characterize the fault injection results. These other configuration bits are portions of the FPGA that are not being used in the current design.

To accomplish this, the continuous CRC-check feature of the Spartan 6 FPGA can be used. This feature causes the FPGA to set the `INIT_B` pin to a logic low when the configuration memory of the FPGA changes. Monitoring this pin determines when a 'CRC Failure' has occurred, indicating the configuration data of the FPGA has been changed by the fault [29].

Once a 'CRC Failure' is detected, the readback feature of the FPGA is used to determine how many bits have flipped. A reference bitstream is first created based on a correctly loaded FPGA, and this reference is then compared to the new read-back file from the FPGA with a CRC failure. Based on the difference between these files the specific number of bits corrupted in the FPGA bitstream can be determined.

5.2.2 Fault Results

The results of various crowbar activation times on faults in the FPGA is given in Table 2. If the crowbar is activated longer than 900 nS, the FPGA enters a reset state and attempts to reload the configuration data.

For small fault injection widths ($\leq$550 nS), the configuration data of the FPGA is only occasionally corrupted (1 of 10 fault attempts causes at least one bit of configuration data corruption at 550 nS). Crowbar activation widths of 600 nS or greater always result in at least one bit of corruption of the configuration data stored in the FPGA. The number of bits corrupted tends to increase non-linearly with relation to crowbar activation time. The total FPGA readback bitstream has 2 452 898 bits, so for example if 1028 bits are corrupted this represents 0.042% of bits corrupted. A graph showing the relationship between glitch length and number of bits corrupted is provided in Fig. 7.

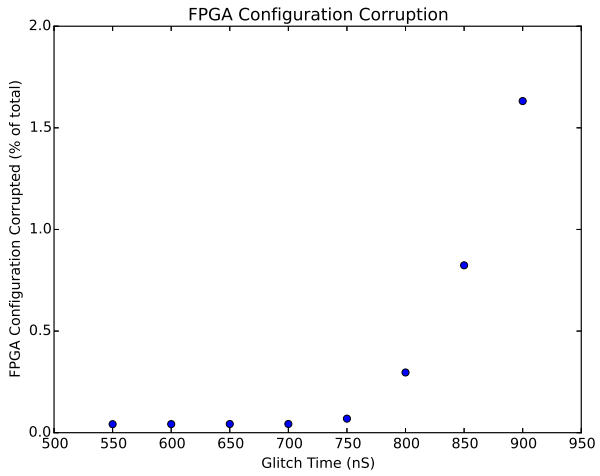

Fig. 7. Amount of FPGA configuration data corrupted based on length of crowbar activation.

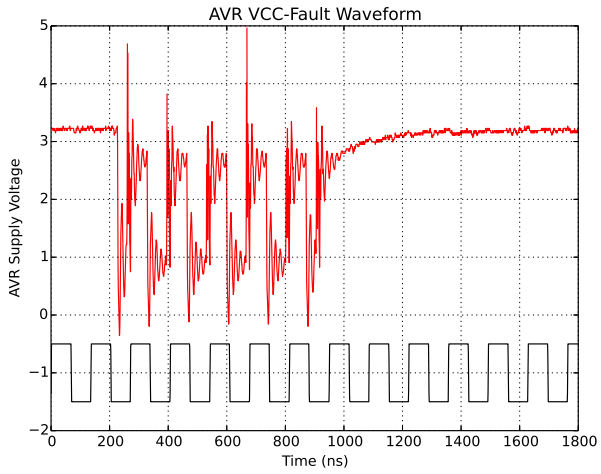

Fig. 8. The signal on the VCC pin when performing high-precision fault injections on the AVR is given in red. The black waveform is the 'glitch clock', which is four times the device clock.

Assuming the objective is to insert a fault into the registers inside the FPGA design, it can be seen there is an optimal glitch width that minimizes the amount of corruption within the configuration data, while still causing the values of registers to change inside the FPGA design. In this specific design a glitch width of about 750 nS would frequently (50 % of the time) result in one or more bit flip(s) in the register(s) without noticeably damaging the FPGA design. Note there *is* some corruption of the FPGA design, but the 'damage' is sparse enough to make a functional failure in the design unlikely.

These results demonstrate it is possible to use a crowbar fault mechanism on a FPGA to introduce random faults into both the configuration information and the registers used in the working FPGA design. Previous work on voltage fault attacks against FPGAs reported the ability to cause bit-flips in registers used within the FPGA design, but not modify the configuration information [19].

5.3 High-Precision Faults on AVR

The high-precision fault insertion uses a more complex fault waveform, shown in Fig. 8. This fault waveform is capable of activating the crowbar circuit for fractions of the clock cycle, and with precise timings from edges of the device clock. This requires access to the device clock to maintain synchronization, but it does not require the ability to manipulate the clock.

While this work did not explore high-precision fault attacks on a device with an internal oscillator or PLL, previous work has demonstrated the ability of a simple circuit to perform the clock recovery when no external oscillator is available [30]. Thus the work in this section should also be applicable to devices with internal oscillators or PLLs, where clock-glitching attacks are not possible.

The high-precision fault injection uses a trigger signal from the target device. The trigger signal indicates when the target device is performing the sensitive operating we wish to fault. For timing the crowbar activation, a 'fault clock' is generated that is phase-locked to the device clock, but operating at four times the frequency of the device clock.

This means the following parameters can be adjusted for each fault operation:

Starting Offset: After the trigger occurs, the number of cycles of the glitch clock before the crowbar is activated. This can be seen as starting the glitch during one of four phases of the device clock (as the glitch clock is four times the device clock).

Cycles Glitched: Number of cycles of the glitch clock during which the crowbar is activated. Note from Fig. 8 the crowbar is only activated for a portion of each cycle.

Phase Offset: The delay from the rising edge of the glitch clock to the crowbar being activated for that cycle. A positive offset indicates it is activated after the rising edge, a negative offset indicates before the rising edge.

Glitch Length: The length of time the crowbar is activated for within each cycle.

Three different code samples are used for fault injection. These samples are designed to test bit-set and bit-reset faults, along with exploring modifying single or multiple bytes within an operation (such as when targeting a specific byte of the AES state).

The code used for detecting bit-set faults is given in Listing 2, and bit-reset faults is given in Listing 3. Both of these samples are designed to use both SRAM and registers, along with repeating the operation over multiple clock cycles.

The results of varying the starting offset, cycles glitched, and phase offset is given in Fig. 9 and Fig. 10 for bit-set and bit-reset faults respectively (these figures appear at the end of this paper). The glitch length was fixed at 16.9 nS (50% of the glitch clock period). For each combination of parameters the output of the code sample given in either Listing 2 or Listing 3 is compared to the expected output. In addition to detecting either single-bit or multi-bit faults, reset of the device (via printing of a start-up sequence) is detected.

TABLE 2
Results of fault injection against Spartan 6 LX75 FPGA, repeated 10× for each width.

Width	SRAM Configuration Data Faults			Design Register Faults		
	CRC Failures	Functional Failures	Avg Bit Diff. of Failure	Set	Reset	Set & Reset
550 nS	1	0	1028	0	0	0
600 nS	10	0	1037	0	0	0
650 nS	10	1	1050	0	0	0
700 nS	10	0	1052	0	0	0
750 nS	10	0	1695	0	2	3
800 nS	10	0	7269	0	1	2
850 nS	10	0	20201	0	1	2
900 nS	10	8	40026	0	2	0

Listing 2. Passing `0x00` and `0x00` for both `a` and `b` allows this code to detect bit-set faults. As `a` is declared volatile the value is loaded from and saved to SRAM after each OR operation as shown in the resulting assembly code.

```
uint8_t glitch_bitset(volatile uint8_t a, uint8_t b) {
   trigger_high();
   a = a | b;
   //Each OR operation compiles to the following ASM:
   //ldd  r24, Y+1
   //or   r24, r22
   //std  Y+1, r24
   a = a | b;
     ...
   a = a | b;
   a = a | b;
   return a;
}

result = glitch_bitset(0x00, 0x00);
```

Listing 3. Passing `0xFF` and `0xFF` for both `a` and `b` allows this code to detect bit-reset faults.

```
uint8_t glitch_bitreset(volatile uint8_t a, uint8_t b){
   trigger_high();
   a = a & b;
   a = a & b;
     ...
   a = a & b;
   a = a & b;
   return a;
}

result = glitch_bitreset(0xFF, 0xFF);
```

The phase offset is varied from $-108°$ to $108°$ in $1.8°$ steps. The cycles glitched is varied from 4 to 40 cycles in 2-cycle steps. For all test cases the device is powered off and on after each fault attempt. This is to avoid errors caused by the device entering a lockup or failed state, or in case some unknown faults have been introduced that would affect future tests. Powering the device completely off and on achieves a reliable known-state for each test to be performed on.

These figures demonstrate that selecting the phase offset is a critical parameter for a successful fault insertion. Tuning of this parameter allows insertion of either single-bit or multi-bit faults in both the bit-set and bit-reset fault case.

To extend this to multi-byte operations, the code from Listing 4 is used. This code applies similar functions to those used in many cryptographic operations, but does not differentiate from bit-set and bit-reset faults.

This attack fixes the phase offset and cycles glitched parameters based on those discovered from the single-bit fault operations, in this case around 72 degrees phase offset and 5 cycles glitched.

The starting offset is then varied to attempt targeting of specific bits and bytes within an 8-byte array. The fault attempt is repeated for each starting offset 20 times, in order to determine the reliability of the fault operation.

As can be see in Fig. 11, faults can be targeted against

Listing 4. By comparing the value of array `a` after the call to `glitch_mb()` we can detect the location of faults across several bytes. Note certain bits are only sensitive to bit-set and certain bits are only sensitive to bit-reset faults.

```
void glitch_mb(uint8_t * a, uint8_t * b){
   trigger_high();
   for (uint8_t i = 0; i < 8;  i++){
      a[i] ^= b[i];
   }
}

void run_test(void){
   uint8_t a[8],b[8];

   for(uint8_t i = 0; i < 8; i++){
      a[i] = 0xAA;
      b[i] = 0xFF;
   }
   glitch_mb(a,b);

   //Value of 'a' is now checked
}
```

specific bytes within the array operation. Specific starting offsets have close to 100% reliability on fault insertion (this figure appears at the end of this paper).

5.4 High-Precision Faults on Raspberry Pi

Finally, we consider the effect of a specific fault attack on the Raspberry Pi. Whereas the results of the AVR high-precision fault attack can be directly applied to an algorithm-level attack, the Raspberry Pi's BCM2835 main SoC with a ARM1176 applications processor core contains a considerably more complex arrangement of registers and memory.

To determine the sensitivity of this device to algorithm-level attacks, we first use a specific cryptographic attack to determine where sensitive information is held, and then perform a targeted attack against that information.

6 Discussion

This section discusses the applicability of crowbar fault injection to real-world platforms.

6.1 Generating Fault Signals

The crowbar attack method requires a very simple fault signal. To replicate the results from Section 5.1, one only requires a pulse generator to drive the MOSFET. This signal can even be generated by a simple microcontroller if a laboratory pulse generator is not available. This makes it possible to add a fault generator to a system (such as connecting to a trusted computing module inside a laptop), while the user of the system is unaware of the fault generators presence. The attacker may choose to activate the fault module at a later point in time, or only have the module active during specific sensitive operations.

Replicating the results in Section 5.3 is easiest when using a FPGA-based system. Our work uses the ChipWhisperer platform [20], but almost any FPGA board is capable of performing the require clock multiplication and shifting used to generate the fault signal. As was mentioned in the results section, the fault waveform was extremely sensitive to the location of the first faulted cycle, along with phase of the fault relative to the device clock edge.

6.2 Finding Vulnerable Supplies

When attacking a device, it is required to determine the vulnerable supply rail. Even very simple devices will typically have at least two power rails (analog and digital), but more complex devices such as SoC could have many more (such as processor, memory, USB, clock domain, and analog).

This work attacked three such SoC devices, with varying levels of public documentation. The Beaglebone Black had full schematics and documentation published, including details of the SoC device. The Raspberry Pi has schematics but no details of the SoC, and the Android phone had no schematics and no details of the SoC. Determining the sensitive rail for each device will be discussed in sequence.

On the Beaglebone Black, the schematic shows there are two power rails of interest: VDD_{CORE} and VDD_{MPU}. Attempting to use a crowbar on the VDD_{CORE} was not successful, where the crowbar was inserted on a number of different locations underneath the BGA package. By comparison using a crowbar against the VDD_{MPU} rail was successful on the first attempt. As VDD_{MPU} is the *MicroProcessor Unit* rail, we would expect this rail to be the sensitive rail.

The Raspberry Pi also had schematics available, but in this case the SoC only had a VDD_{CORE} rail. Glitching against a randomly selected decoupling capacitor from this rail was successful.

The Android phone presented the most difficulty in determining the sensitive supply. There is no public documentation for the main SoC (Qualcomm MSM7227) device, and of course no schematics for the phone. Probing the decoupling capacitors mounted around the device showed 2.6V, 1.8V, 1.25V, and 1.33V being present. Based on the layout the 1.8V capacitors were likely part of the memory interface, leaving the 1.25V and 1.33V rails. Ultimately we found the 1.33V rail, using the point from Fig. 2, was a vulnerable location for fault insertion.

6.3 Triggering Faults

It has been shown in Section 5.3 that a very high precision of timing allows crowbar fault injection to achieve extremely high reliability. In practice, this can be achieved by using either a trigger signal from the target device (in the case of instrumentation purposely added), or with a trigger based on a power consumption or I/O activity trigger of the target device [20].

7 Conclusion

We have introduced a novel method of injecting voltage faults into hardware devices using a MOSFET to short the power supply of the device with very precise control over

timing of the faults. This is called the *crowbar* injection technique. The use of this technique against several platforms, including devices used in previous publications, has been presented. In addition several platforms are standard 'off-the-shelf' boards, showing how the crowbar technique can be used on real embedded systems.

The crowbar technique takes advantage of the properties of the power distribution networks on printed circuit boards to generate ringing in these networks. This ringing is presumed to perturb the power distribution networks on the target chip itself, which is known to cause faulty operations [14].

The use of fine control over the fault timing has also demonstrated that faults with very high reliability can be inserted, determining for example if a single- or multi-bit fault should be introduced, or to fault a single byte out of a larger array operation.

Currently this 'high-precision' faulting has only been demonstrated on simple 8-bit Atmel AVR microcontrollers. Future work is needed to test larger platforms such as embedded Linux computers to determine the reliability of high-precision fault attacks, and their ability to target very specific instructions or data.

8 Acknowledgments

This work funded by NSERC Canada Graduate Scholarship (CGS) for Colin O'Flynn.

References

[1] R. Rodrigues, S. Kundu, and O. Khan, "Shadow checker (sc): A low-cost hardware scheme for online detection of faults in small memory structures of a microprocessor," in *Test Conference (ITC), 2010 IEEE International*, Nov 2010, pp. 1–10.

[2] G.-C. Cardarilli, F. Kaddour, A. Leandri, M. Ottavi, S. Pontarelli, and R. Velazco, "Bit flip injection in processor-based architectures: a case study," in *On-Line Testing Workshop, 2002. Proceedings of the Eighth IEEE International*. IEEE, 2002, pp. 117–127.

[3] R. Anderson and M. Kuhn, "Low cost attacks on tamper resistant devices," in *Security Protocols*. Springer, 1998, pp. 125–136.

[4] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The Sorcerer's Apprentice Guide to Fault Attacks," *Proceedings of the IEEE*, vol. 94, no. 2, pp. 370–382, Feb 2006.

[5] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Advances in Cryptology – CRYPTO '97*, ser. Lecture Notes in Computer Science, J. Kaliski, BurtonS., Ed. Springer Berlin Heidelberg, 1997, vol. 1294, pp. 513–525.

[6] P. Dusart, G. Letourneux, and O. Vivolo, "Differential fault analysis on A.E.S." ser. Applied Cryptography and Network Security – ACNS 2003. Springer, 2003, vol. 2846, pp. 293–306.

[7] H. Choukri and M. Tunstall, "Round Reduction using Faults," in *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2005 Workshop on*, 2005, pp. 13–24.

[8] A. Barenghi, G. Bertoni, L. Breveglieri, M. Pellicioli, and G. Pelosi, "Fault attack on AES with single-bit induced faults," in *Information Assurance and Security (IAS), 2010 Sixth International Conference on*, Aug 2010, pp. 167–172.

[9] D. Boneh, R. DeMillo, and R. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," in *Advances in Cryptology – EUROCRYPT '97*, ser. Lecture Notes in Computer Science, W. Fumy, Ed. Springer Berlin Heidelberg, 1997, vol. 1233, pp. 37–51.

[10] J. Schmidt and C. Herbst, "A practical fault attack on square and multiply," in *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2008 Workshop on*, Aug 2008, pp. 53–58.

[11] A. Barenghi, G. Bertoni, E. Parrinello, and G. Pelosi, "Low Voltage Fault Attacks on the RSA Cryptosystem," in *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2009 Workshop on*, Sept 2009, pp. 23–31.

[12] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures," *Proceedings of the IEEE*, vol. 100, no. 11, pp. 3056–3076, Nov 2012.

[13] T. Hummel, "Exploring Effects of Electromagnetic Fault Injection on a 32-bit High Speed Embedded Device Microprocessor," Master's thesis, University of Twente, July 2014. [Online]. Available: http://essay.utwente.nl/65596/

[14] L. Zussa, J.-M. Dutertre, J. Clediere, and B. Robisson, "Analysis of the fault injection mechanism related to negative and positive power supply glitches using an on-chip voltmeter," in *Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on*, May 2014, pp. 130–135.

[15] 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*, 2011, pp. 105–114.

[16] T. Korak and M. Hoefle, "On the Effects of Clock and Power Supply Tampering on Two Microcontroller Platforms," in *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2014 Workshop on*, September 2014.

[17] N. Moro, A. Dehbaoui, K. Heydemann, B. Robisson, and E. Encrenaz, "Electromagnetic fault injection: Towards a fault model on a 32-bit microcontroller," in *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013 Workshop on*, Aug 2013, pp. 77–88.

[18] R. Carpi, S. Picek, L. Batina, F. Menarini, D. Jakobovic, and M. Golub, "Glitch It If You Can: Parameter Search Strategies for Successful Fault Injection," in *Smart Card Research and Advanced Application – CARDIS 2013*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 236–252.

[19] G. Canivet, P. Maistri, R. Leveugle, J. Clédière, F. Valette, and M. Renaudin, "Glitch and laser fault attacks onto a secure aes implementation on a sram-based fpga," *Journal of Cryptology*, vol. 24, no. 2, pp. 247–268, 2011. [Online]. Available: http://dx.doi.org/10.1007/s00145-010-9083-9

[20] C. O'Flynn and Z. Chen, "ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research," in *Constructive Side-Channel Analysis and Secure Design*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2014, vol. 8622, pp. 243–260.

[21] L. Balogh, "Design and application guide for high speed MOSFET gate drive circuits." Texas Instruments/Unitrode Corporation, Power Supply Design Seminar, SEM, 2001.

[22] H. Guntur, J. Ishii, and A. Satoh, "Side-channel attack user reference architecture board sakura-g," in *Consumer Electronics (GCCE), 2014 IEEE 3rd Global Conference on*, Oct 2014, pp. 271–274.

[23] F. Kastensmidt, L. Carro, and R. Reis, *Fault-Tolerance Techniques for SRAM-based FPGAs*. Springer, 2006, vol. 32.

[24] F. Khelil, M. Hamdi, S. Guilley, J. Danger, and N. Selmane, "Fault analysis attack on an fpga aes implementation," in *New Technologies, Mobility and Security, 2008. NTMS '08.*, Nov 2008, pp. 1–5.

[25] M. Violante, L. Sterpone, A. Manuzzato, S. Gerardin, P. Rech, M. Bagatin, A. Paccagnella, C. Andreani, G. Gorini, A. Pietropaolo *et al.*, "A new hardware/software platform and a new 1/e neutron source for soft error studies: Testing fpgas at the isis facility," *Nuclear Science, IEEE Transactions on*, vol. 54, no. 4, pp. 1184–1189, 2007.

[26] I. Chadjiminas, C. Kyrkou, T. Theocharides, M. Michael, and C. Ttofis, "In-field vulnerability analysis of hardware-accelerated computer vision applications," in *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*, Sept 2015, pp. 1–4.

[27] M. Alderighi, F. Casini, S. d'Angelo, M. Mancini, S. Pastore, and G. Sechi, "Evaluation of single event upset mitigation schemes for sram based fpgas using the flipper fault injection platform," in *Defect and Fault-Tolerance in VLSI Systems, 2007. DFT '07. 22nd IEEE International Symposium on*, Sept 2007, pp. 105–113.

[28] J. Wang, R. Katz, J. Sun, B. Cronquist, J. McCollum, T. Speers, and W. Plants, "Sram based re-programmable fpga for space applications," *Nuclear Science, IEEE Transactions on*, vol. 46, no. 6, pp. 1728–1735, Dec 1999.

[29] Xilinx, "Spartan-6 fpga configuration user guide (ug380)," Tech. Rep., 2015.

[30] C. O'Flynn and Z. Chen, "Synchronous sampling and clock recovery of internal oscillators for side channel analysis and fault injection," *Journal of Cryptographic Engineering*, vol. 5, no. 1, pp. 53–69, 2015.

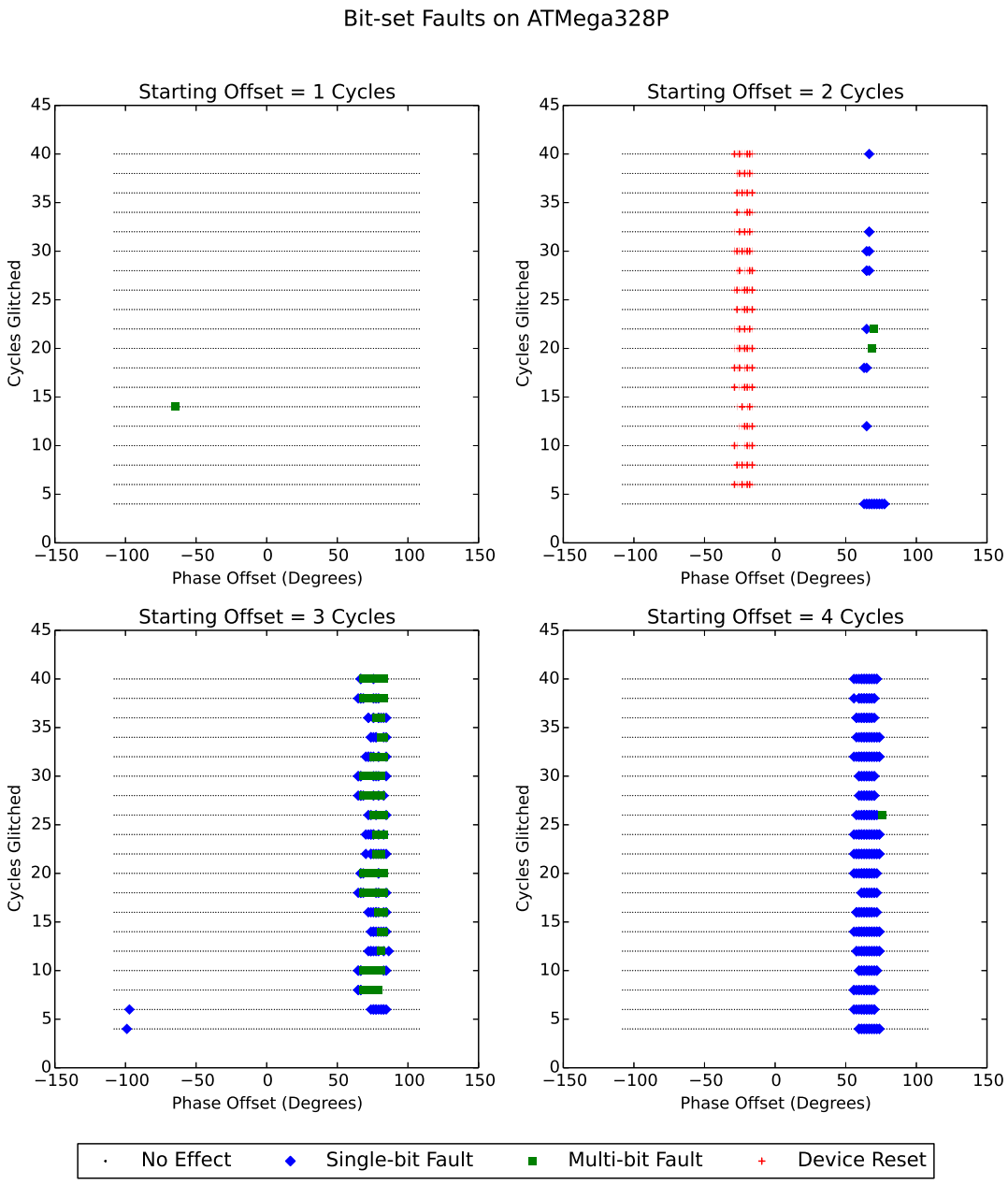

Fig. 9. Bit-set faults mean at least one bit that should have been a ‘0’ was read as a ‘1’. It can be seen both single-bit and multi-bit faults can be injected depending on the phase and starting offset.

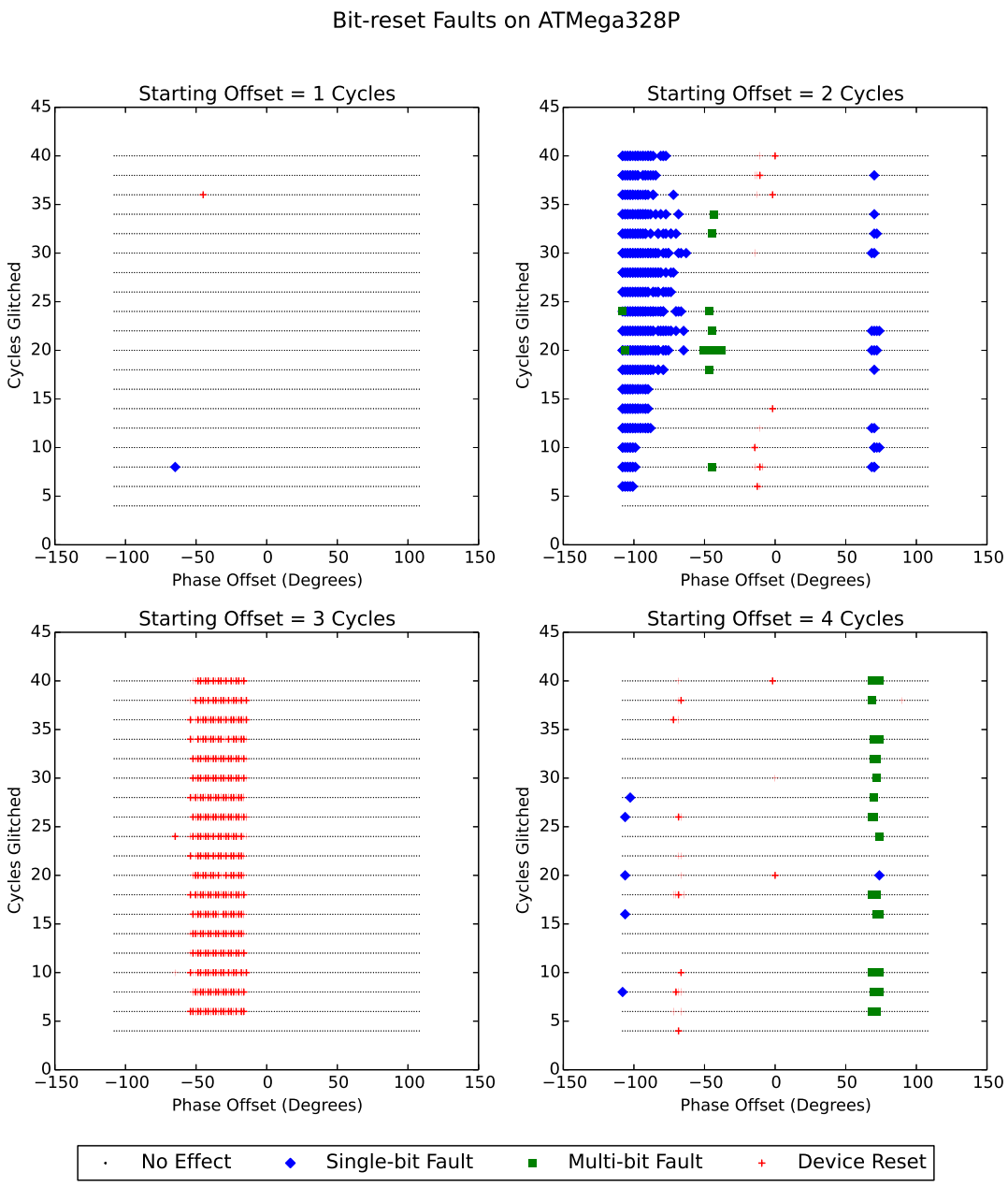

Fig. 10. Bit-reset faults mean at least one bit that should have been a ‘1’ was read as a ‘0’. It can be seen both single-bit and multi-bit faults can be injected depending on the phase and starting offset.

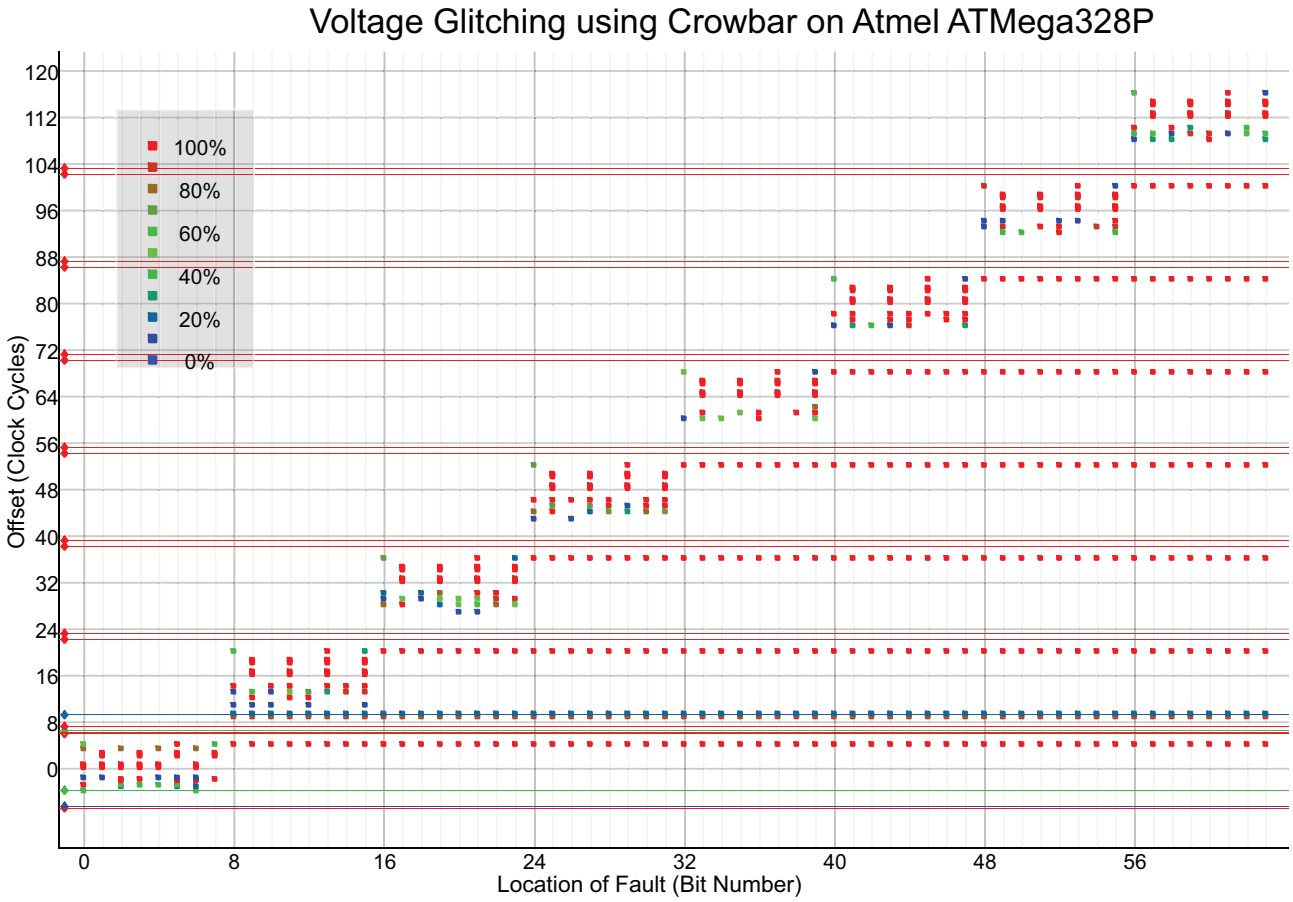

Fig. 11. A 64-bit number is manipulated, and the reliability of fault insertion on each bit for different glitch locations is graphed. Squares indicate a single bit fault, horizontal lines indicate a device reset. The color of the square (or line) indicates empirical probability of that fault result for a given offset. For example at an offset of 16.25 clock cycles results in the same four bits located within the second byte of the copy operating being marked as incorrect for 100% of observations. As the data being copied is `10101010` binary, this may indicate only those bits set to '1' were affected by a bit-reset fault.