This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2020-3369C

Information Leakage Analysis of a RISC-V Microprocessor using Accelerated Fault Injection

Tom J. Mannos (SNL), Jim Plusquellic (UNM), Brian J. Dziki (DoD)

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Bottom Line Up Front

Fault Testing Architecture

- Berkeley RISC-V (Rocket RV64IMA) microprocessor used here extends previous work on a 32-bit Leon3 microprocessor
- RISC-V architecture has more than 85K fault injection sites, requiring an accelerated fault testing and data collection approach.
- Advanced features of FPGA emulation platform are leveraged for acceleration, including use of dynamic reconfiguration and processor-side/programmable-logic-side high speed GPIO
- Emulation platform enabled 'scrubbing' between each fault experiment, and additional clarity on level of fault propagation across fault tests

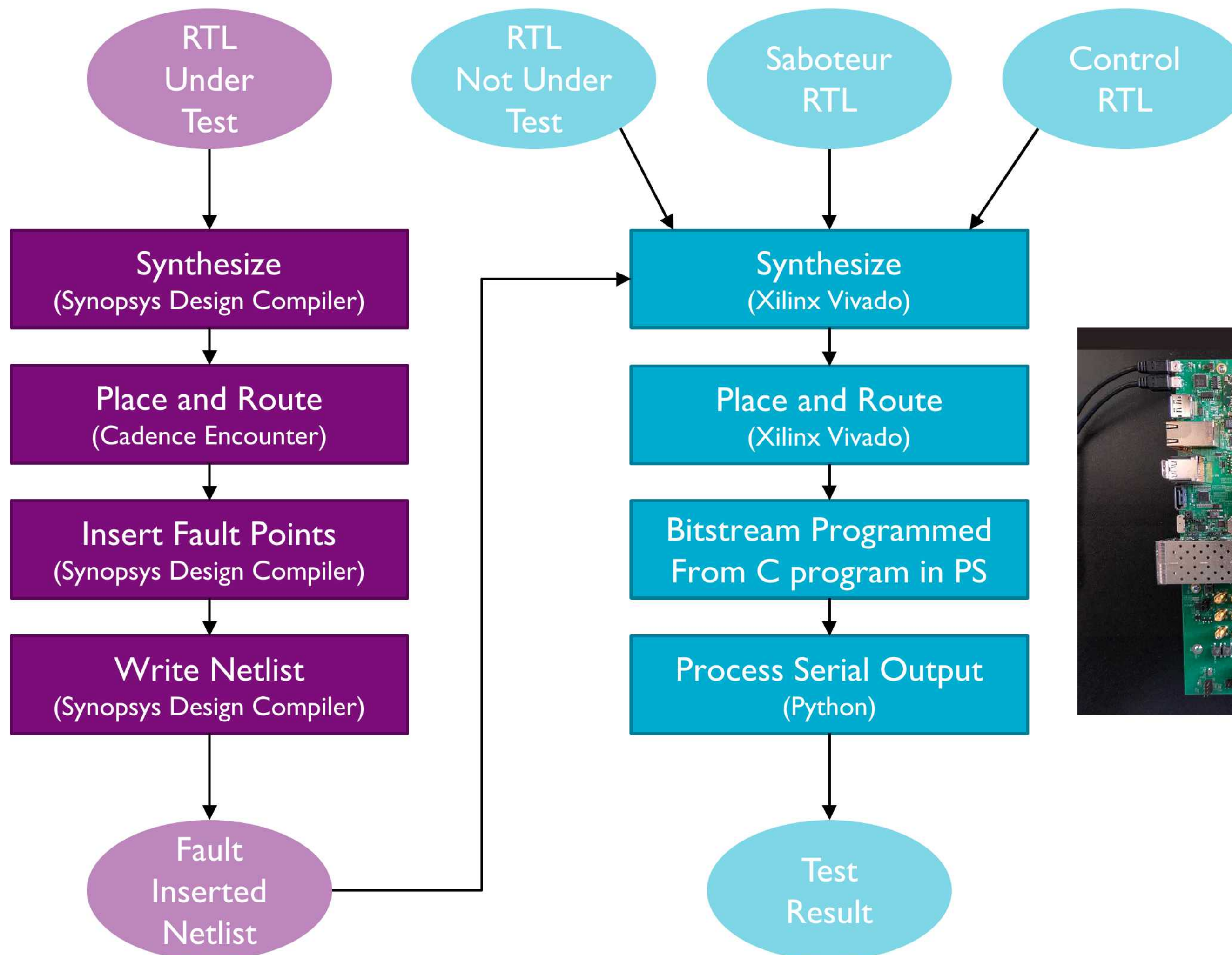
Fault Testing Results

- Tested 4 fault types, 1-5 faults inserted simultaneously, two AES key-plaintext encryptions, with and without scrubbing: More than 6.8 million fault testing experiments
- The Rocket implementation was more susceptible to data leakage from hardware faults than the Leon3 core tested last year.

Outline

- RISC-V CAD tool flow and implementations
- Saboteur Circuit
- Testing Process, Scan Chain, GPIO interface, PCAP and C program
- Expected and Observed Output
- Experimental Results
- Next Steps

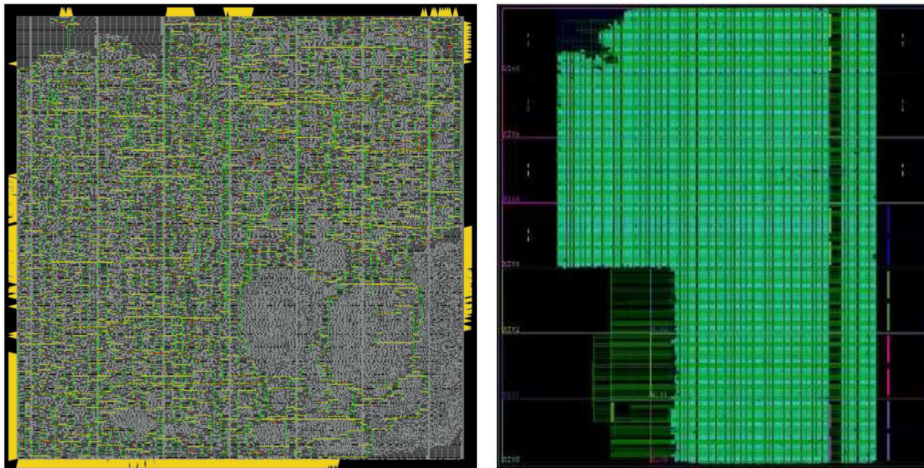
Design Flow



UNCLASSIFIED UNLIMITED RELEASE

RISC-V Test Design

- Based on River SoC https://github.com/sergeykhbr/riscv_vhdl
- 64 KB block ROM for application code
- 8192 KB block RAM for scratch memory
- ASAP7 7nm academic library from ASU

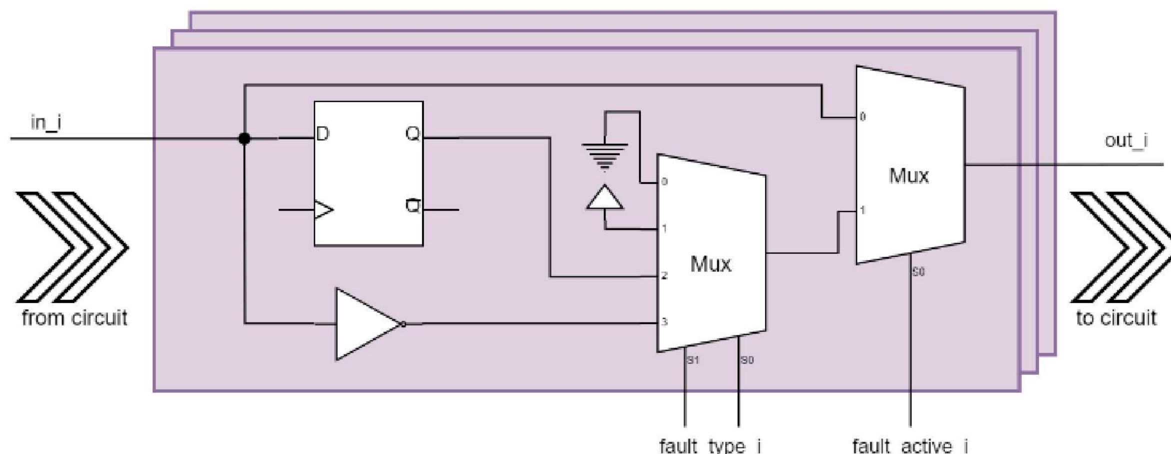


RISC-V layout (left) and Zynq UltraScale+ FPGA test platform (right).

AES Program Listing

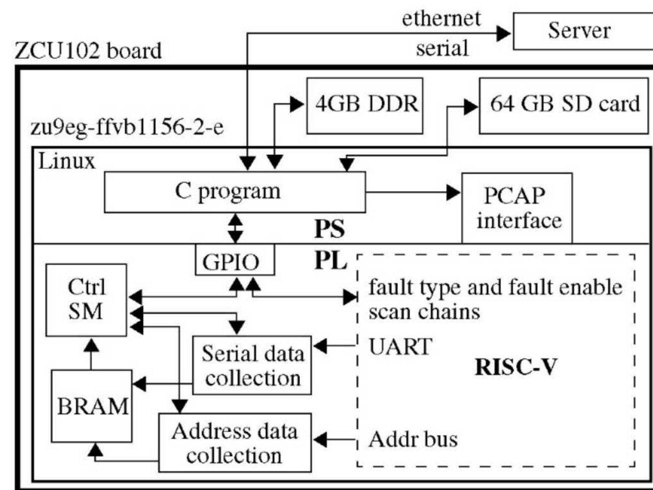
```
void fips ()
{
    int match;
    aes256_context ctx;
    uint8_t key[32]={0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,
                     0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,0x10,0x11,
                     0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1a,
                     0x1b,0x1c,0x1d,0x1e,0x1f};

    uint8_t i;
    uint8_t buf[16]={0xab,0xcd,0x22,0x33,0x44,0x55,0x66,0x77,
                     0x88,0x99,0xaa,0xbb,0xcc,0xdd,0xee,0xff};

    uint8_t chk[16]={0x8E,0xA2,0xB7,0xCA,0x51,0x67,0x45,0xBF,
                     0xEA,0xFC,0x49,0x90,0x4B,0x49,0x60,0x89};


    /* put a test vector */
    DUMP("txt: ", i, buf, sizeof(buf));
    DUMP("key: ", i, key, sizeof(key));
        print_uart("---\n",4);

    aes256_init(&ctx, key);
    aes256_encrypt_ecb(&ctx, buf);

    DUMP("enc: ", i, buf, sizeof(buf));
    DUMP("tst: ", i, chk, sizeof(chk));

        match = equality(buf, chk, 16);
        if (match == 1)
                print_uart("Match\n",6);

        if (match == 0)
                print_uart("Error  \n",8);

    aes256_done(&ctx);

} /* fips */
```

Saboteur Circuit



- Two modes: By-pass and fault insert
- Four fault types: Stuck-0, Stuck-1, delay and invert
- 85,714 copies in one scan chain, controlled by C program
- Supports any number (and fault type) of simultaneously inserted fault conditions
- We ran Rocket with AES encryption program 6.8 million times, testing:
  - Each of the 4 fault types
  - Fault combinations with 1 to 5 copies of each fault
  - Two plaintext-key encryptions
  - Under scrubbing and no scrubbing conditions

Testing Process



- Emulation Platform: Zynq UltraScale+ MPSoc with Linux/C program running on processor (PS) and RISC-V + state machines in programmable logic (PL)
- High speed GPIO PS-PL interface used for data transfer and control, e.g., 24 MHz PL clock rate with 32-bit transfers per AXI bus cycle
- Processor Configuration Access Port (PCAP) used to implement 'scrubbing' where entire PL side reprogrammed with fresh image after each test
- 64 KB of serial data and 64 32-bit address data values transferred after each test, at rate of approx. 2 (with scrubbing) and 4 (without scrubbing) tests/second
- Complex 'test complete' criteria implemented where serial port data collected continuously until 64 KB buffer fills or timeout occurs (some test durations in minute range)

Expected Output

```
<------ test 2 -------->          (SERIAL OVERFLOW? 0)¶
Boot . . .OK^M¶
txt: 00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff¶
key: 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11
12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f¶
---¶
enc: 8e a2 b7 ca 51 67 45 bf ea fc 49 90 4b 49 60 89¶
tst: 8e a2 b7 ca 51 67 45 bf ea fc 49 90 4b 49 60 89¶
Match¶
----¶
number of serial bytes: 288¶
number of address lines: 52147¶
1007FB80¶
1007FB90¶
1007FBA0¶
1007FBB0§
```

Continues with listing of the last 64 address changes

Serial data output

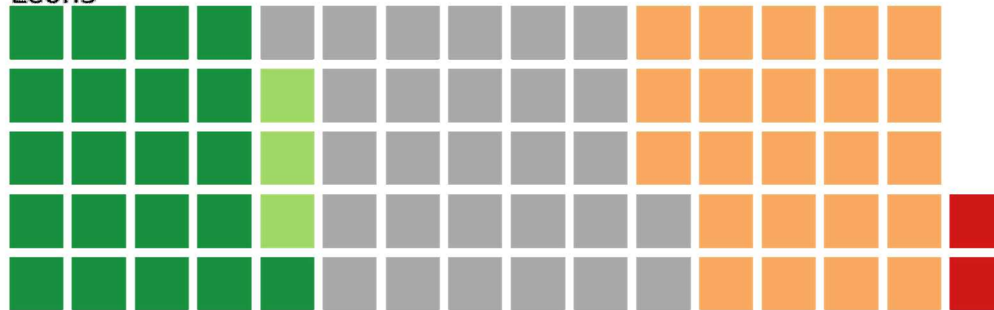
Address data output

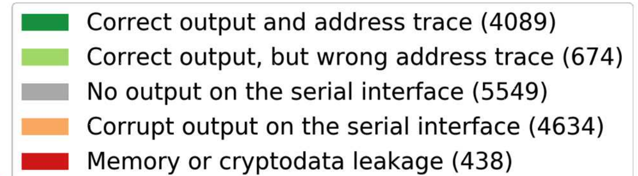
- Fault-free output prints Boot message, the plaintext and key, and then the computed and stored version of the ciphertext
- Rocket program compares the ciphertexts and prints 'Match' or 'MisMatch' to the serial port
- Our PS-side C program counts the number of bytes printed to serial port and the number of address line changes (fetched from a PL-side register)
- The last 64 addresses are then printed
- Faulty output varied widely, from no output to 64 KB bytes generated on serial port, with runtimes increasing to several minutes in some 'continuous reboot' cases

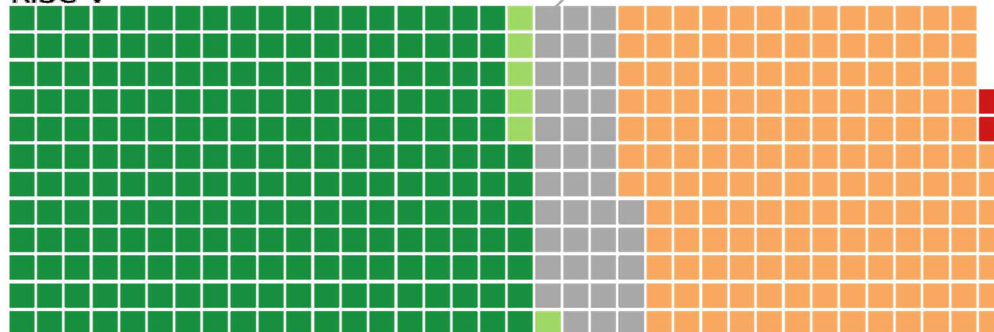
The Perfect Storm

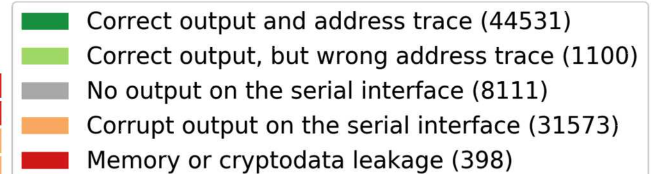
- Stock AES256 software implementation
- No software fault mitigations
- No hardware fault mitigations
- Debug information printed to serial port
- What could go wrong?

How does RISC-V compare to Leon3?





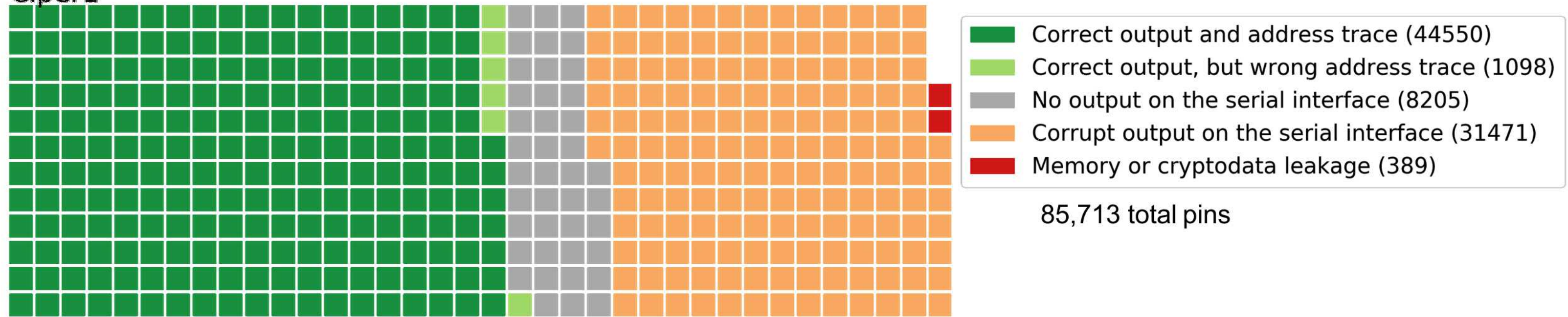




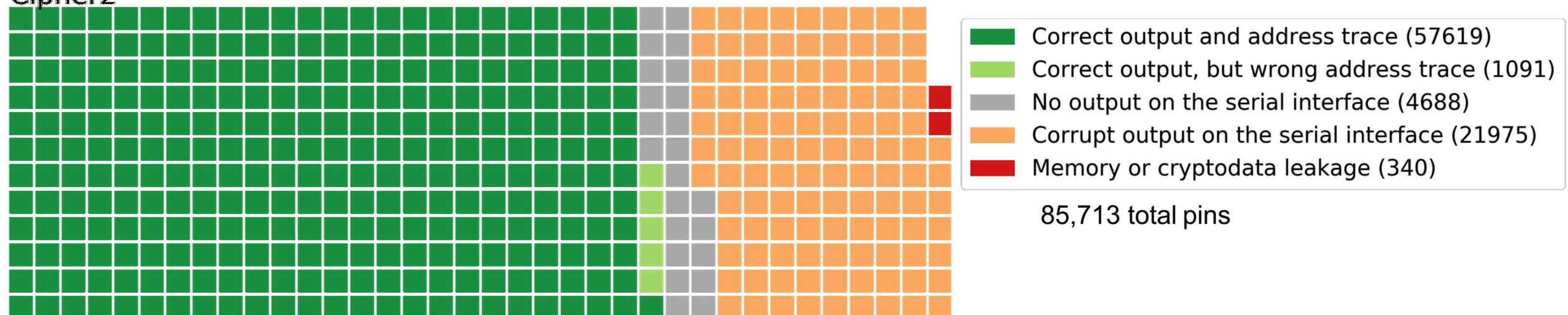
85,713 total pins

- RISC-V had about the same number of leakage susceptible pins as Leon3 (398 vs. 438).
- However, RISC-V had a lower percentage of susceptible pins than Leon3 (0.46% vs. 2.8%).
- About half the RISC-V pins were single-fault safe (for this application).

How does the choice of key/plaintext affect fault testing?

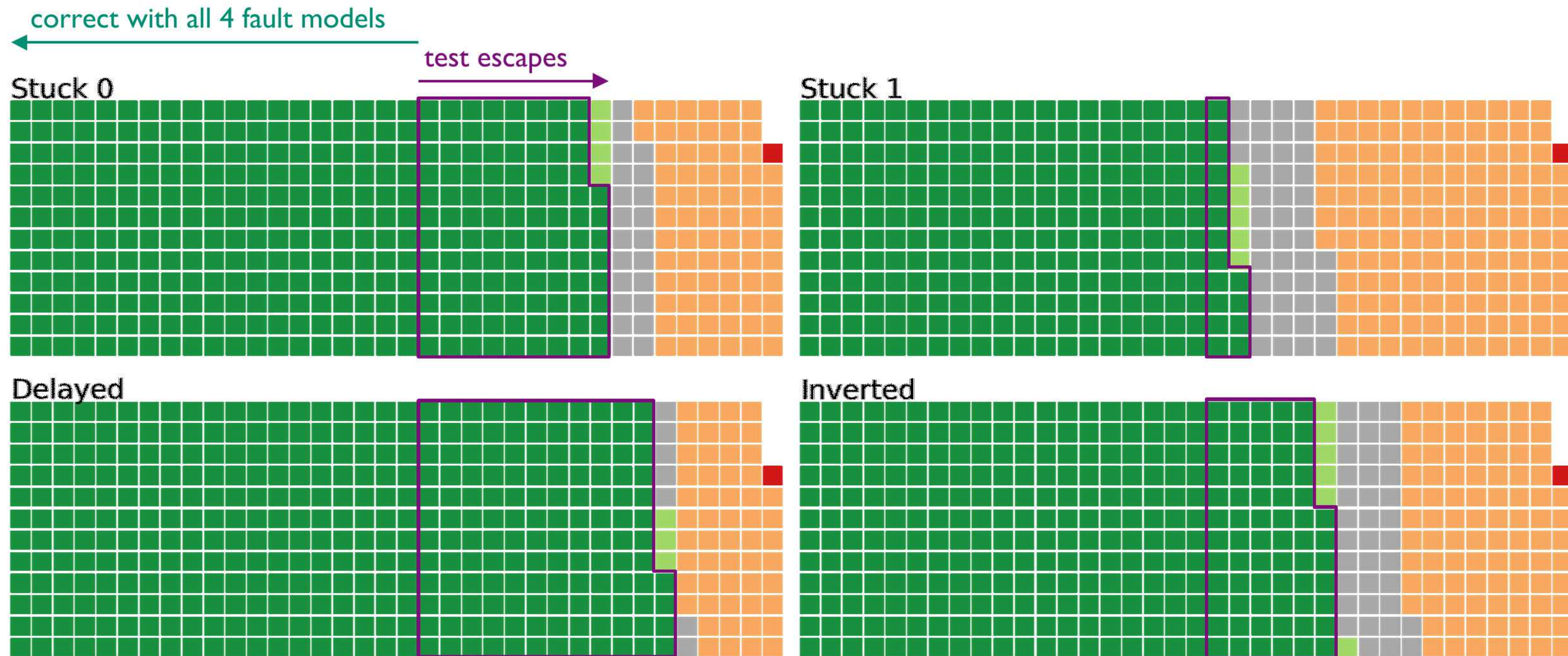


■ = 200 pins



- Cipher1 detected considerably more null and corrupt output conditions and slightly more leakage conditions.
- Fault testing with multiple key/plaintext values is imperative.

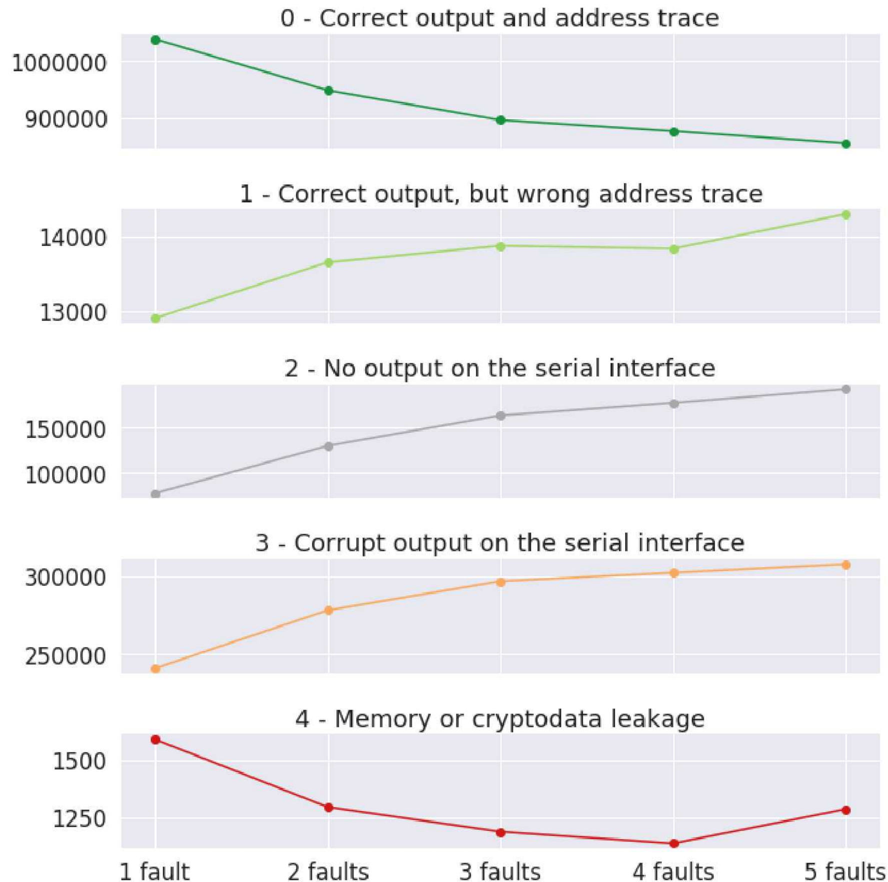
Are some fault models more effective at finding susceptible pins?



Susceptible pin behavior	Stuck 0	Stuck 1	Delayed	Inverted
1 - Correct output, but wrong address trace	11%	54%	9%	26%
2 - No output on the serial interface	9%	89%	0%	2%
3 - Corrupt output on the serial interface	6%	89%	3%	2%
4 - Memory or cryptodata leakage	20%	18%	42%	21%

- Each fault model identified susceptible pins not detected by any other model.
- Stuck-1 identified the most new susceptible pins across categories 1 through 3.
- Delay found twice as many new pins in category 4 (leakage) than any other model.

How do multiple faults affect behavior?

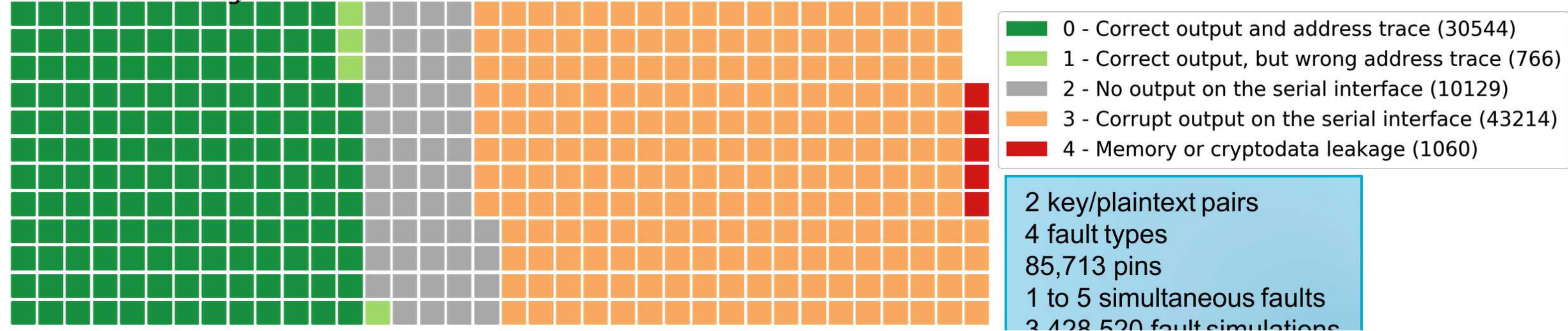


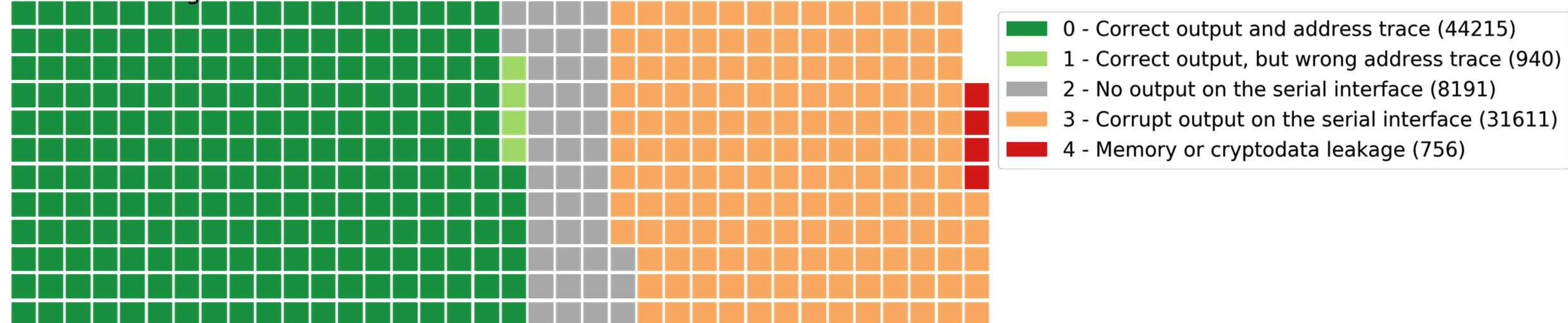
diagnosis	ser_type \ dataset	1 fault	2 faults	3 faults	4 faults	5 faults
0	perfect/match	1.03926e+06	948590	896032	876622	855356
1	perfect/match	12901	13652	13875	13837	14300
2	empty	77122	129795	163444	177437	192621
3	agree/error	8	4	--	--	--
	agree/match	618	905	867	833	792
	agree/none	18	37	21	27	19
	correct/error	564	359	227	214	178
	correct/match	474	491	488	511	500
	correct/none	1332	1306	812	738	637
	corrupt	13571	14808	14046	15077	15205
	corrupt/error	568	612	536	545	542
	corrupt/match	102	78	41	37	20
	disagree/error	31223	29150	27591	25843	24792
	disagree/match	214	215	180	131	114
	disagree/none	1285	985	1035	955	933
	long	3348	3428	3554	3836	4339
	match/error	52	40	28	16	16
	repeat/boot	994	1220	1348	1219	1220
	repeat/enc	--	--	--	4	2
	repeat/error	9	7	--	--	--
	repeat/match	147	196	187	178	172
	repeat/pattern	2776	3639	3815	3918	4051
	repeat/txt	200	238	243	212	182
	short	183029	220358	241850	248081	254131
4	leak/crypto	94	54	91	89	113
	leak/key	1383	1153	1081	1029	1162
	leak/txt	112	88	16	19	11

6,857,040 total fault simulations

- Not surprisingly, desirable behavior decreases and undesirable behavior increases as more simultaneous faults are added.
- Leakage decreases to a point, then upticks after 4 or 5 simultaneous faults.
- Within these categories, some novel behavior was seen only after 4 simultaneous faults.

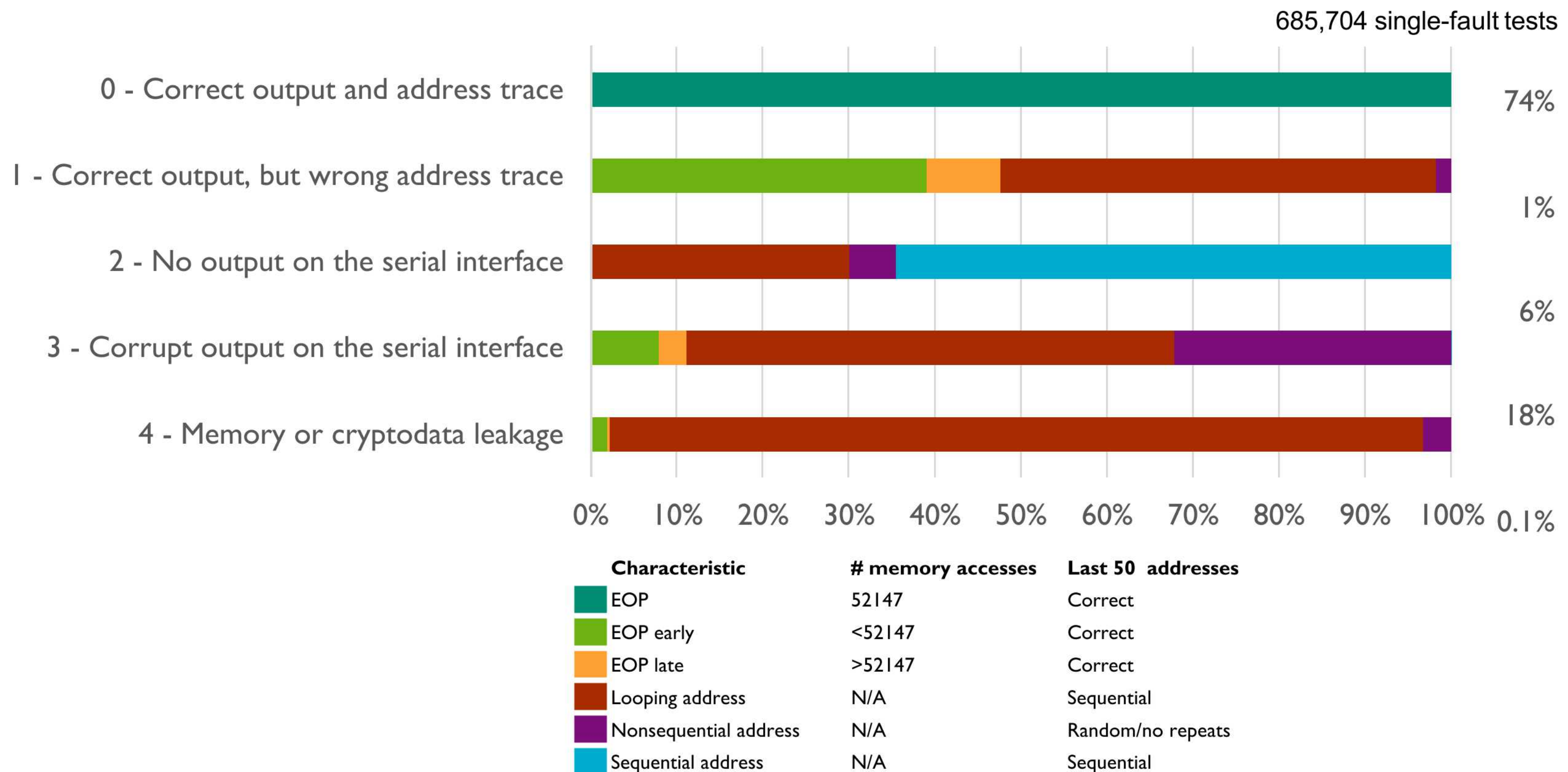
Does scrubbing memory improve fault performance?





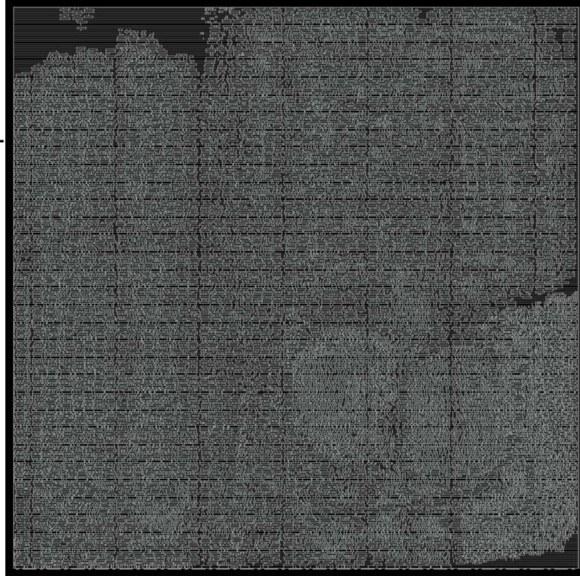
- Scrubbing memory after each encrypt operation was somewhat effective in reducing null output, corrupt output, and leakage.
- Additional mitigations are necessary to ensure a secure, robust system.

Is the address trace predictive of program behavior?



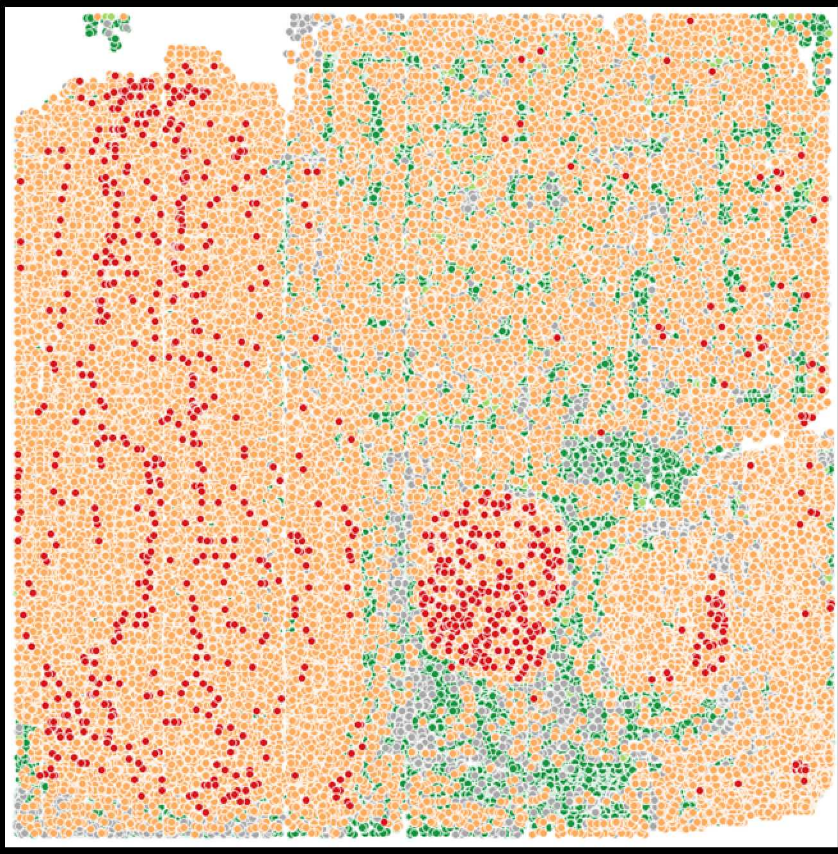
- Surprisingly, about 2500 of the tests that terminated early still produced the correct output!
- Most of the leakage behavior is associated with a looping address.
- Most of the null output behavior is associated with a runaway sequential address.
- However, the address categories are generally not predictive of program behavior.

Where are the faulty pins located on the ASIC?

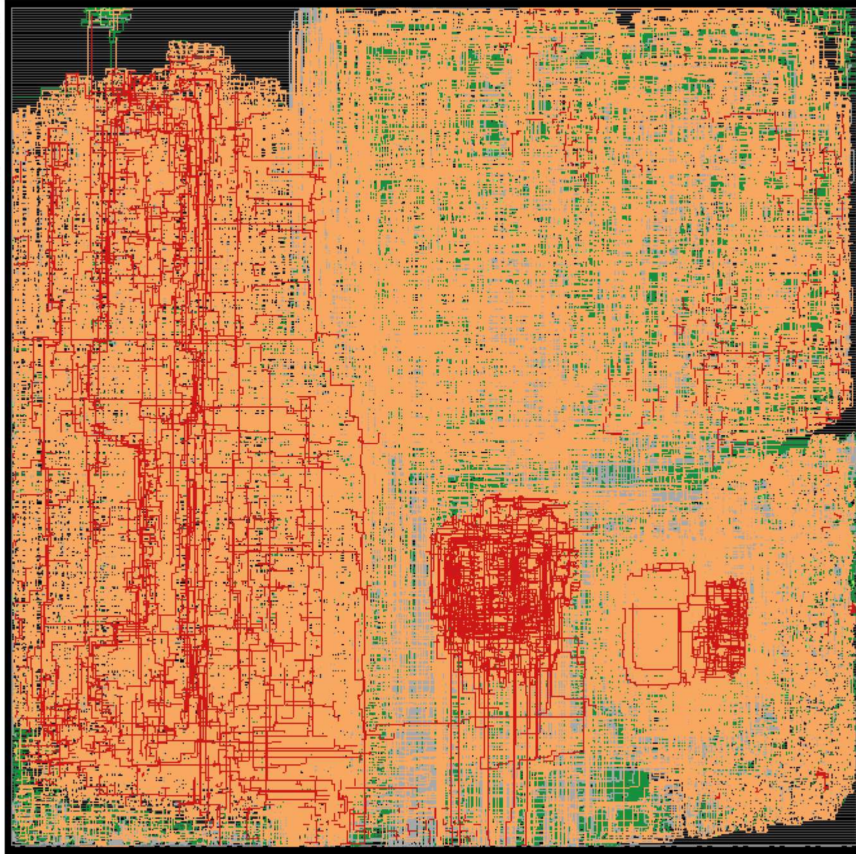


Cell placement

Hierarchical placement



Susceptible pins



Connected nets

Summary

Fault Testing Architecture

- RISC-V architecture had more than 85K fault injection sites, requiring an accelerated fault testing and data collection approach.
- Advanced features of FPGA emulation platform leveraged for acceleration, including use of dynamic reconfiguration and processor-side/programmable logic side high speed GPIO
- Emulation platform enabled 'scrubbing' between each fault experiment, and additional clarity on level of fault propagation across fault tests

Fault Testing Results

- Tested 4 fault types, 1-5 faults inserted simultaneously, two AES key-plaintext encryptions, with and without scrubbing, more than 6.8 million fault testing experiments
- The Rocket implementation was less susceptible to data leakage from hardware faults than the Leon3 core tested last yea
- Scrubbing memory between operations was marginally effective as a solo mitigation strategy

Next Steps

- Fault test the RISC-V running a Pseudo Random Number Generator (PRNG)
- Test the effectiveness of common hardware and software mitigations
- Develop hypotheses on why particular failures are causing security degradation
- Develop low-overhead checking logic that can be applied broadly across common RISC-V microprocessor implementations.