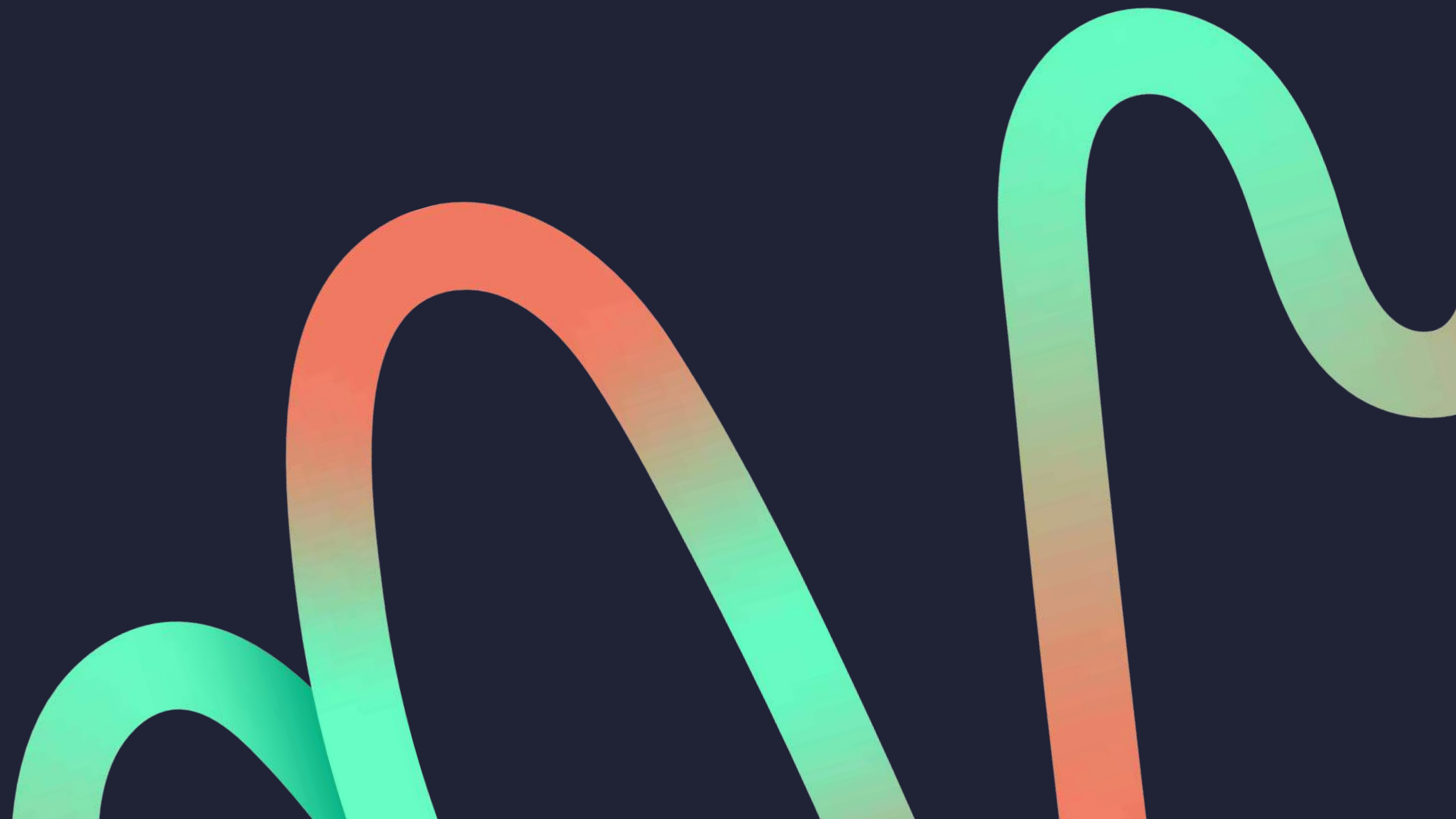
BUILDING GLITCH-RESISTANT FIRMWARE

Practical Software Countermeasures for Hardware glitch attacks

Arshid Shyam Kumar
Technical Expert, Siemens

Chinmay Krishna R
Student, IIIT-Bangalore

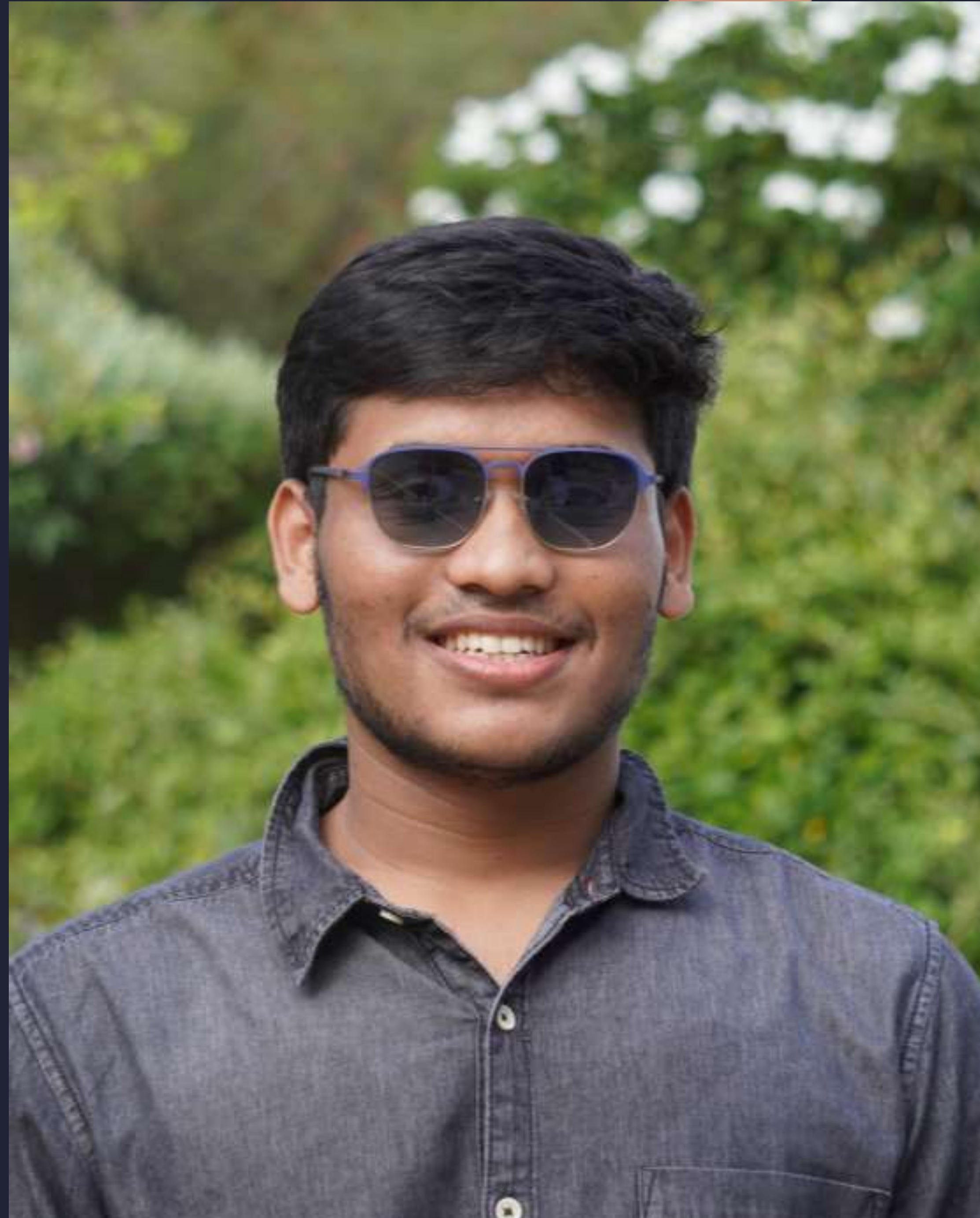
NULLCON GOA 2025



Arshid Shyam Kumar

- Hardware security, cloud security at Siemens Technology.
- Previously Cybersecurity work at NCIIPC and secure embedded systems development at ISRO.

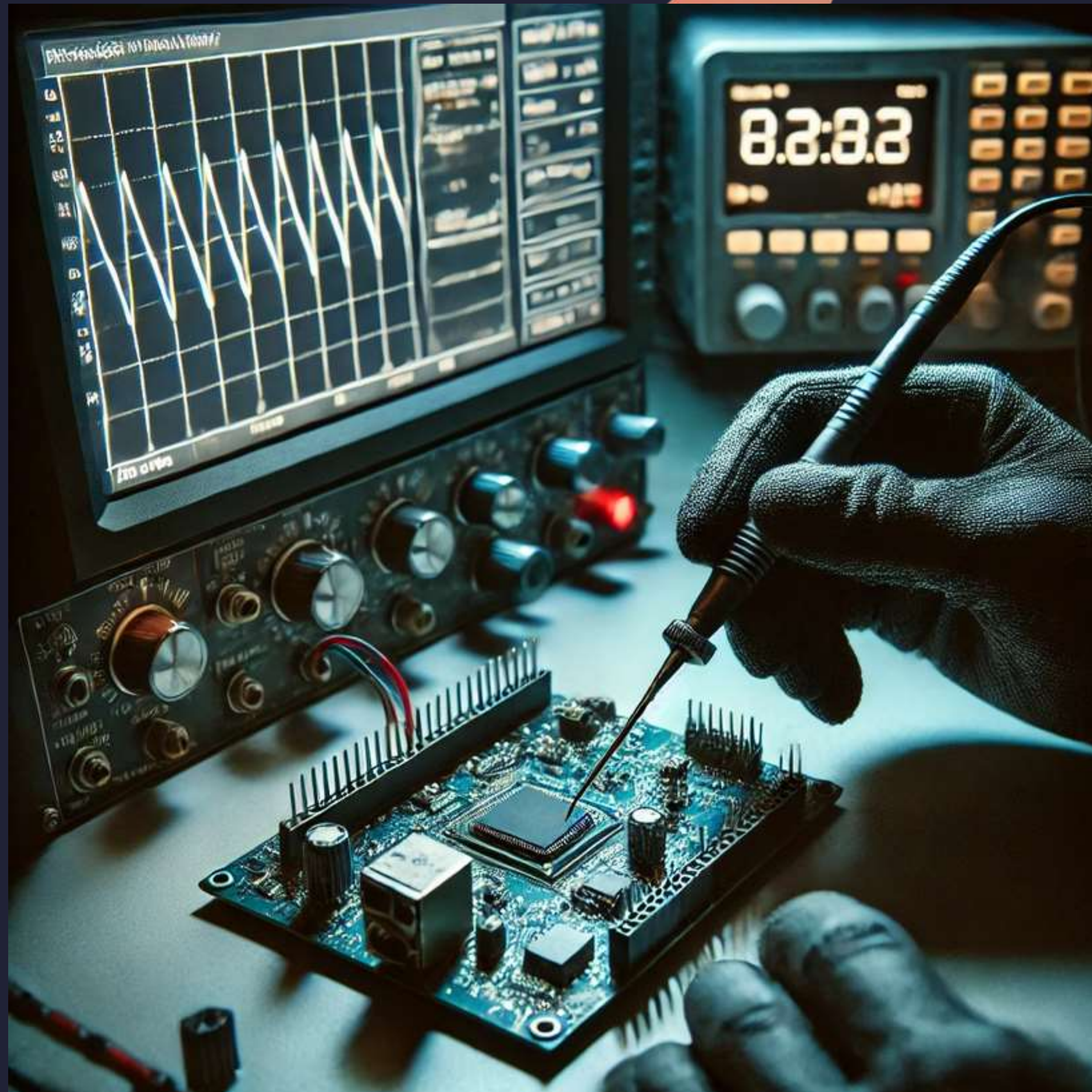
arshid-shyam.kumar@siemens.com



Chinmay Krishna R

- Junior year integrated Master's student in Electronics and Communications Engineering at IIIT-Bangalore
- Hardware security intern at Siemens Technology - Summer 2024.
- Digital VLSI design, FPGA’s, ASIC’s and embedded systems

Chinmay.Krishna@iiitb.ac.in



What are glitch attacks?

- Glitches manipulate hardware behavior to extract sensitive data, bypass authentication, or alter system functionality.
- These attacks exploit the physical nature of hardware, making them a significant threat to secure systems.
- Eg: voltage and clock glitching attacks

Voltage glitch

- **Apply glitch briefly** – Long enough to induce a faulty state but short enough to prevent a reset.
- **Precise timing** – Target vulnerable moments for effective glitching.
- **Modify components** – Desolder/bypass decoupling capacitors if needed.

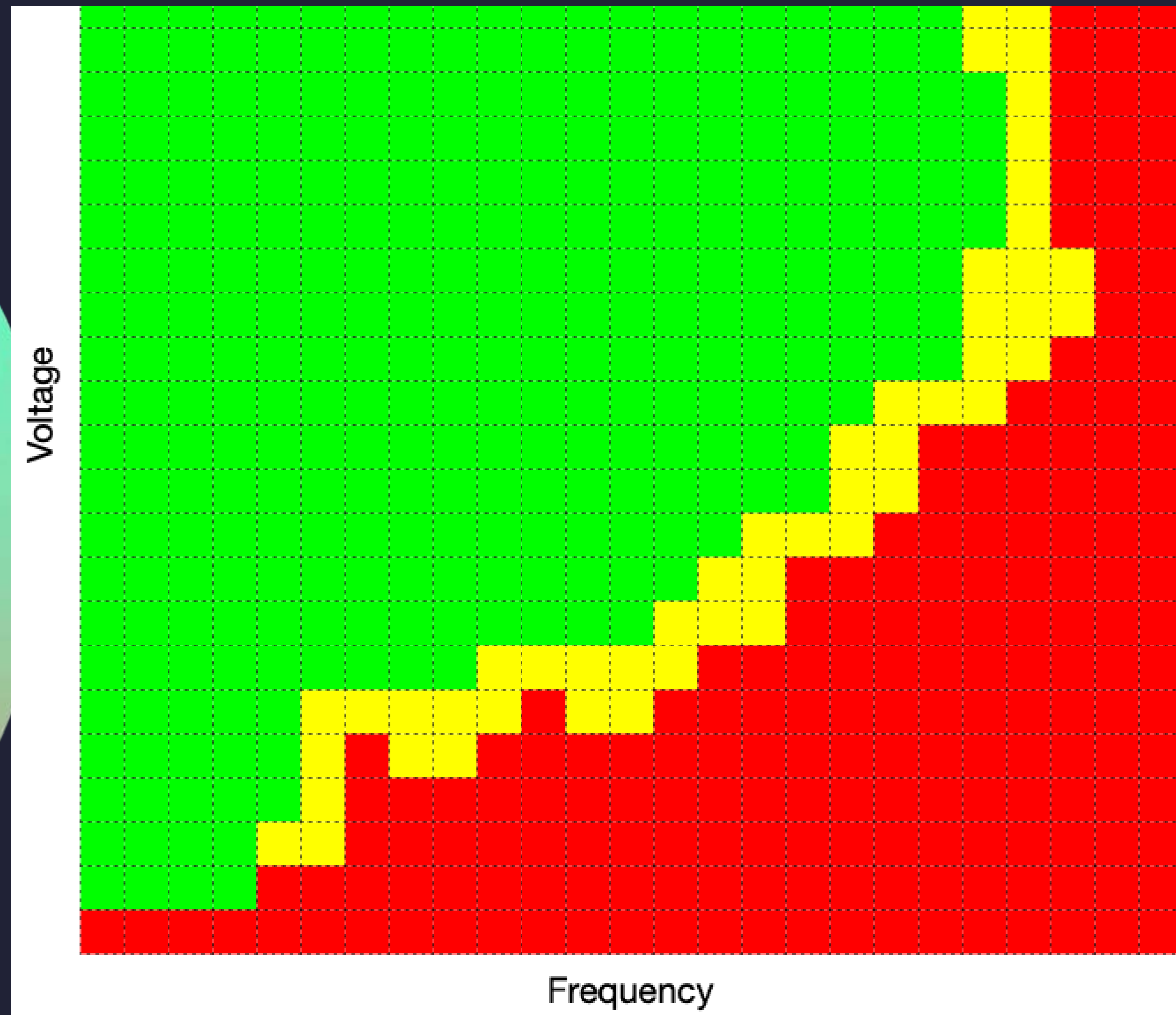
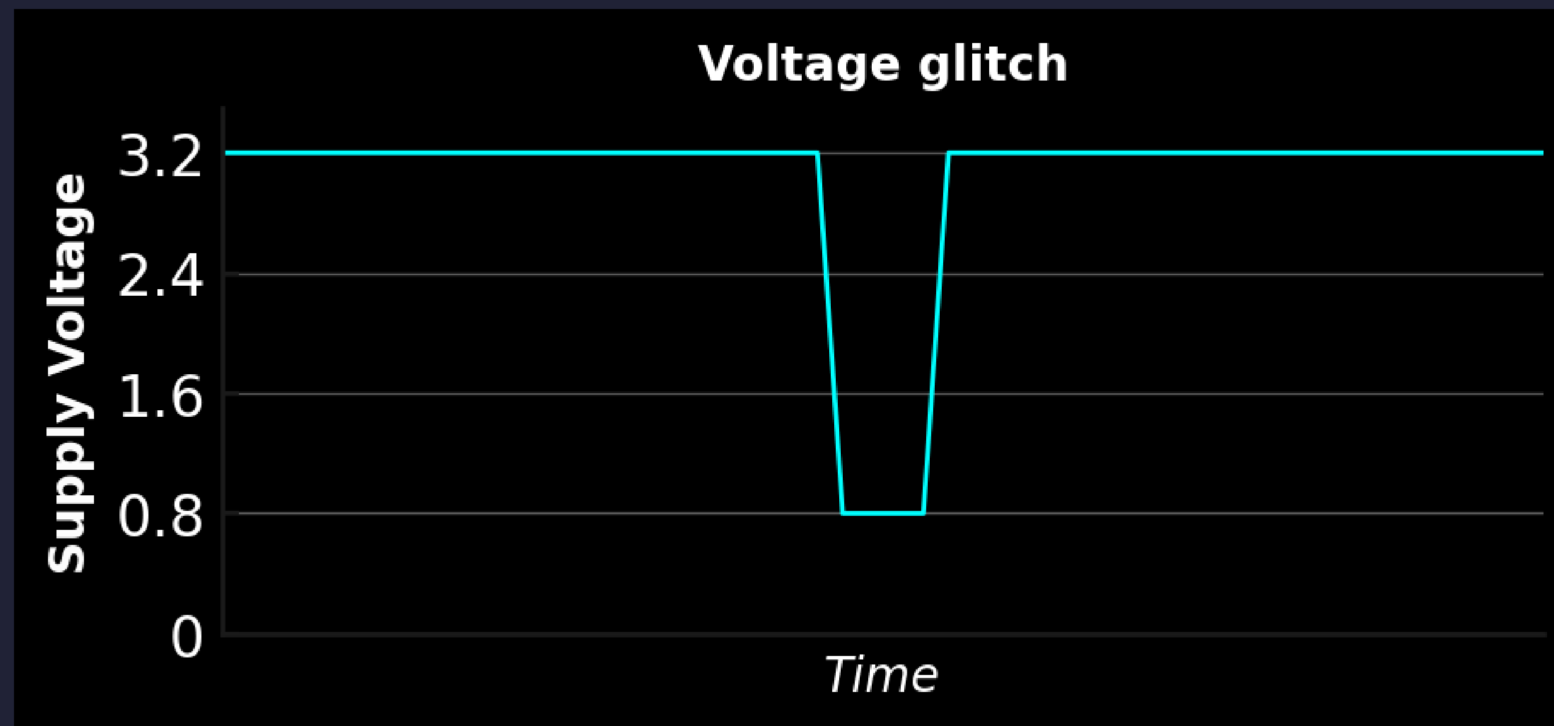
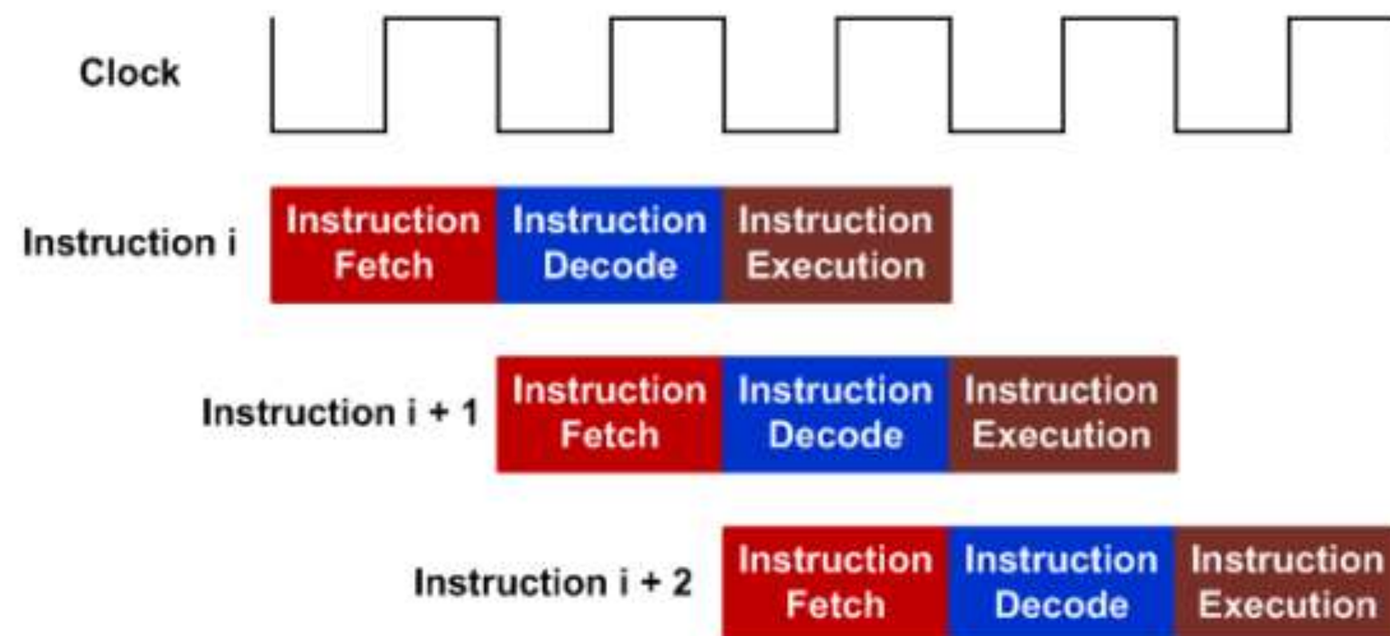


Image Ref: nccgroup

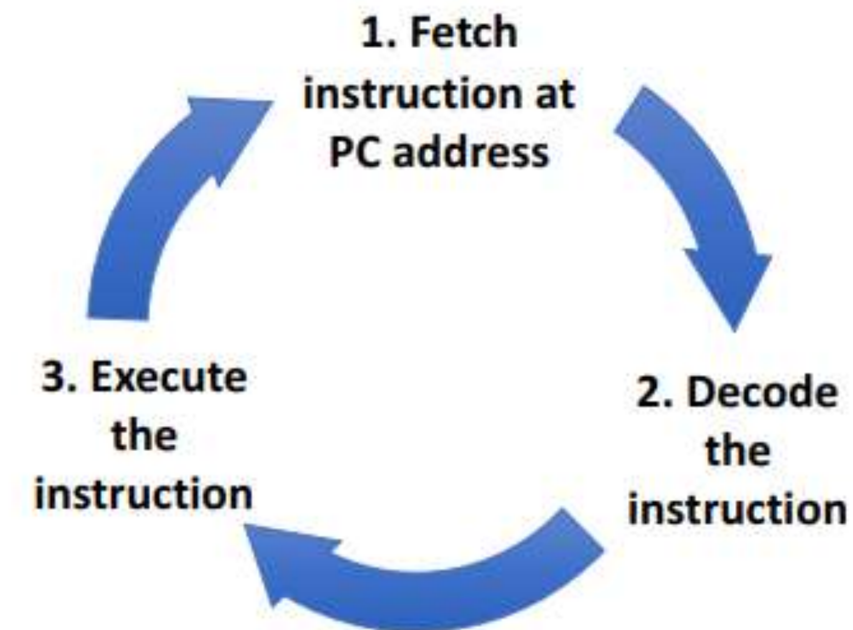


Cortex-M4 Pipeline

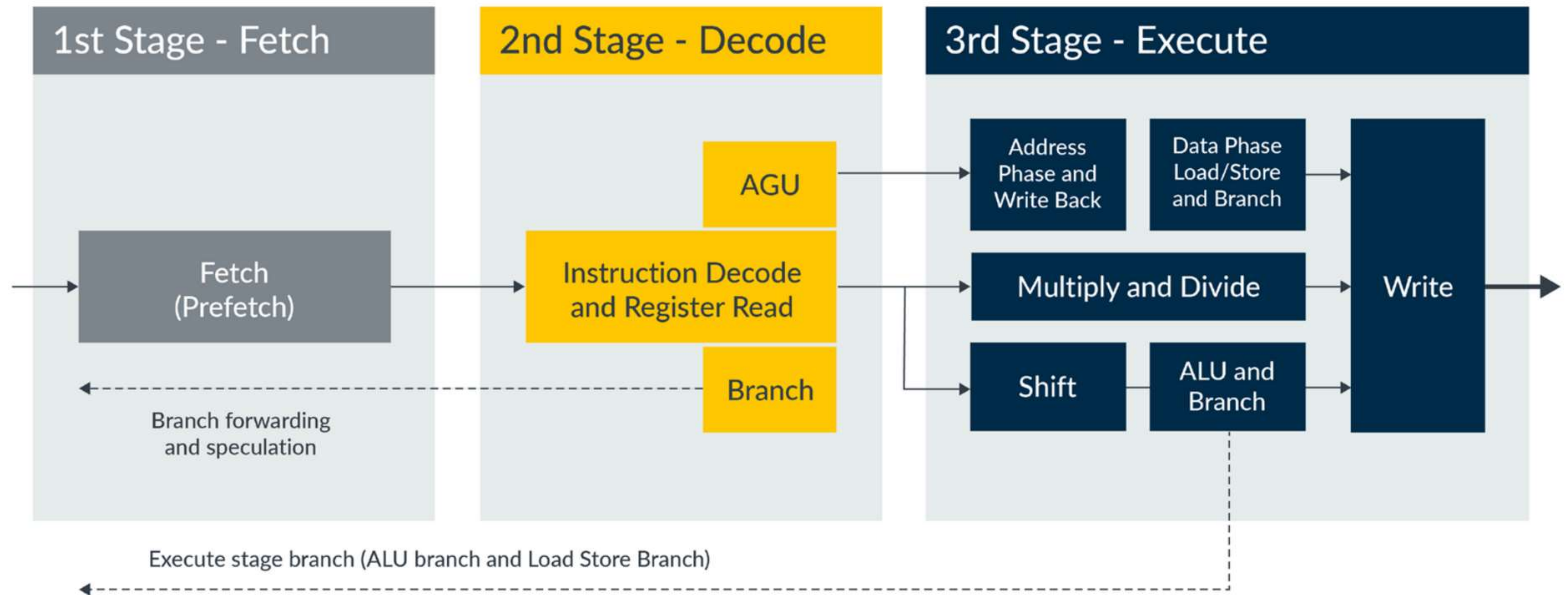
- Processor pipeline stages
  - Three-stage pipeline: fetch, decode, and execution
  - Some instructions may take multiple cycles to execute, in which case the pipeline will be stalled
  - The pipeline will be flushed if a branch instruction is executed
  - Up to two instructions can be fetched in one transfer (16- bit instructions)

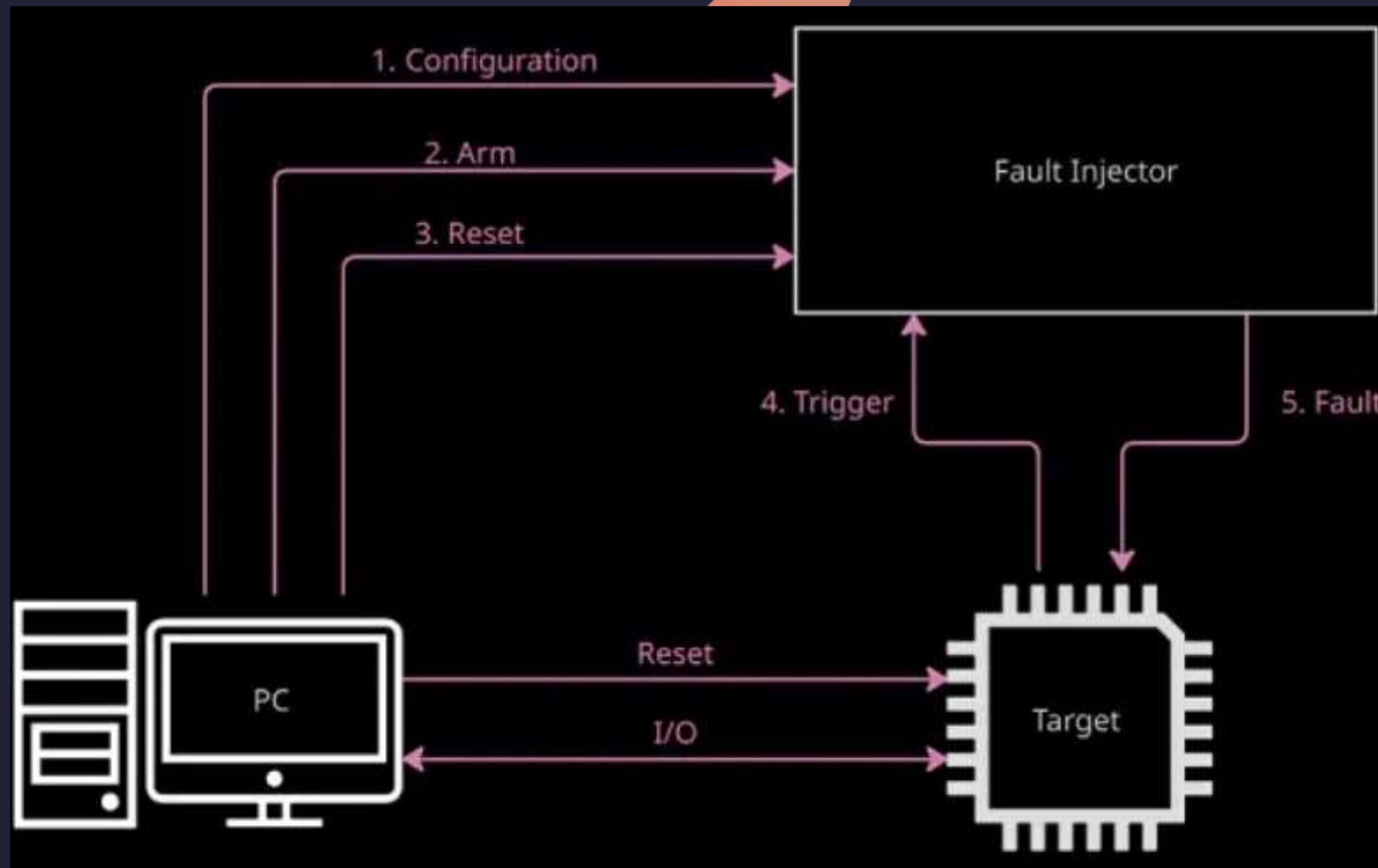


Pipeline of 32-bit instructions



Cortex-M4 Pipeline



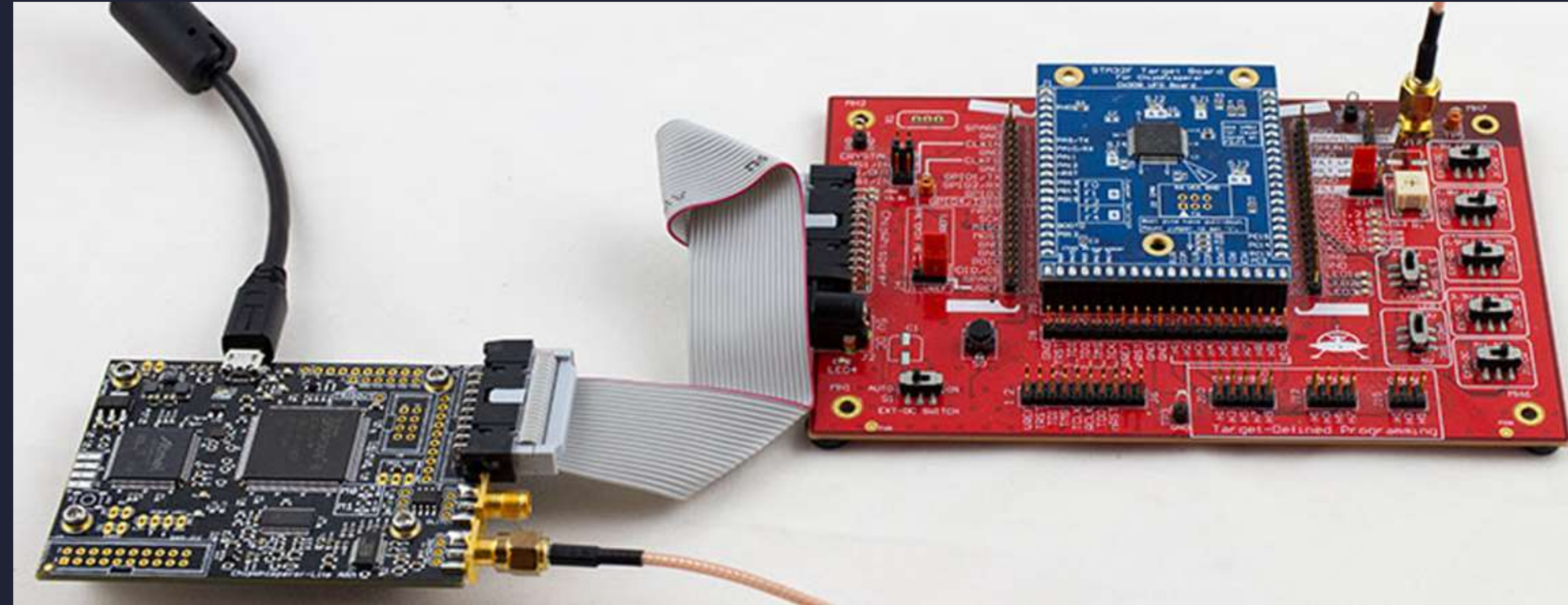


ChipWhisperer Lite Kit

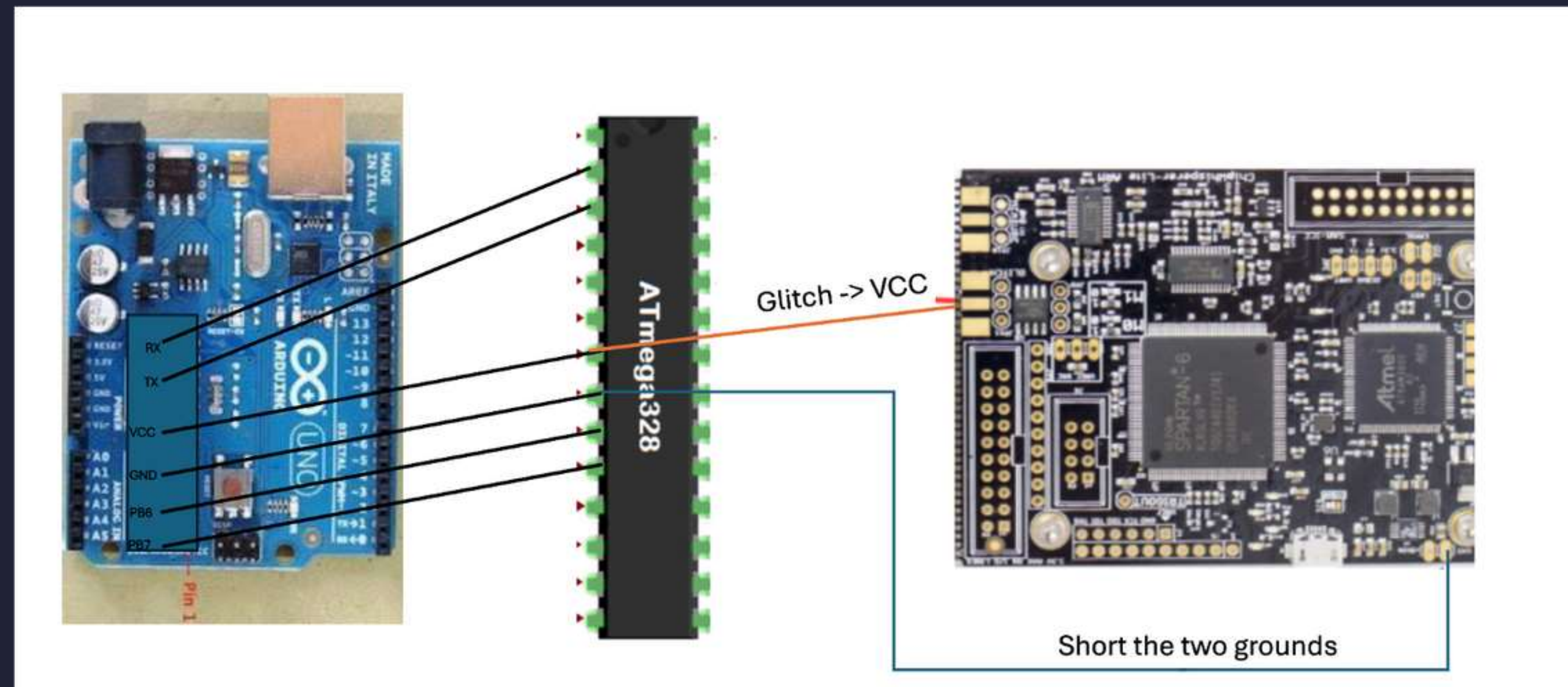
- One of the most popular open-source tools (hardware, software, firmware & FPGA code) for hardware security.
- Mainly used for side-channel power analysis and glitching attack.
- This research used the CW-lite with the provided STM32F3 32-bit target board.

Performing the glitch

Chipwhisper Lite kit



External boards



What do glitches do to the code?

```
(a) Original C code

int absdiff(int x, int y) {
    if (x < y)
        return y - x;
    else
        return x - y;
}

(c) Generated assembly code

x at %ebp+8, y at %ebp+12
movl    8(%ebp), %edx        Get x
movl    12(%ebp), %eax       Get y
cmpl    %eax, %edx           Compare x:y
jge     .L2                  If x >= y goto .L2
subl    %edx, %eax           Compute result y-x
jmp     .L3                  Goto done
.L2:
subl    %eax, %edx           Compute result x-y
movl    %edx, %eax           Set result as return value
.L3:                         done: Begin completion code
```

TAKE ADVANTAGE OF THIS FAULT TO SKIP INSTRUCTIONS FROM THE CRITICAL SECTION CODE

Password Bypass

```
75  uint8_t password(uint8_t* pw, uint8_t len)
76  #endif
77  {
78      char passwd[] = "touch";
79      char passok = 1;
80      int cnt=0;
81
82      trigger_high();
83
84      for(cnt = 0; cnt < 5; cnt++){
85          if (pw[cnt] != passwd[cnt]){        <-- Glitch here
86              passok = 0;
87          }
88      }
89
90      trigger_low();
91
92      simpleserial_put('r', 1, (uint8_t*)&passok);
93      return 0x00;
94  }
```

Glitch here

The glitch skips the
password check altogether

Countermeasures

Hardware Methods	Software Methods
• Brown-out Detection (BOD) circuitry • Clock and Power Integrity Checks • Shadow registers can improve fault resiliency • Hardware-based pointer authentication	• Redundant Computation • Timing Randomization • Control Flow Integrity (CFI) • Runtime Integrity Checks

HARDware methods

- Long redesign lead times
- Significant production cost overhead
- Difficult retrofits for existing designs
- Limited post-deployment updates

Image Ref: Sciencedirect

The way you write code matters!

Simple check

```
75  uint8_t password(uint8_t* pw, uint8_t len)
76  #endif
77  {
78      char passwd[] = "touch";
79      char passok = 1;
80      int cnt=0;
81
82      trigger_high();
83
84      for(cnt = 0; cnt < 5; cnt++){
85          if (pw[cnt] != passwd[cnt]){
86              passok = 0;
87          }
88      }
89
90      trigger_low();
91
92      simpleserial_put('r', 1, (uint8_t*)&passok);
93      return 0x00;
94  }
```

Attempt	Glitch Success	Glitch Failure
1	82	6796
2	65	7345
3	74	7145

Volatile loop counter

```c
#if SS_VER == SS_VER_2_1
uint8_t password(uint8_t cmd, uint8_t scmd, uint8_t len, uint8_t* pw)
#else
uint8_t password(uint8_t* pw, uint8_t len)
#endif
{
    char passwd[] = "touch";
    char passok = 1;
    volatile int cnt=0;

    trigger_high();

    for(cnt = 0; cnt < 5; cnt++){
        if (pw[cnt] != passwd[cnt]){
            passok = 0;
        }
    }

    trigger_low();

    simpleserial_put('r', 1, (uint8_t*)&passok);
    return 0x00;
}
```

Attempt	Glitch Success	Glitch Failure
1	1	8369
2	0	9203
3	0	9203

Why did that happen?

VOLATILE

Non volatile	Volatile
84:simpleserial-glitch.c **** for(cnt = 0; cnt < 5; cnt++){ 264 .loc 1 84 5 view .LVU45 265 00f0 F901 movw r30,r18 266 00f2 DE01 movw r26,r28 267 00f4 1196 adiw r26,1 268 00f6 2B5F subi r18,-5 269 00f8 3F4F sbci r19,-1 271 .L14:	84:simpleserial-glitch.c **** for(cnt = 0; cnt < 5; cnt++){ 269 00f6 1F82 std Y+7,__zero_reg__ 270 00f8 1886 std Y+8,__zero_reg__ 271 .loc 1 84 5 view .LVU48 272 00fa 41E0 ldi r20,lo8(1) 274 .L13: 275 .loc 1 84 22 is_stmt 1 discriminator 1 view .LVU49 276 00fc 2F81 ldd r18,Y+7 277 00fe 3885 ldd r19,Y+8 278 0100 2530 cpi r18,5 279 0102 3105 cpc r19,__zero_reg__ 280 0104 04F0 brlt .L15

MAJOR CHANGES IN

Duplicating variables

```
#if SS_VER == SS_VER_2_1
uint8_t password(uint8_t cmd, uint8_t scmd, uint8_t len, uint8_t* pw)
#else
uint8_t password(uint8_t* pw, uint8_t len)
#endif
{
    char passwd[] = "touch";
    char passok = 0;
    int cnt=0;
    char pw1[4];
    for(int i=0; i<4; i++){
        pw1[i] = passwd[i];
    }
    trigger_high();


    if (strcmp(pw1, passwd) == 0){
        for(cnt = 0; cnt < 5; cnt++){
            if (pw[cnt] == passwd[cnt]){
                passok = 1;
            }
            else{
                passok = 0;
            }
        }
    }

    trigger_low();

    simpleserial_put('r', 1, (uint8_t*)&passok);
    return 0x00;
}
```

Attempt	Glitch Success	Glitch Failure
1	1	14964
2	0	14944
3	0	14915

Why did that happen?

DUPLICATION

- Duplicate password variable for strcmp.
- Extra verification layer resists glitches.
- Corruption triggers early strcmp failure.
- More complexity requires longer bypass.

Inverting variables

```c
uint8_t password(uint8_t* pw, uint8_t len)
#endif
{
    char passwd[] = "touch";
    char passok = 0;
    int cnt=0;
    char pw1[5];
    for(int i=0; i<5; i++){
        pw1[i] = ~passwd[i];
    }

    trigger_high();

    char pw2[5];
    for(int i=0; i<5; i++){
        pw2[i] = ~passwd[i];
    }

    if (strcmp(pw1, pw2) == 0){
        for(cnt = 0; cnt < 5; cnt++){
            if (pw[cnt] == passwd[cnt]){
                passok = 1;
            }
            else{
                passok = 0;
            }
        }
    }

    trigger_low();

    simpleserial_put('r', 1, (uint8_t*)&passok);
    return 0x00;
}
```

Attempt	Glitch Success	Glitch Failure
1	0	4275
2	0	4265
3	0	4225

Why did that happen?

INVERTING

- Inverted password outside the trigger.
- Secondary inversion inside trigger_high().
- Multiple operations need perfect timing.
- Added strcmp increases glitch difficulty.

Masking variables

```c
uint8_t password(uint8_t* pw, uint8_t len)
#endif
{
    char passwd[] = "touch";
    char passok = 0;
    int cnt=0;
    char pw1[5];
    for(int i=0; i<5; i++){
        pw1[i] = passwd[i]*5;
    }

    trigger_high();

    char pw2[5];
    for(int i=0; i<5; i++){
        pw2[i] = passwd[i]*5;
    }

    if (strcmp(pw1, pw2) == 0){
        for(cnt = 0; cnt < 5; cnt++){
            if (pw[cnt] == passwd[cnt]){
                passok = 1;
            }
            else{
                passok = 0;
            }
        }
    }

    trigger_low();

    simpleserial_put('r', 1, (uint8_t*)&passok);
    return 0x00;
}
```

Attempt	Glitch Success	Glitch Failure
1	0	8012
2	0	7958
3	0	8115

Why did that happen?

MASKING

- Password masked by multiplying characters outside glitch.
- Masked password made again inside glitch
- Glitch must hit masking and comparison.
- Double masking increases glitch difficulty.

Negating the logic

```c
#if SS_VER == SS_VER_2_1
uint8_t password(uint8_t cmd, uint8_t scmd, uint8_t len, uint8_t* pw)
#else
uint8_t password(uint8_t* pw, uint8_t len)
#endif
{
    char passwd[] = "touch";
    char passok = 0;
    int cnt=0;

    trigger_high();

    for(cnt = 0; cnt < 5; cnt++){
        if (pw[cnt] == passwd[cnt]){
            passok = 1;
        }
        else{
            passok = 0;
            break;
        }
    }

    trigger_low();

    simpleserial_put('r', 1, (uint8_t*)&passok);
    return 0x00;
}
```

Attempt	Glitch Success	Glitch Failure
1	2731	9966
2	2393	10035
3	2408	9912

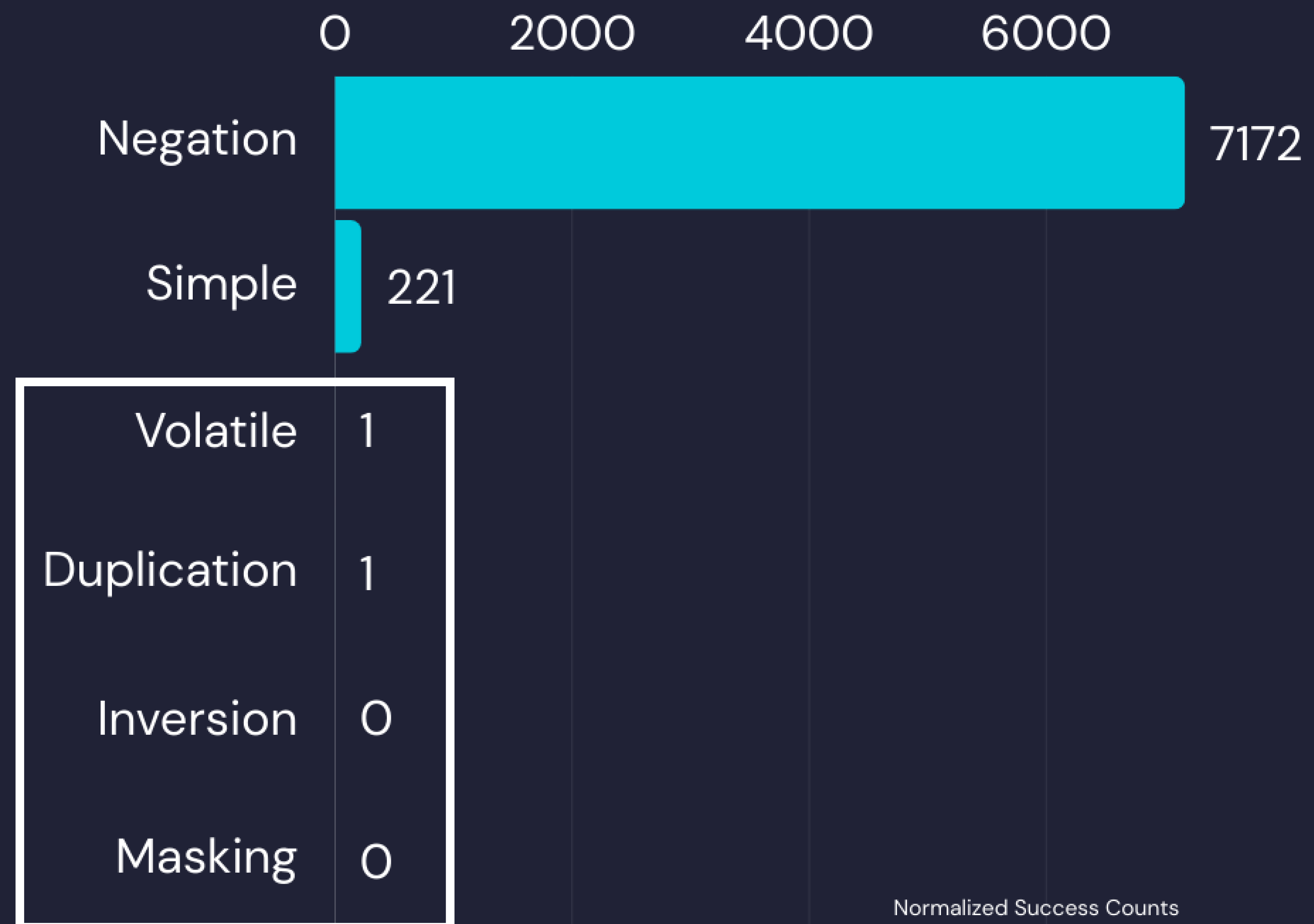
Why did that happen?

NEGATING

- Single fault can bypass check
- Lack of strong failure handling
- Early termination exploit

Results

*All tests were ran for 10 minutes each



These work best

Software Glitch Defenses in the Real World

1

Arm's TrustedFirmware

Use FIH library for glitch resilience, now a standard recommendation

2

Open-Source Tools/Libraries

GlitchResistor, ChipArmour etc.

3

WolfBoot & Industry Adoption

Bootloader to implement mitigations against glitching attacks

Key Takeaways

1

Faults Are Physical, Mitigations Are Logical

2

Minimal Code Tweaks → Big Gains

3

Secure Coding & use of Libraries

DO YOU HAVE ANY QUESTIONS?