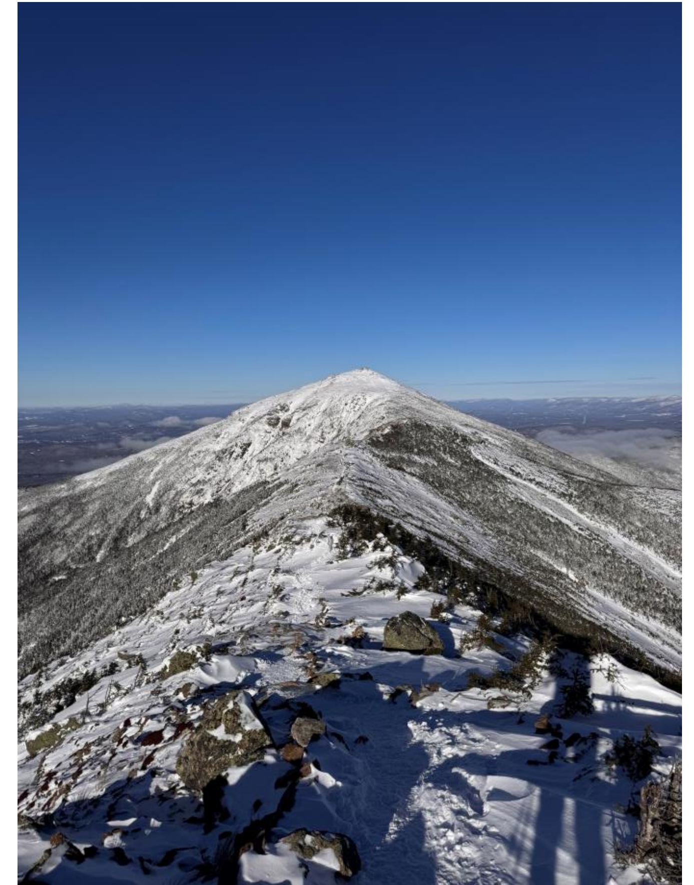
Fuzzing Zephyr Apps – Struggles of Dynamic Analysis on Embedded Applications

Jayashree Srinivasan
Senior Engineer, Research Science & Engineering
Product Security Team

About me

- Embedded Security Enthusiast
- Senior Engineer in Product Security Team, Analog Devices
  - Security solutions for ADI products
  - Work with devices based on Arm TrustZone-M extensions and custom IP
    - Trusted Firmware-M
    - Zephyr
    - MCUboot
- Topic of research during Masters at Purdue
  - Dynamic Analysis on Embedded Applications in a HW agnostic way
- Outside work, I like hiking and learning Classical music



Franconia Ridge, White Mountains, NH

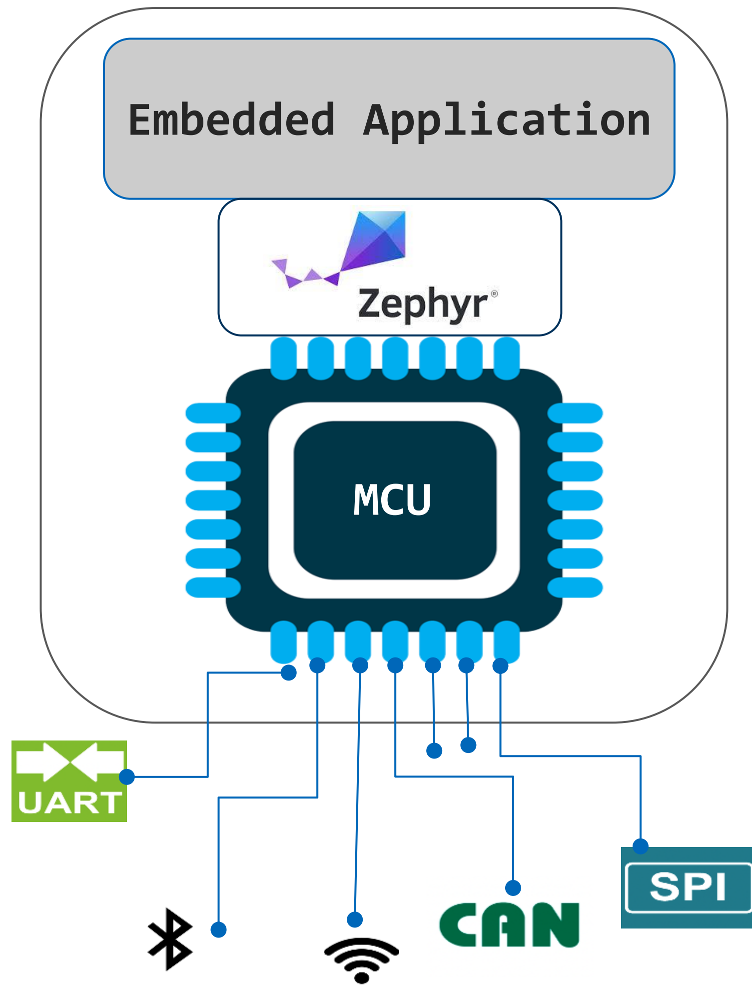
©2026 Analog Devices, Inc. All Rights Reserved.

ANALOG DEVICES

Agenda

- What is Fuzzing?
  - Why do we need it?
  - Overview
- Fuzzing Embedded Apps
  - Motivation
  - Challenges
  - Solutions
- Fuzzing in Zephyr
  - Current State
  - Need for AFL++
- Conclusion
- Q&A

Introduction



- Embedded Systems are widely used in safety-critical applications.
- They are deployed in a connected environment interacting with the external world.
- Security analysis of these systems is crucial.

Let’s do Fuzzing!

©2026 Analog Devices, Inc. All Rights Reserved.

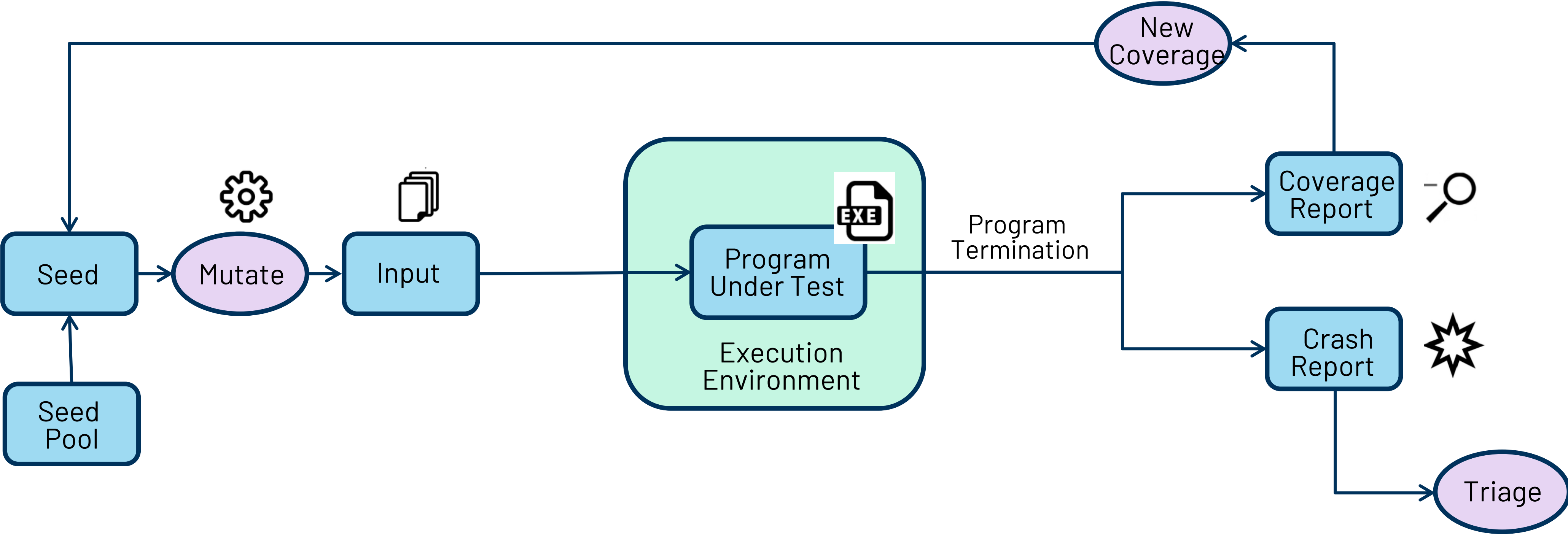
What is Fuzzing?

What is Fuzzing?

- Run-time Software Testing Technique
- Feed random, malformed, or unexpected input data to a program
- Monitor for crashes, memory leaks, or unexpected behavior

©2026 Analog Devices, Inc. All Rights Reserved.

What is Fuzzing?



©2026 Analog Devices, Inc. All Rights Reserved.

Why do Fuzzing?

- Comparing against **Static Analysis**
  - **(-)** Limited coverage – only tests executed code paths
  - **(-)** Slower – requires building, running, and testing
  - **(+)** Lower false positive rate – if it crashes, it's a real bug
  - **(+)** Detects issues that appear under specific conditions

©2026 Analog Devices, Inc. All Rights Reserved.

Fuzzing Embedded Applications

Motivation

STM32CubeF4 v1.28.0
Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_rcc.c

```
903      case RCC_CFGR_SWS_PLL:  /* PLL used as system clock  source */
904      {
905        /* PLL_VCO = (HSE_VALUE or HSI_VALUE / PLLM) * PLLN
906        SYSCLK = PLL_VCO / PLLP */
907        pllm = RCC->PLLCFGR & RCC_PLLCFGR_PLLM;
908        if(__HAL_RCC_GET_PLL_OSCSOURCE() != RCC_PLLSOURCE_HSI)
909        {
910          /* HSE used as PLL clock source */
911          pllvco = (uint32_t) ((((uint64_t) HSE_VALUE * ((uint64_t) ((RCC->PLLCFGR & RCC_PLLCFGR_PLLN) >> RCC_PLLCFGR_PLLN_Pos)))) / (uint64_t)pllm);
912        }
913        else
914        {
915          /* HSI used as PLL clock source */
916          pllvco = (uint32_t) ((((uint64_t) HSI_VALUE * ((uint64_t) ((RCC->PLLCFGR & RCC_PLLCFGR_PLLN) >> RCC_PLLCFGR_PLLN_Pos)))) / (uint64_t)pllm);
917        }
918        pllp = ((((RCC->PLLCFGR & RCC_PLLCFGR_PLLP) >> RCC_PLLCFGR_PLLP_Pos) + 1U) *2U);
```

The pllm value is directly read from Hardware registers and is used as divisor without any checks

[J. Srinivasan, S. R. Tanksalkar, P. C. Amusuo, J. C. Davis, and A. Machiry, "Towards rehosting embedded applications as Linux applications", in IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2023.]

©2026 Analog Devices, Inc. All Rights Reserved.

Motivation

Zephyr 2.4.0
zephyr/subsys/net/ip/6lo.c

```
1358                    NET_DBG("Not enough tailroom. Get new fragment");
1359                    cursor =  pkt->buffer->data;
1360                    frag = net_pkt_get_frag(pkt, NET_6LO_RX_PKT_TIMEOUT);
1361                    if (!frag) {
1362                            NET_ERR("Can't get frag for uncompression");
1363                            return false;
1364                    }
1365
1366                    net_buf_pull(pkt->buffer, compressed_hdr_size);
1367                    net_buf_add(frag, nhc ? NET_IPV6UDPH_LEN : NET_IPV6H_LEN);
1368            }
```

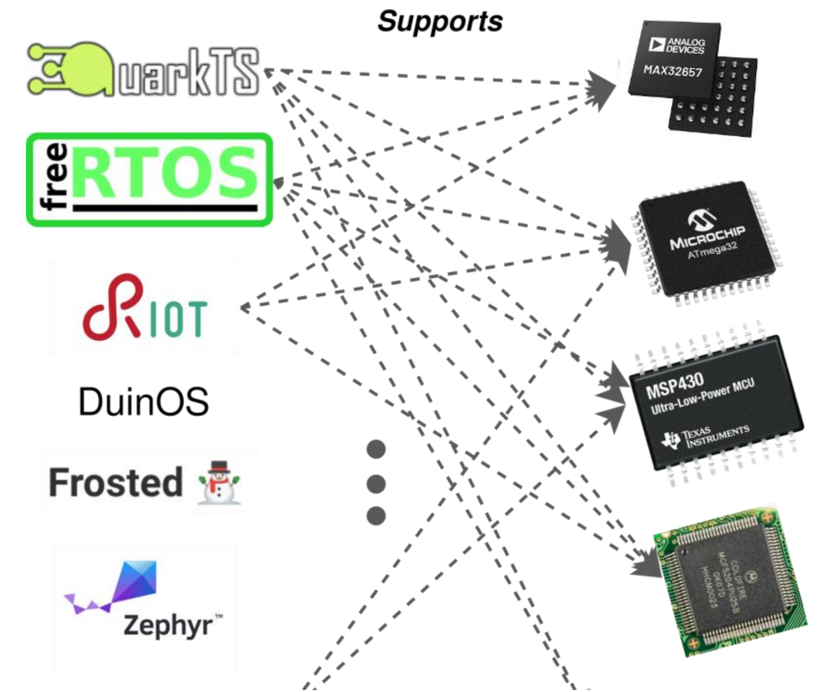
Missing checks on network packet size in uncompress_IPHC_header leads to an integer underflow, resulting in corrupted net_buf bounds (CVE-2021-3323)

[T. Scharnowski, N. Bars, M. Schloegel, E. Gustafson, M. Muench, G. Vigna, C. Kruegel, T. Holz, and A. Abbasi, "Fuzzware: Using precise MMIO modeling for effective firmware fuzzing," in USENIX Security Symposium, 2022.]

©2026 Analog Devices, Inc. All Rights Reserved.

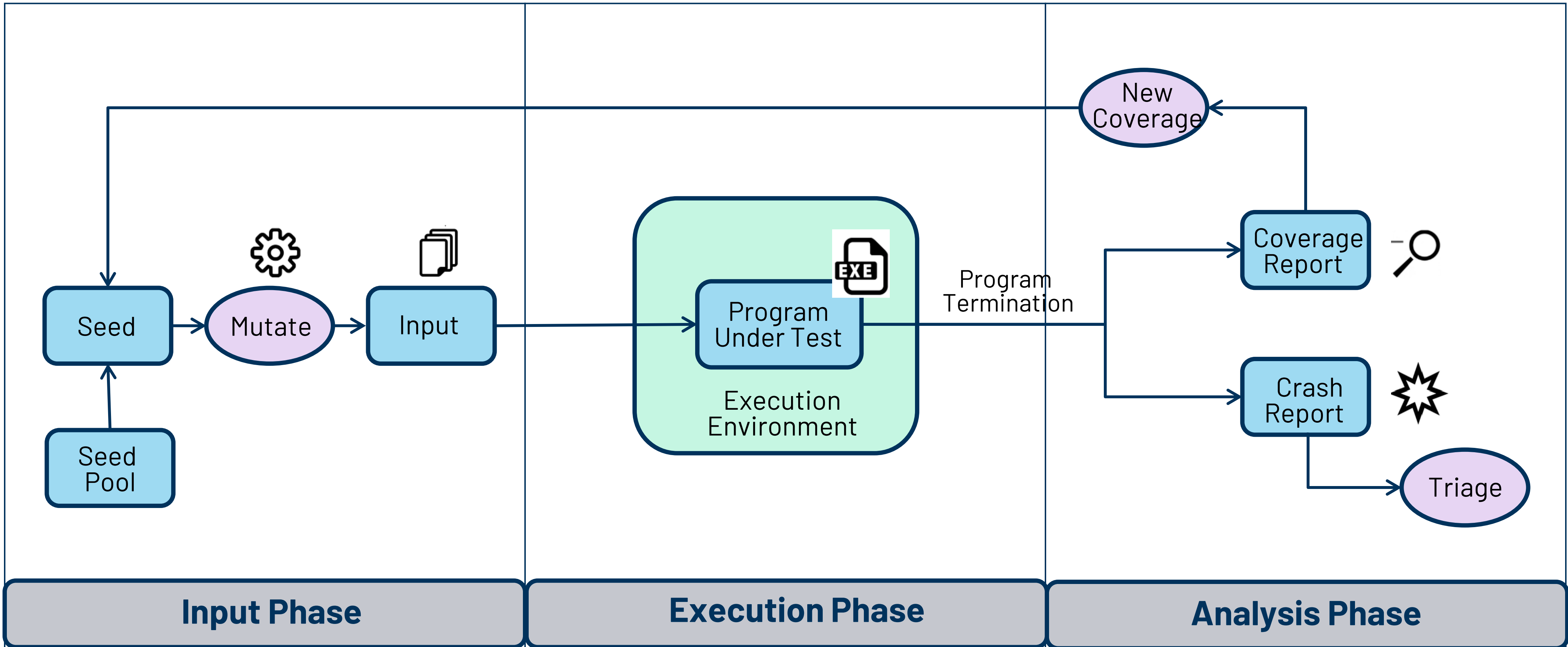
Fuzzing Embedded Applications - Challenges

- The curse of extreme diversity
  - Different architectures
  - Very less standardization across stacks



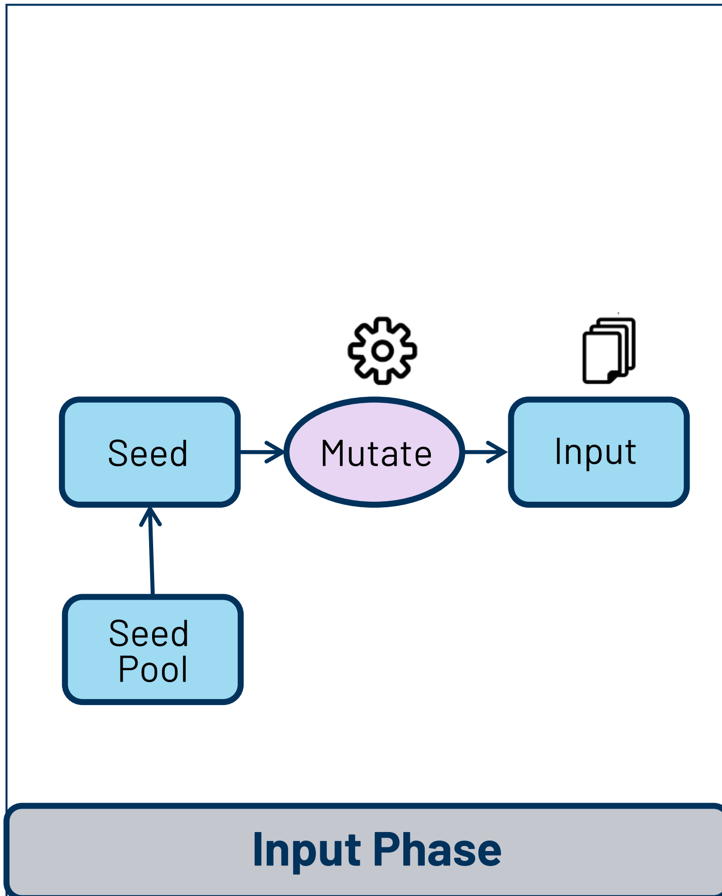
 ©2026 Analog Devices, Inc. All Rights Reserved.

Challenges



©2026 Analog Devices, Inc. All Rights Reserved.

Challenges – Input Phase

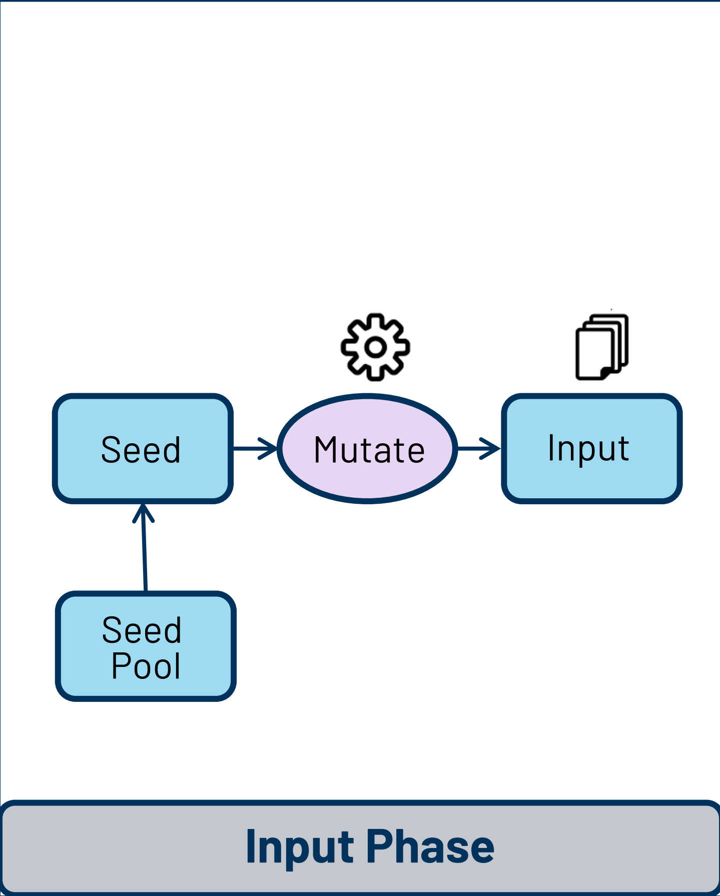


- Highly Specialized Input

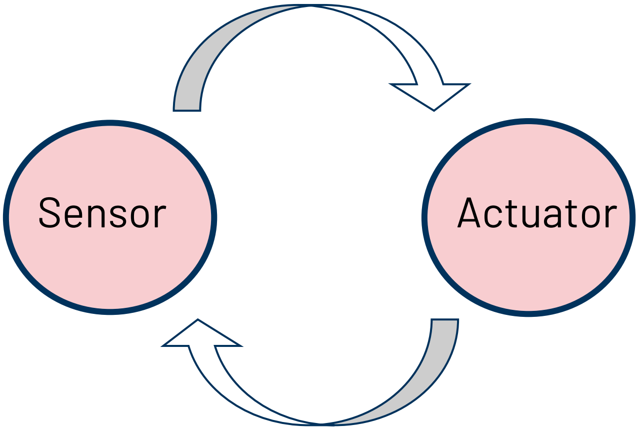
```
3920 ADI_HAL_FORCE_INLINE uint8_t adi_hal_spi_01_get_fifo_stat_rx(uintptr_t base_addr)
3921 {
3922     uint16_t reg_val;
3923     uint16_t raw;
3924
3925     reg_val = mmio_read16(base_addr + SPI_01_REG_FIFO_STAT_OFFSET);
3926
3927     raw = (reg_val & (uint16_t)SPI_01_BM_FIFO_STAT_RX) >>
3928                     (uint16_t)SPI_01_BP_FIFO_STAT_RX;
3929     return (uint8_t)raw;
3930 }
```


©2026 Analog Devices, Inc. All Rights Reserved.

Challenges – Input Phase

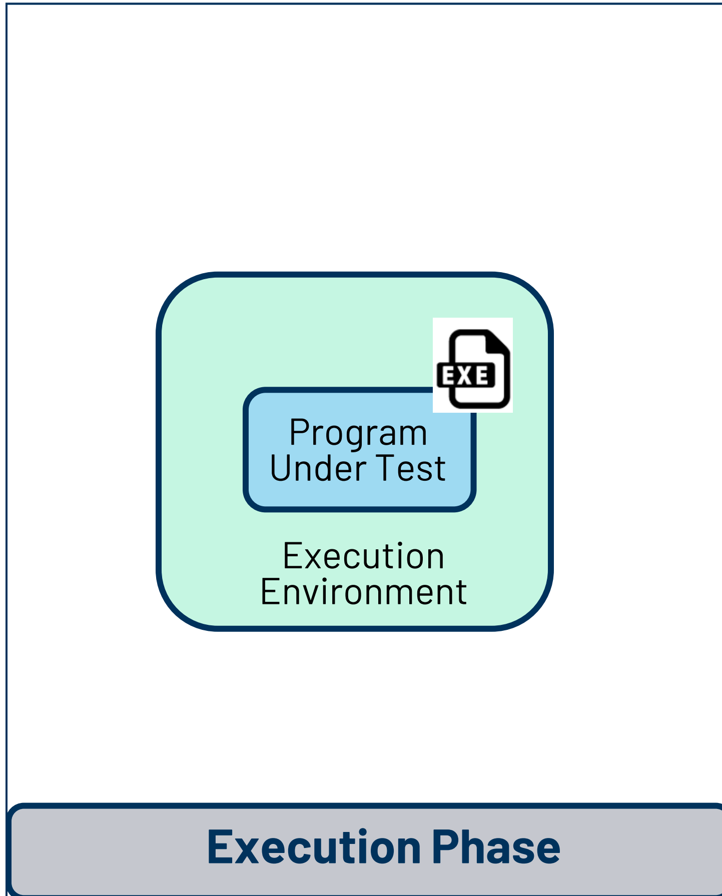


- Physical/ environmental Dependencies



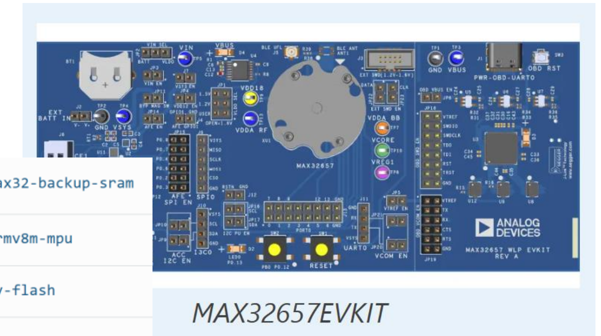
©2026 Analog Devices, Inc. All Rights Reserved.

Challenges – Execution Phase



- Highly hardware dependent – sensors, storage devices, etc.

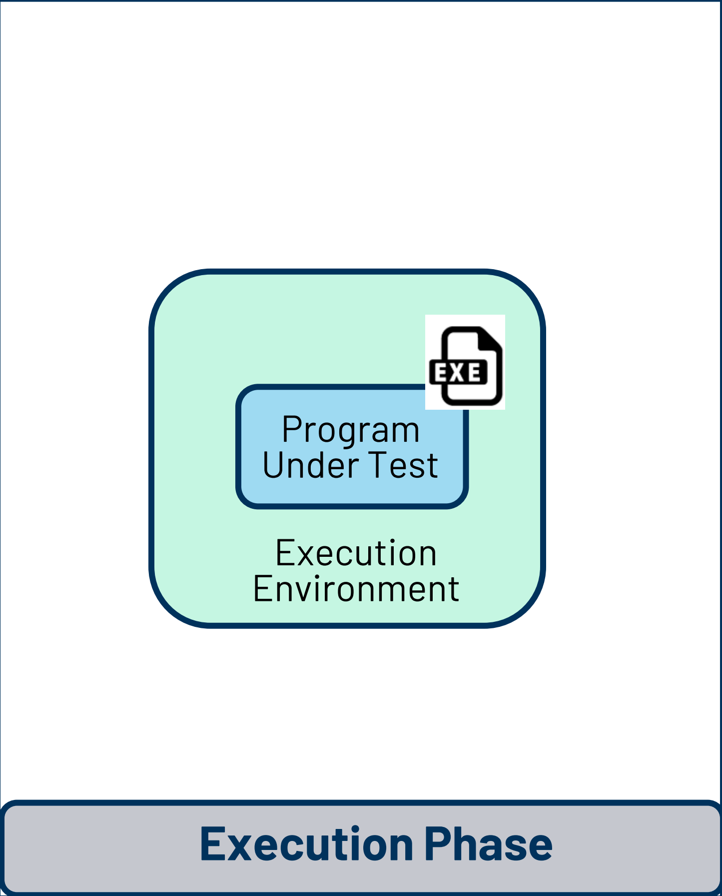
Memory controller	on-chip	ADI MAX32 Backup SRAM	× 5	adi,max32-backup-sram
MMU / MPU	on-chip	ARMv8-M MPU (Memory Protection Unit)	× 1	arm,armv8m-mpu
MTD	on-chip	Flash node	× 1	soc-nv-flash
	on-board	Fixed partitions of a flash (or other non-volatile storage) memory	× 1	fixed-partitions
Pin control	on-chip	MAX32 Pin Controller	× 1	adi,max32-pinctrl
PWM	on-chip	ADI MAX32 PWM	× 6	adi,max32-pwm
RNG	on-chip	ADI MAX32XXX TRNG	× 1	adi,max32-trng
Sensors	on-board	ADXL367 3-axis nanopower accelerometer, accessed through I2C bus	× 1	adi,adxl367
Serial	[illegible]	[illegible]	[illegible]	[illegible]



MAX32657EVKIT

©2026 Analog Devices, Inc. All Rights Reserved.

Challenges – Execution Phase

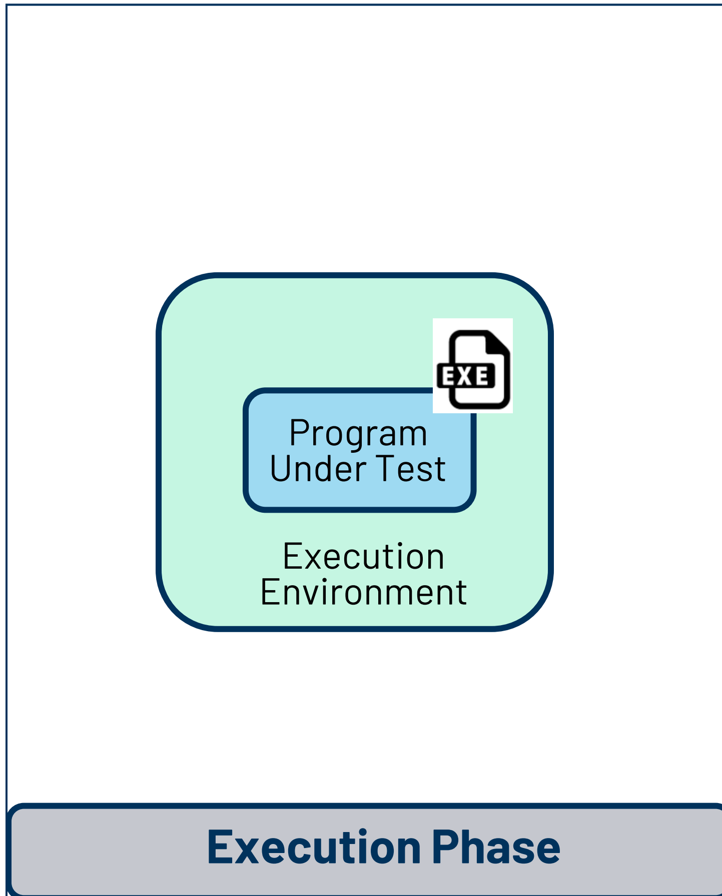


- Instrumentation Difficulties

```
zephyr
  cmake
    toolchain
      arcmwdt
      armclang
      cross-compile
      espressif
      gnuarmemb
      host
      iar
      llvm
      oneApi
      xcc
      xt-clang
      zephyr
```

 ©2026 Analog Devices, Inc. All Rights Reserved.

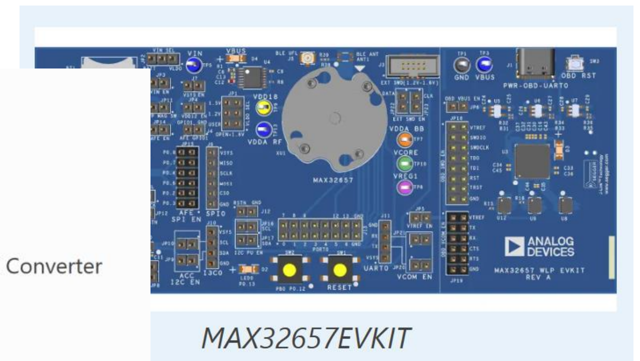
Challenges – Execution Phase



- Limited resources – Memory, Compute etc.

Hardware

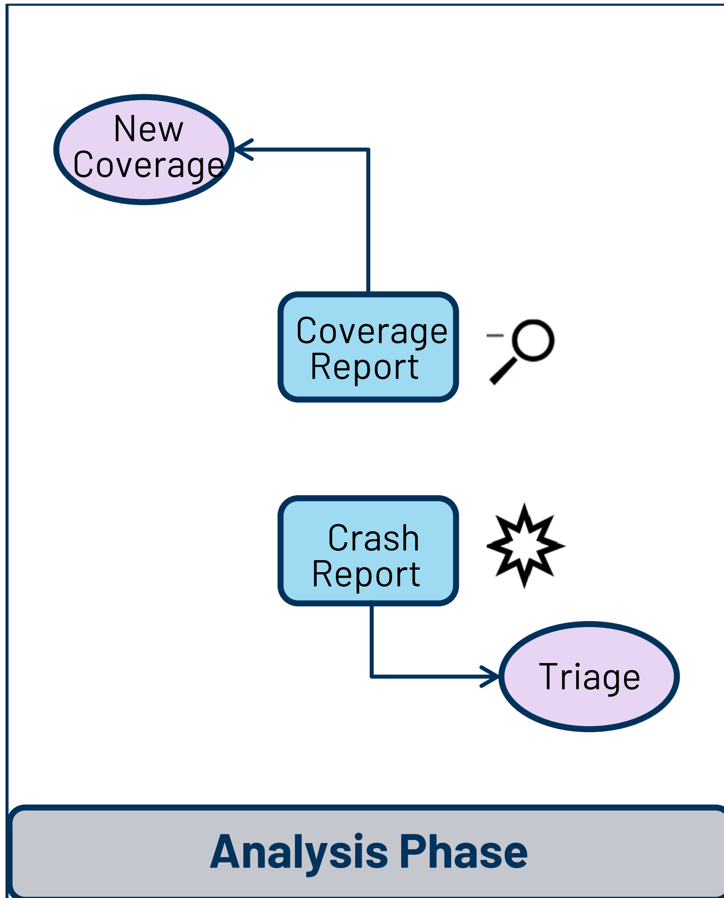
- MAX32657 MCU:
  - Arm Cortex-M33 CPU with TrustZone® and FPU
  - 1.2V to 1.6V Input Range for Integrated Boost DC-DC Converter
  - 50MHz Low Power Oscillator
  - External Crystal Support
    - 32MHz required for BLE
  - 1MB Internal Flash with ECC
  - 256kB Internal SRAM
  - 8kB Cache
  - 32.768kHz RTC external crystal



MAX32657EVKIT

©2026 Analog Devices, Inc. All Rights Reserved.

Challenges – Analysis Phase

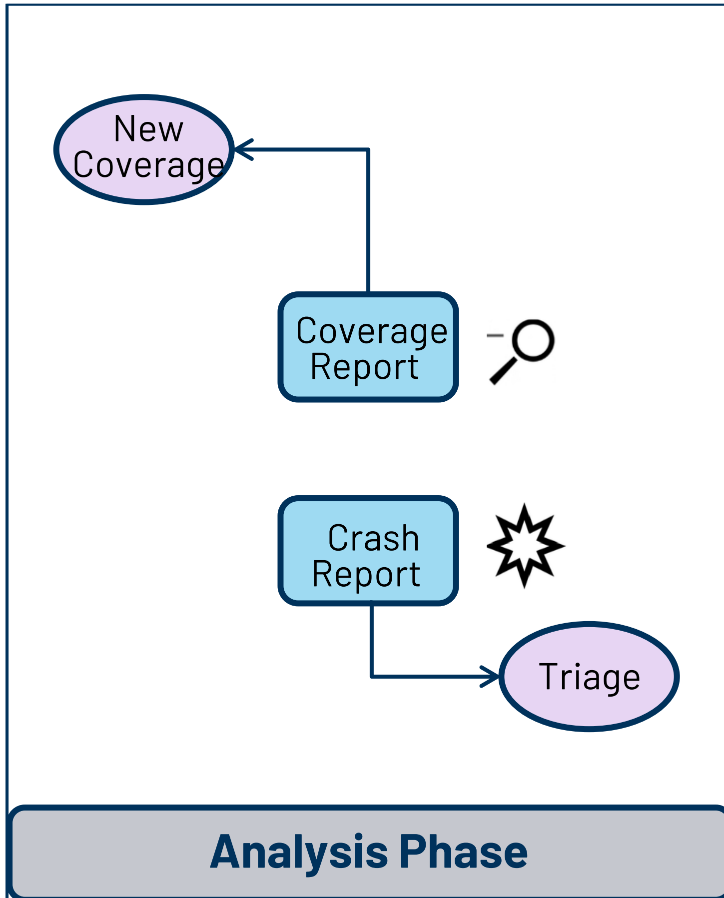


- Crash Detection

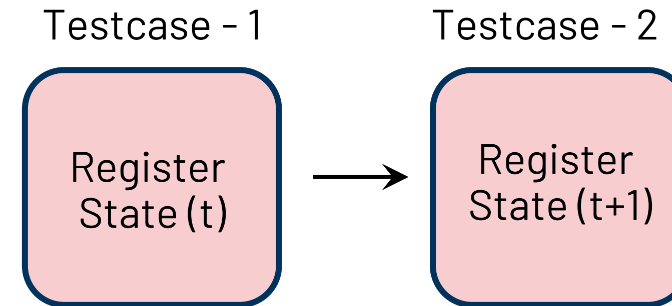
```
while (1) {
        . . .
}
```

 ©2026 Analog Devices, Inc. All Rights Reserved.

Challenges – Analysis Phase

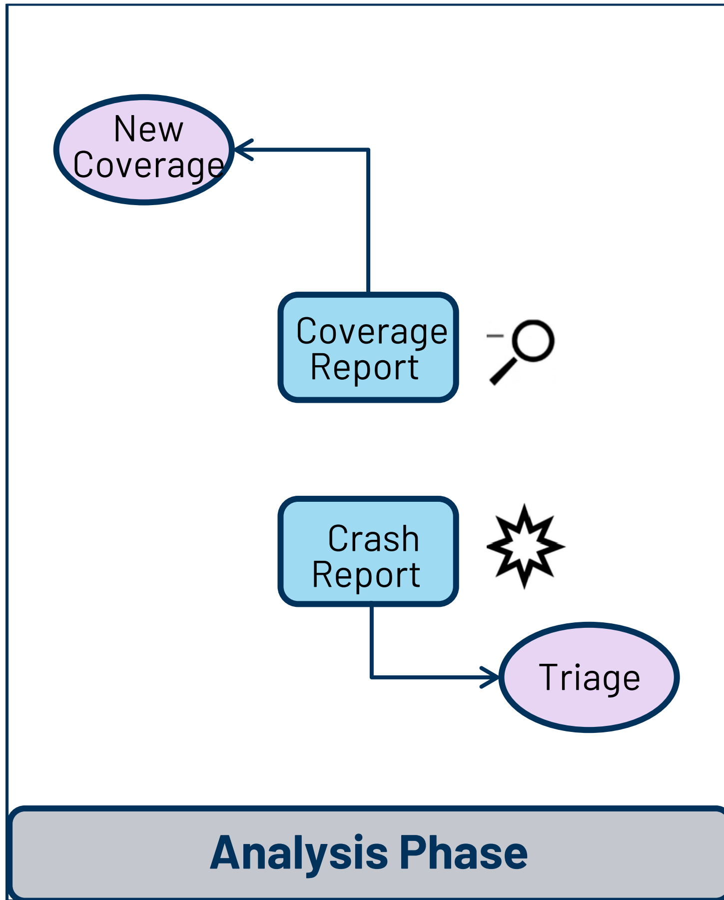


- State/ Persistence Issues

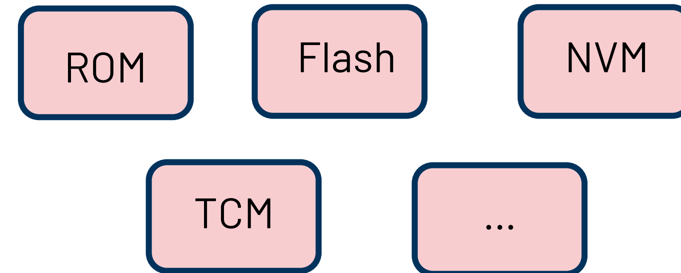


©2026 Analog Devices, Inc. All Rights Reserved.

Challenges – Analysis Phase

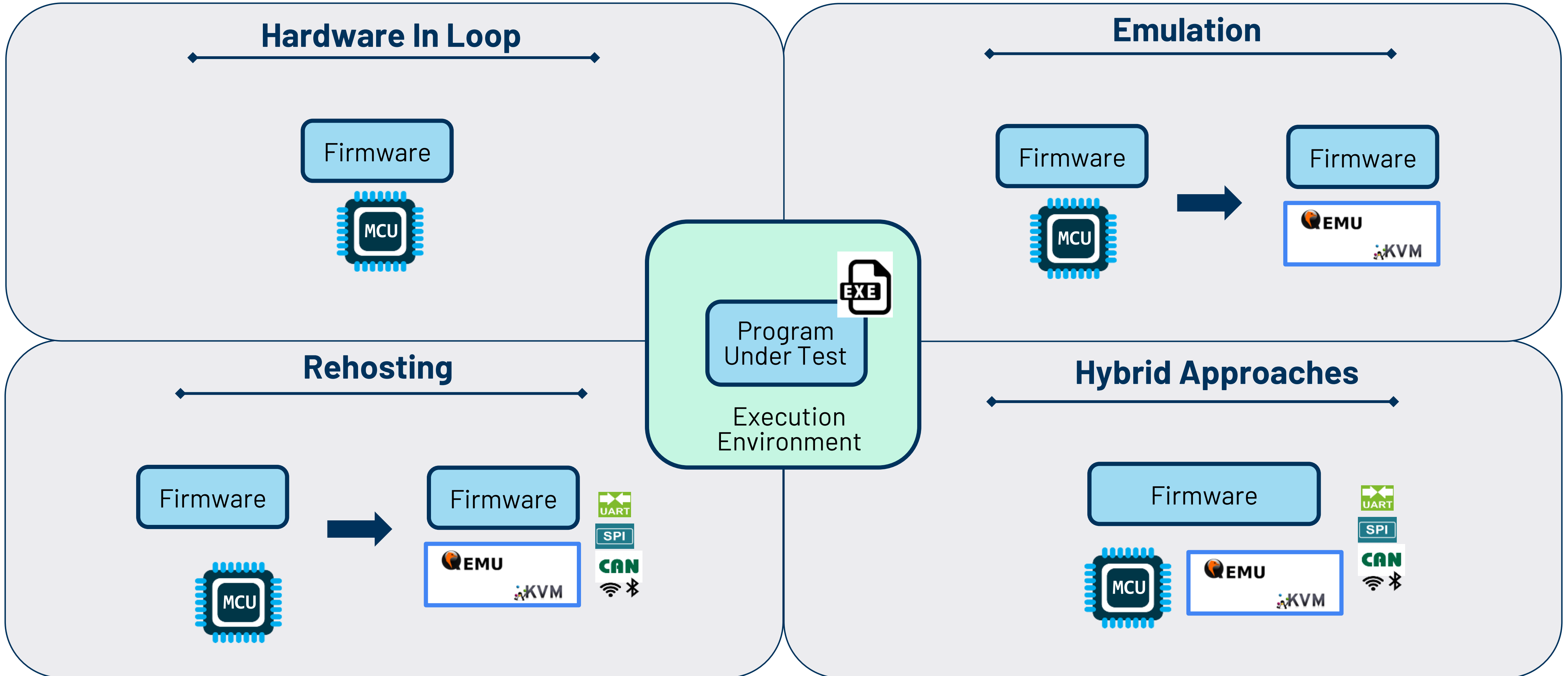


- FW Extraction and Analysis



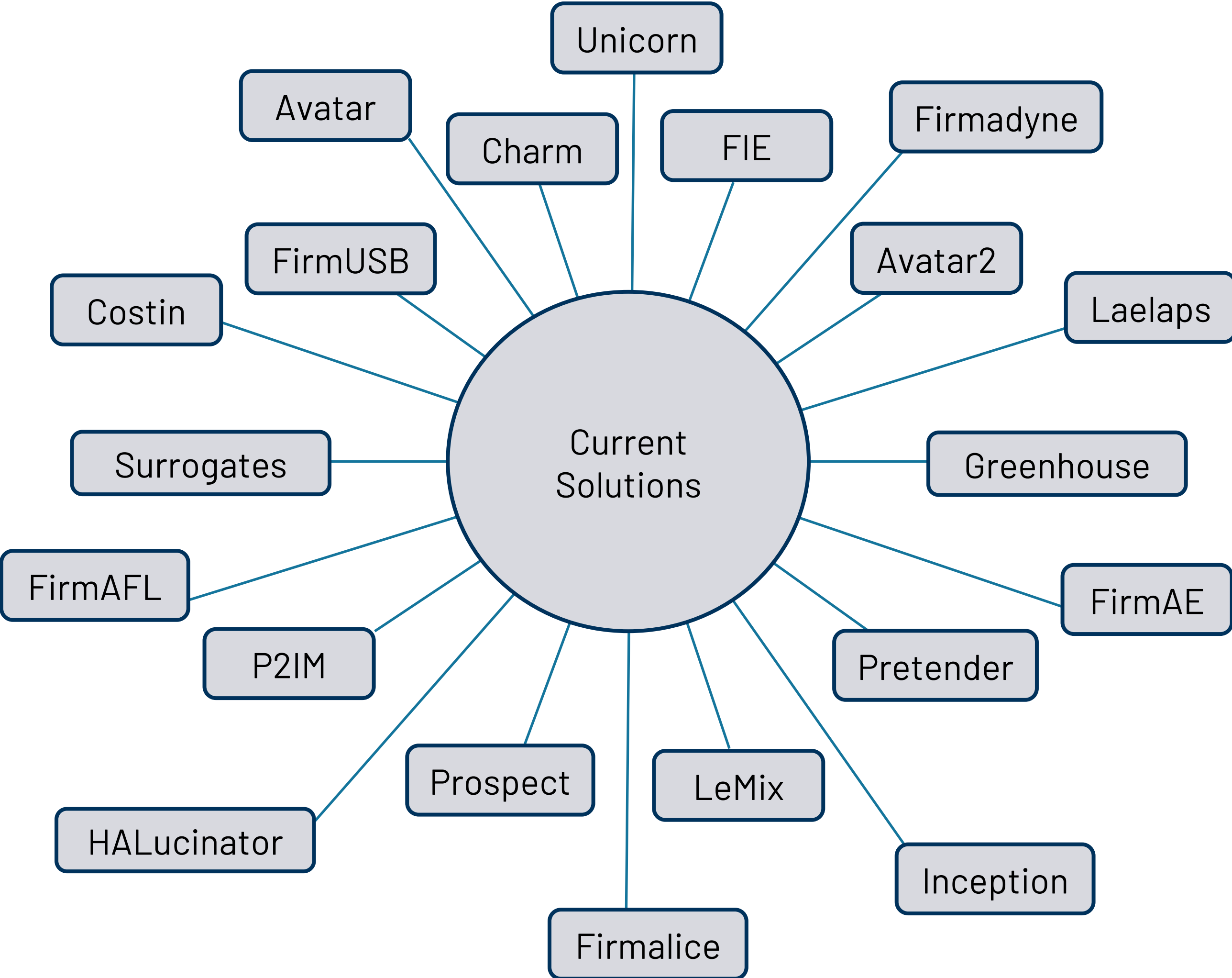
©2026 Analog Devices, Inc. All Rights Reserved.

Solutions



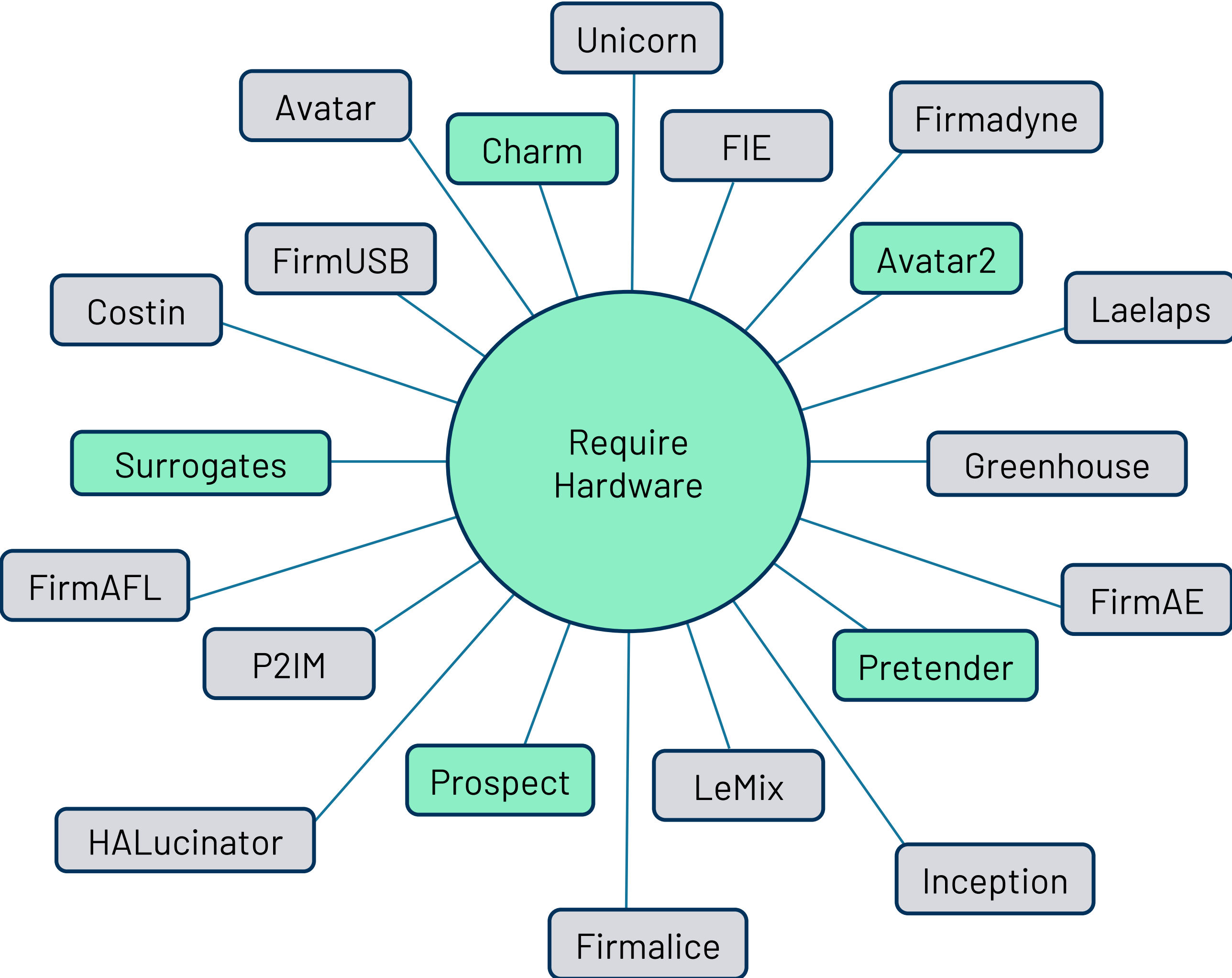
©2026 Analog Devices, Inc. All Rights Reserved.

Solutions – Academic Research



©2026 Analog Devices, Inc. All Rights Reserved.

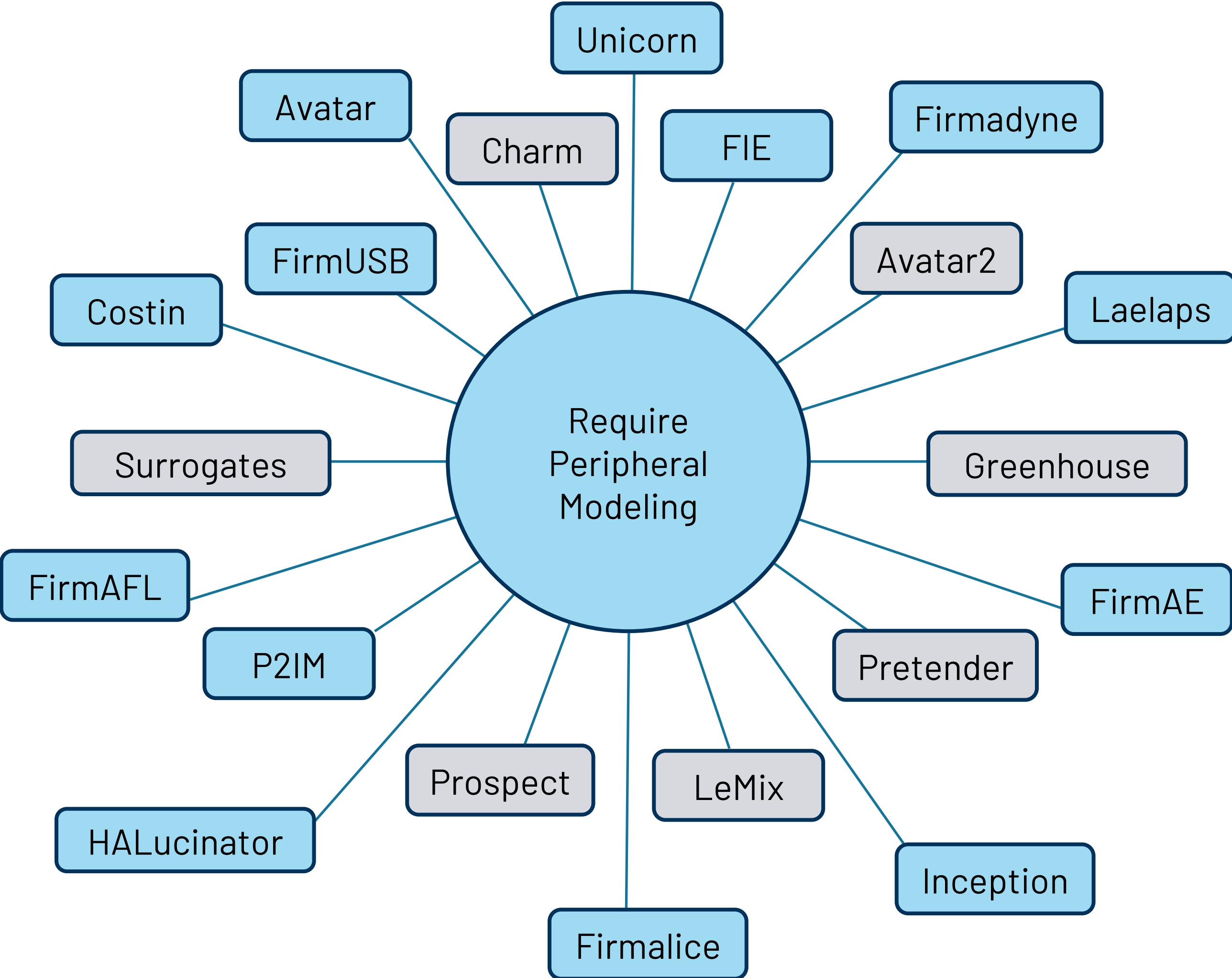
Solutions – Academic Research



©2026 Analog Devices, Inc. All Rights Reserved.

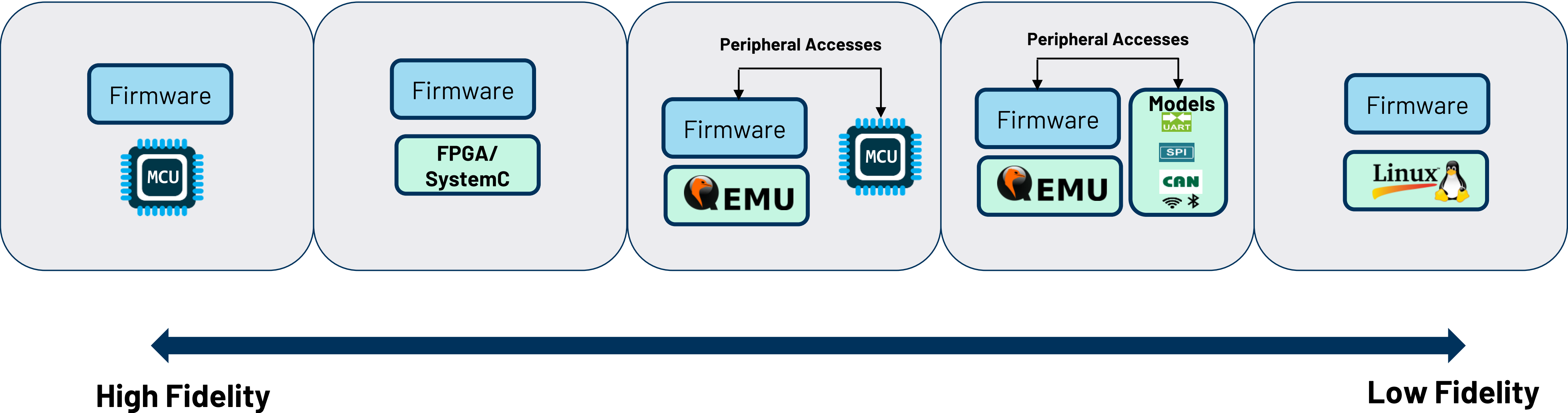
ANALOG DEVICES

Solutions – Academic Research



©2026 Analog Devices, Inc. All Rights Reserved.

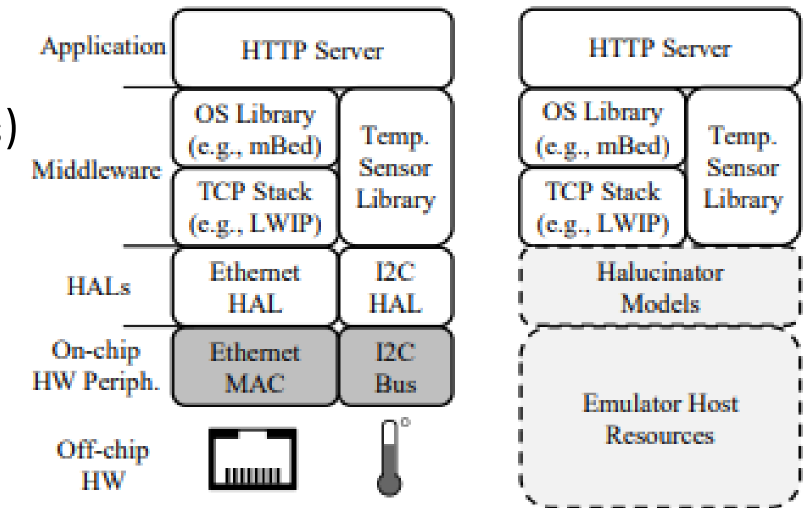
Solutions – More on Rehosting



©2026 Analog Devices, Inc. All Rights Reserved.

Solutions – HALucinator

- Firmware usually uses a HAL library that provides standard API to access Hardware
- HALucinator replace HAL functions with software models (Python handlers)
- Fuzzer input is a shared byte stream consumed by all HAL I/O handlers
- 16 firmware samples rehosted across 3 HAL libraries (STM32, NXP, Atmel ASF)
- Found real vulnerabilities in network-facing firmware

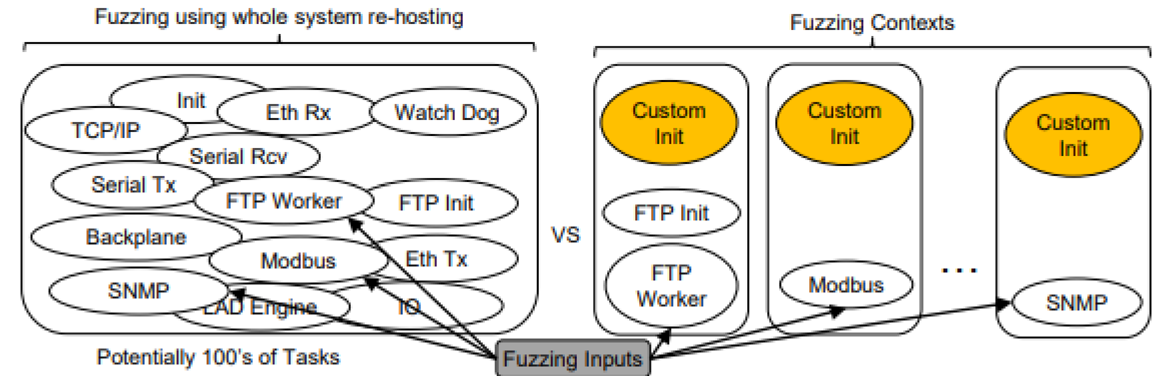


[A. A. Clements, E. Gustafson, T. Scharnowski, P. Grosen, D. Fritz, C. Kruegel, G. Vigna, S. Bagchi, and M. Payer, "Halucinator: Firmware re-hosting through abstraction layer emulation," in USENIX Security, 2020.]

©2026 Analog Devices, Inc. All Rights Reserved.

Solutions – RT-Fuzzer

- RTOS-based embedded systems run hundreds of concurrent tasks with asynchronous execution
- Whole-system rehosting is hard — complex hardware init, non-deterministic task scheduling breaks coverage-guided fuzzing



- Fuzz each task independently in its own isolated "fuzzing context" instead of rehosting the whole system
- Tested on Schneider Electric Modicon M340 PLC (commercial) and Apache NuttX FTP server (open-source); Found 3 real-world CVEs

[A. A. Clements , A. Rivera , R. Jiayang, K. Levchenko, R. Kennell, and G. Ciocarlie, "RT-Fuzzer: Task Driven Fuzzing of Real Time Operating System Firmware," in Workshop on Binary Analysis Research (BAR), 2026.]

©2026 Analog Devices, Inc. All Rights Reserved.

Fuzzing Zephyr Applications

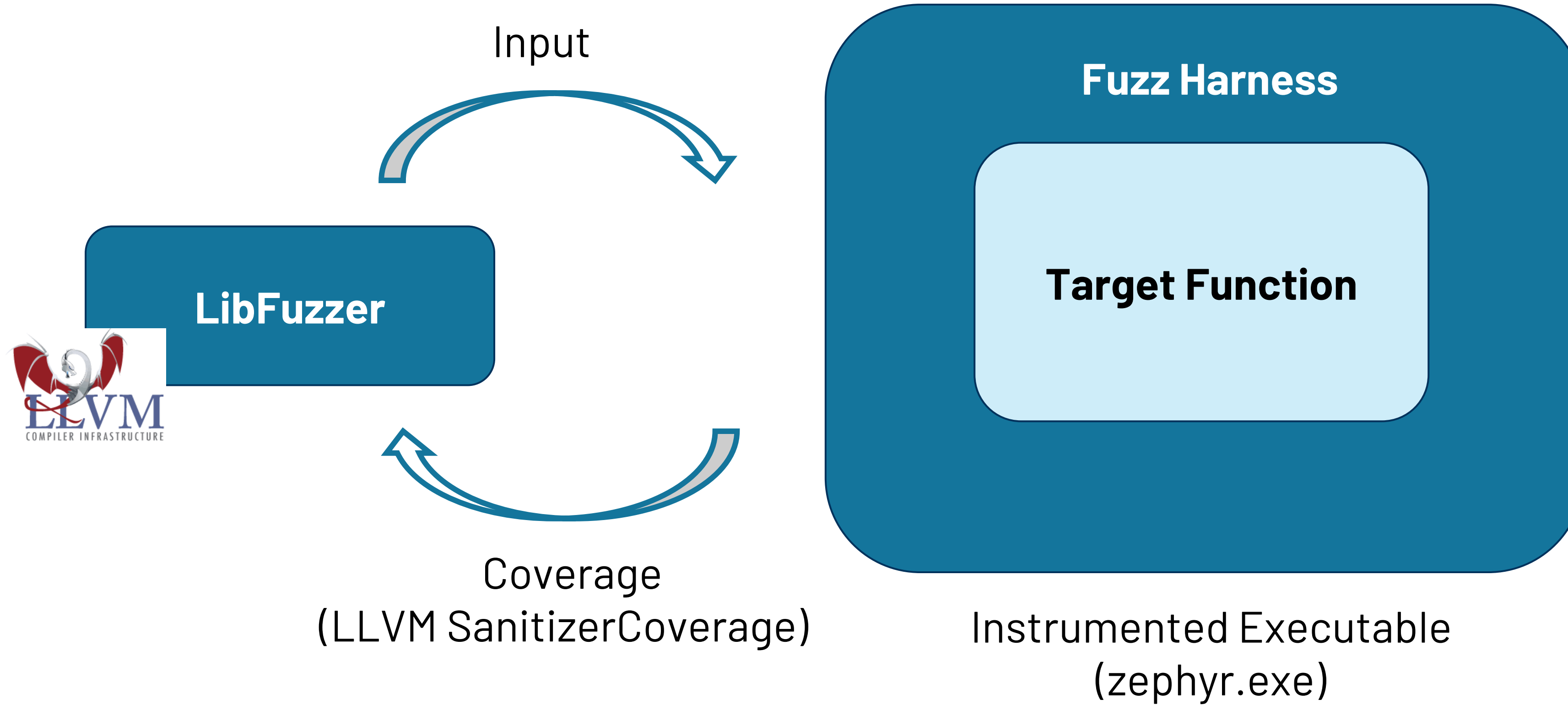
Fuzzing in Zephyr

- Integrated with Libfuzzer – https://llvm.org/docs/LibFuzzer.html
- An in-process coverage guided fuzzer from LLVM.org
- Only works with 64-bit Clang

```
$ clang --version
clang version 14.0.6
Target: x86_64-pc-linux-gnu
Thread model: posix
InstalledDir: /usr/bin
$ export ZEPHYR_TOOLCHAIN_VARIANT=host/llvm
$ west build -t run -b native_sim/native/64 samples/subsys/debug/fuzz
```

©2026 Analog Devices, Inc. All Rights Reserved.

How LibFuzzer works?



©2026 Analog Devices, Inc. All Rights Reserved.

How LibFuzzer works?

```
// fuzz_target.cc
extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
  DoSomethingInterestingWithMyAPI(Data, Size);
  return 0;  // Values other than 0 and -1 are reserved for future use.
}
```

zephyr-workspace > zephyr > samples > subsys > debug > fuzz > src > C main.c > ...

Fuzz Harness

```
101  #endif
102  int LLVMFuzzerTestOneInput(const uint8_t *data, size_t sz)
103  {
104      static bool runner_initialized;
105
106      if (!runner_initialized) {
107          nsi_init(0, NULL);
108          runner_initialized = true;
109      }
110
111      /* Provide the fuzz data to the embedded OS as an interrupt, with
112       * "DMA-like" data placed into native_fuzz_buf/sz
113       */
114      fuzz_buf = (void *)data;
115      fuzz_sz = sz;
116
117      hw_irq_ctrl_set_irq(CONFIG_ARCH_POSIX_FUZZ_IRQ);
118
119      /* Give the OS time to process whatever happened in that
120       * interrupt and reach an idle state.
121       */
122      nsi_exec_for(k_ticks_to_us_ceil64(CONFIG_ARCH_POSIX_FUZZ_TICKS));
123
124      return 0;
125  }
```

zephyr-workspace > zephyr > samples > subsys > debug > fuzz > src > C main.c > LLVMFuzzerTestC

```
64
65  static void fuzz_isr(const void *arg)
66  {
67      /* We could call check0() to execute the fuzz case here, but
68       * pass it through to the main thread instead to get more OS
69       * coverage.
70       */
71      k_sem_give(&fuzz_sem);
72  }
73
74  int main(void)
75  {
76      printk("Hello World! %s\n", CONFIG_BOARD);
77
78      IRQ_CONNECT(CONFIG_ARCH_POSIX_FUZZ_IRQ, 0, fuzz_isr, NULL, 0);
79      irq_enable(CONFIG_ARCH_POSIX_FUZZ_IRQ);
80
81      while (true) {
82          k_sem_take(&fuzz_sem, K_FOREVER);
83
84          /* Execute the fuzz case we got from LLVM and passed
85           * through an interrupt to this thread.
86           */
87          check0(fuzz_buf, fuzz_sz);
88      }
89      return 0;
90  }
```

Target Function

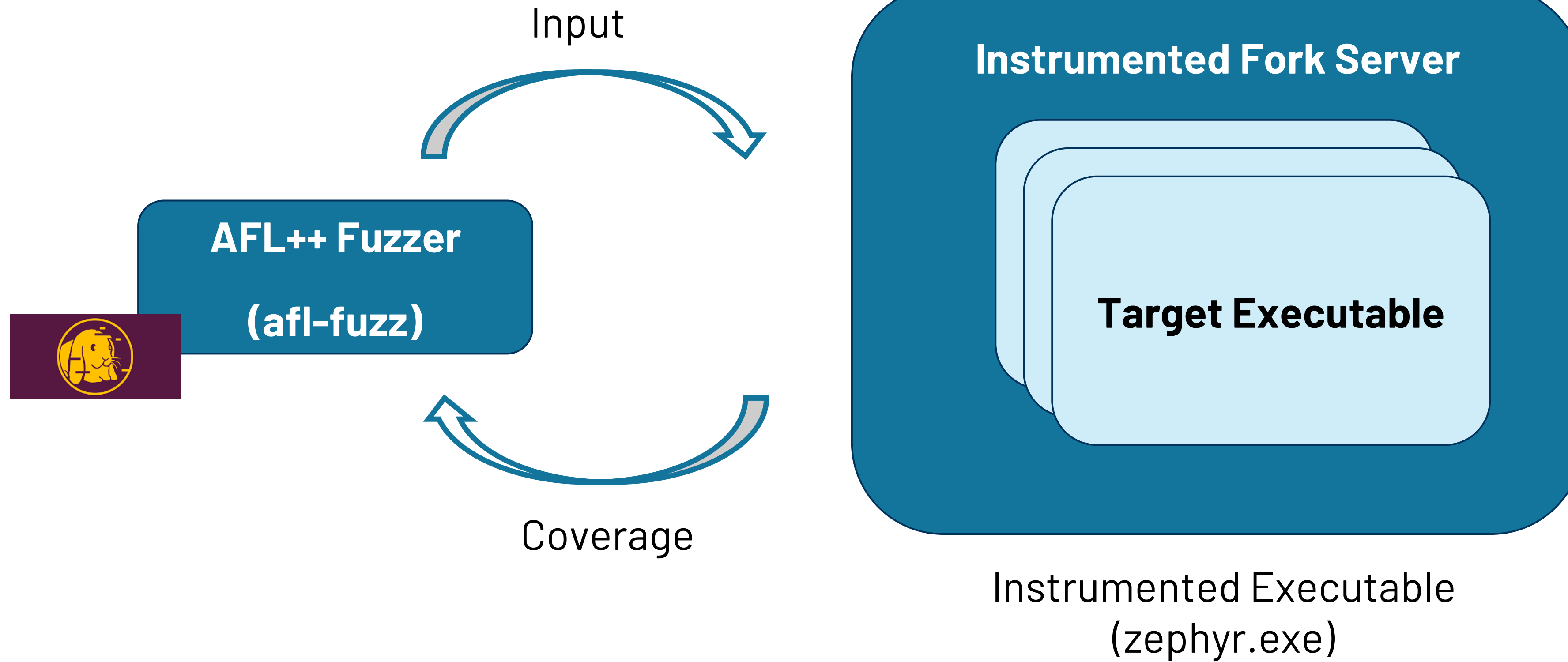
©2026 Analog Devices, Inc. All Rights Reserved.

What is AFL++?

- Popular open-source coverage-guided fuzzer
- Very widely used for general purpose application fuzzing and in research community
- A fork of Google's American Fuzzy Lop (AFL) fuzzer
- Better mutation strategies, good crash exploration and highly optimized for performance

©2026 Analog Devices, Inc. All Rights Reserved.

How AFL++ works?



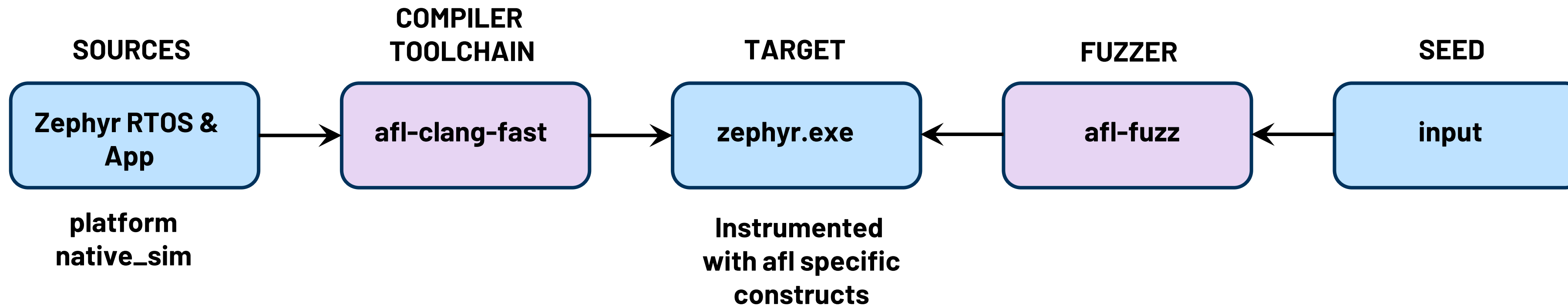
©2026 Analog Devices, Inc. All Rights Reserved.

LibFuzzer vs AFL++

LibFuzzer	AFL++
In-process – Fuzzer and target run in the same process	Out-of-process – Fuzzer forks new process for test case
(+) Minimal overhead (no process creation) – Fast	**(-)** Process creation overhead in traditional mode
(-) Crashes kill the fuzzer too; Poor crash isolation	**(+)** Fuzzer remains isolated from target crashes
(-) Part of LLVM/Clang – supports Clang only	**(+)** Standalone tool – supports GCC too
(-) Compiler-based - requires source	**(+)** Compiler-based; Can fuzz binaries in QEMU mode
(-) Harness Required	**(+)** No harness required in traditional mode
Use for simple setup – good for CI with short fuzzing runs	Use for long-term fuzzing campaigns
Fuzzing clean library APIs is suitable	Fuzzing complete applications is suitable

©2026 Analog Devices, Inc. All Rights Reserved.

Fuzzing Zephyr Apps with AFL++




```
$ west -v build -b native_sim/native/64 samples/subsys/debug/fuzz_afl/
  -p -- -DZEPHYR_TOOLCHAIN_VARIANT=llvm -DCONFIG_ARCH_POSIX_AFLPLUSPLUS=y

$ afl-fuzz     -i input/     -o findings/  -- build/zephyr/zephyr.exe
```


©2026 Analog Devices, Inc. All Rights Reserved.

Fuzzing Zephyr Apps with AFL++

- Fuzzing zephyr.exe with afl-fuzz setup
- Crashes (Injected) can be detected
- Running the code with crashing input can give us the backtrace
- -DCONFIG_ARCH_POSIX_AFLPLUSPLUS compilation enables compile with afl toolchain
- Limited to native_sim for now

```
  american fuzzy lop ++4.36a {default} (../build/zephyr/zephyr.exe) [explore]
┌─ process timing ────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 15 min, 39 sec      │  cycles done : 553   │
│   last new find : 0 days, 0 hrs, 7 min, 28 sec       │ corpus count : 10    │
│last saved crash : 0 days, 0 hrs, 4 min, 30 sec       │saved crashes : 1     │
│ last saved hang : none seen yet                      │  saved hangs : 0     │
├─ cycle progress ─────────────────────┬─ map coverage─┴──────────────────────┤
│  now processing : 9.431 (90.0%)      │    map density : 18.63% / 19.34%     │
│  runs timed out : 0 (0.00%)          │ count coverage : 2.23 bits/tuple     │
├─ stage progress ─────────────────────┼─ findings in depth ──────────────────┤
│  now trying : havoc                  │ favored items : 4 (40.00%)           │
│ stage execs : 120/1200 (10.00%)      │  new edges on : 9 (90.00%)           │
│ total execs : 2.32M                  │ total crashes : 4 (1 saved)          │
│  exec speed : 3028/sec               │  total tmouts : 2 (0 saved)          │
├─ fuzzing strategy yields ────────────┴───────────────┬─ item geometry ───────┤
│   bit flips : 0/0, 0/0, 0/0                          │    levels : 8         │
│  byte flips : 0/0, 0/0, 0/0                          │   pending : 0         │
│ arithmetics : 0/0, 0/0, 0/0                          │  pend fav : 0         │
│  known ints : 0/0, 0/0, 0/0                          │ own finds : 9         │
│  dictionary : 0/0, 0/0, 0/0, 0/0                     │  imported : 0         │
│havoc/splice : 9/2.32M, 0/0                           │ stability : 97.24%    │
│py/custom/rq : unused, unused, unused, unused         ├───────────────────────┘
│    trim/eff : 21.00%/17, n/a                         │          [cpu000: 18%]
└─ strategy: explore ────────── state: in progress ────┘
```

RFC: https://github.com/zephyrproject-rtos/zephyr/issues/103685
PR: https://github.com/zephyrproject-rtos/zephyr/pull/103876

©2026 Analog Devices, Inc. All Rights Reserved.

ANALOG DEVICES

Fuzzing in Zephyr – Possible Roadmap

Target Platforms

- Integrate on Posix platforms
- Target QEMU based platforms initially

Possible Generic Approach

- Models for generic peripherals like HALucinator
- Task based fuzzing like RT-Fuzzer

Immediate Action Items

- Brainstorm with the Security Working Group
- Gather ideas from the community

©2026 Analog Devices, Inc. All Rights Reserved.

Conclusion

Conclusion & Takeaways

- Fuzzing is a run time testing technique to find security flaws in software.
- Fuzzing Embedded Applications presents challenges in providing input, creating the execution environment and in improving coverage and detecting crashes.
- Existing solutions use hardware, emulation, or rehosted systems with modeled peripherals, employing full source code level, binary-only or API-level fuzzing.
- Zephyr's current LibFuzzer integration targets unit-level API fuzzing but has limitations.
- We aim to integrate AFL++, a popular fuzzing engine, to create a generalized fuzzing strategy across Zephyr's supported platforms.

©2026 Analog Devices, Inc. All Rights Reserved.

AHEAD OF WHAT'S POSSIBLE

analog.com

©2026 Analog Devices, Inc. All Rights Reserved.