# EMBEDDED SECURITY

PUTrequest_
Robert Szczepański
23.11.2022

# CZYM JEST EMBEDDED SECURITY?

*Embedded systems security is a design methodology, implementation, and commitment that companies embrace to limit the threat exposure of the devices they build and the data these devices generate. [...] It starts well before the first line of code is written, includes protection in case a device falls into the hands of attackers, and continues until a device has been decommissioned.*

*Źródło: https://www.windriver.com/solutions/learning/embedded-systems-security*

# PRZYKŁADY ATAKÓW

# PRZYKŁADY ATAKÓW

- Side-channel
- Rowhammer
- Buffer overflow
- Trojany sprzętowe
- ...

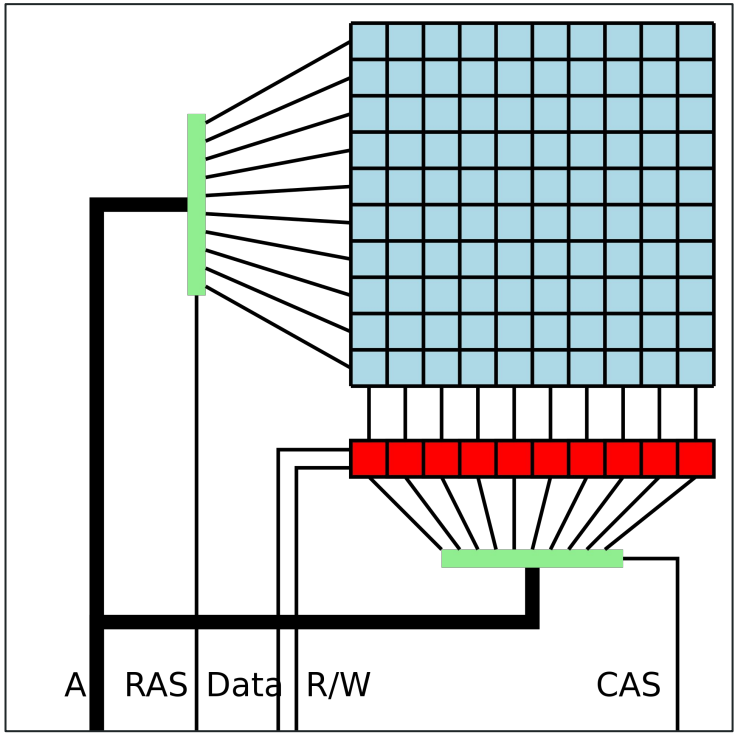
# SIDE-CHANNEL ATTACK

- Pasywne:
  - Analiza poboru prądu
  - Analiza pola elektromagnetycznego
  - Refleksy
  - “Patrzenie przez ramię”
  - ...
- Aktywne:
  - Clock/power glitching
  - Błędy elektromagnetyczne (EMFI)
  - Błędy laserowe/optyczne
  - ...

*Źródło: https://arxiv.org/pdf/1611.03748.pdf*

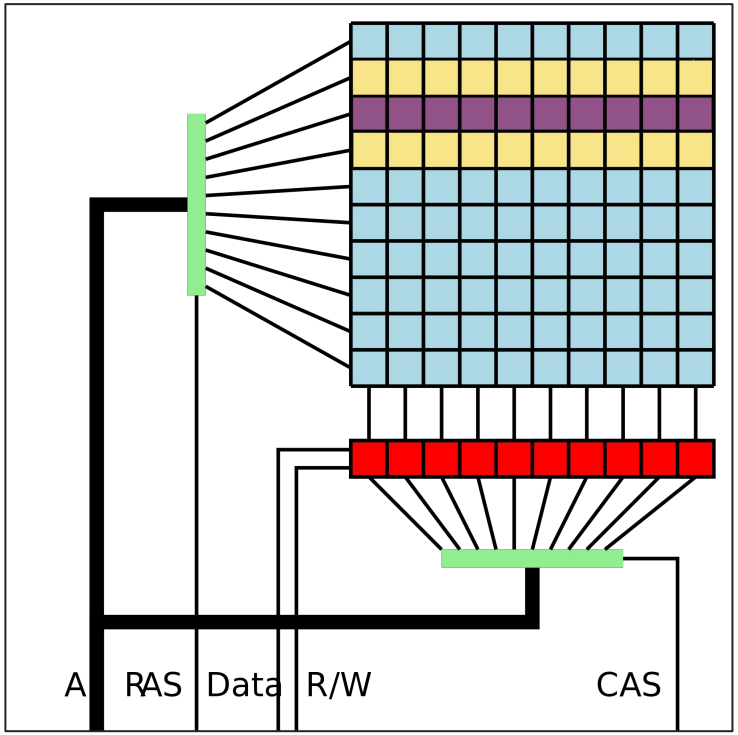
# ROWHAMMER

- Wykorzystuje fizyczne własności pamięci DRAM:
  - Komórki pamięci są umieszczone bardzo blisko siebie,
  - Ładunki elektryczne “wyciekają” z komórek blisko siebie,
  - Komórki składają się z kondensatora i tranzystora,
  - Ładunek przechowywany w kondensatorze jest ulotny i wymaga “odświeżania”.



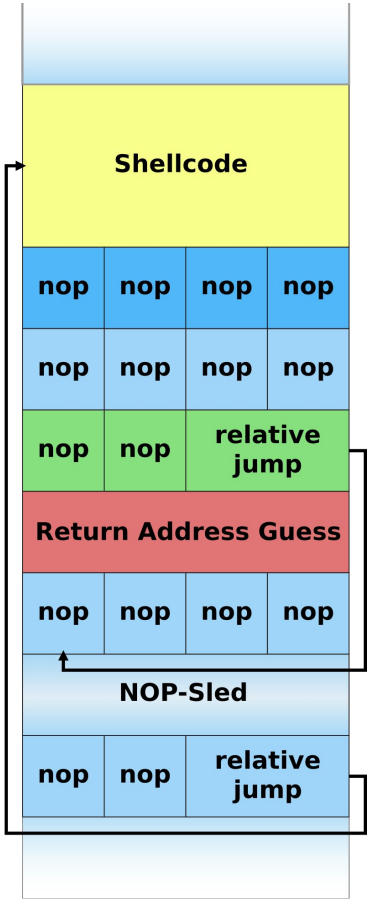
# ROWHAMMER

- Szybkie odczyty dwóch rzędów otaczających inny powoduje możliwość zmiany bitu (bitflip) w rzędzie pomiędzy nimi.
- Efekt występuje także dla odczytów oddalonych od siebie oraz pojedynczych rzędów.



# BUFFER OVERFLOW

- Dane programu przechowywane są w odpowiednich buforach.
- W najlepszym przypadku atak zakończy się błędem programu np. “segmentation fault”.
- Możliwe jest zmanipulowanie go do przeskoczenia i wykonania kodu podanego przez atakującego.



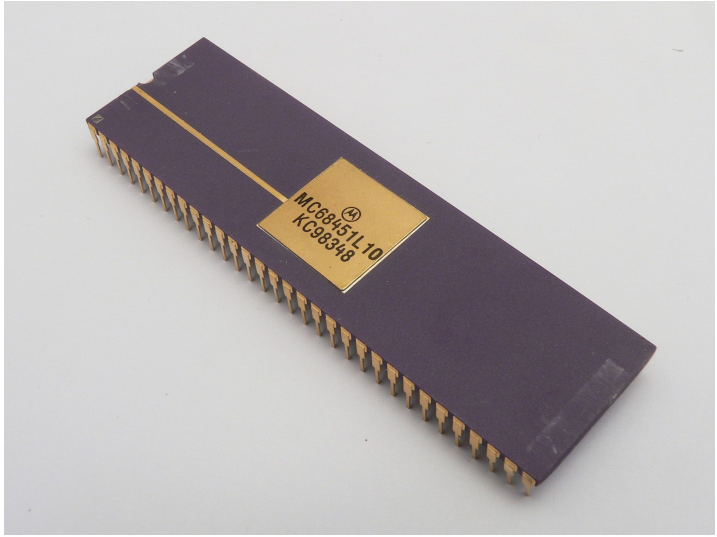
# PRZYKŁADY ZABEZPIECZEŃ

# PRZYKŁADY ZABEZPIECZEŃ

- Memory Management Unit (MMU)
- Memory Protection Unit (MPU)
- Error Correction Code Memory (ECCM)
- ...

# MEMORY MANAGEMENT UNIT

- Jednostka sprzętowa, której zadaniem jest m.in. tłumaczenie adresów wirtualnych na fizyczne i odwrotnie.
- Poza tym odpowiada za:
  - zarządzanie pamięcią wirtualną,
  - ochronę niedostępnej pamięci,
  - kontrolę pamięci cache,
  - zarządzanie adresami magistrali.



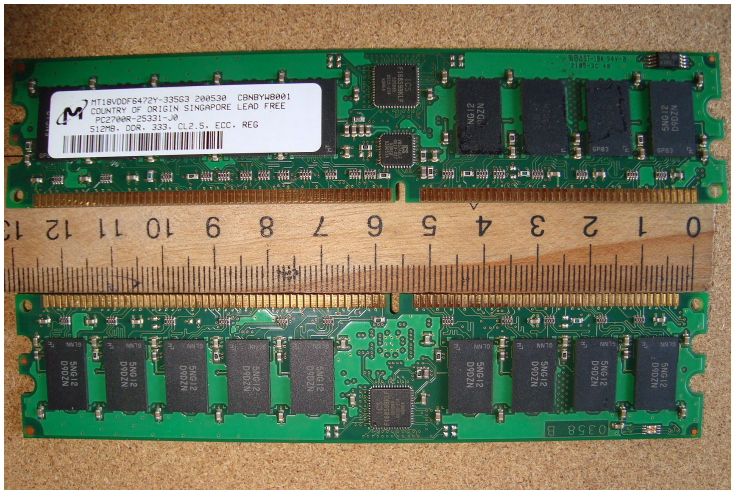
# MEMORY PROTECTION UNIT

- “Zmniejszona” wersja MMU.
- Jedynym zadaniem układu jest ochrona pamięci.
- Pozwala na przypisywanie poziomu uprawnień do fragmentów pamięci i kontroluje dostęp do nich.

# ERROR CORRECTION CODE MEMORY

Pamięć wyposażona w układ ECC:

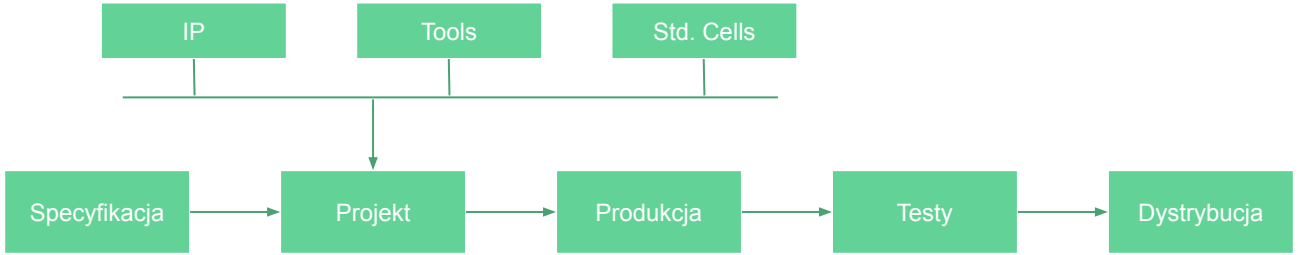
- Error Correction Code dzięki redundancji zapobiega błędom i uszkodzeniom danych w pamięci,
- Bywa pomocne przy atakach rowhammer.



# TROJANY SPRZĘTOWE

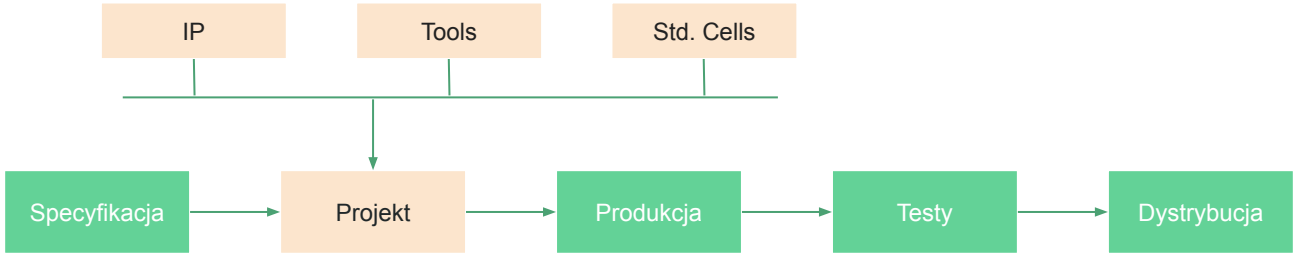
# CYKL WYTWARZANIA SW

SW - systemy wbudowane



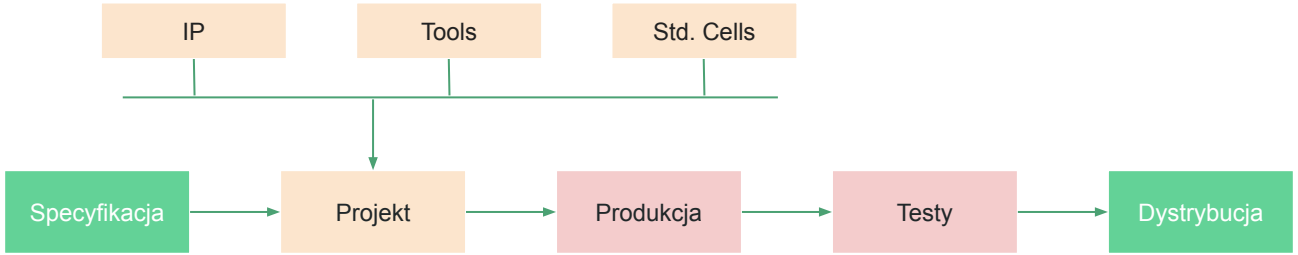
# CYKL WYTWARZANIA SW

SW - systemy wbudowane



# CYKL WYTWARZANIA SW

SW - systemy wbudowane



# TROJAN SPRZĘTOWY

- Bardzo trudno wykrywalny, nieznacznie modyfikuje urządzenie.
- Może zmieniać działanie, obniżać wydajność, zmniejszać bezpieczeństwo, a nawet uszkodzić układ.
- Umieszczany w sprzęcie zazwyczaj drogą niepożądaną oraz niezależną od projektanta.

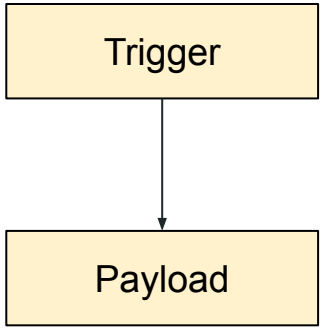
# TROJAN SPRZĘTOWY - STRUKTURA

Trigger - trojan uruchamiany jest po wystąpieniu określonego wydarzenia, np.:

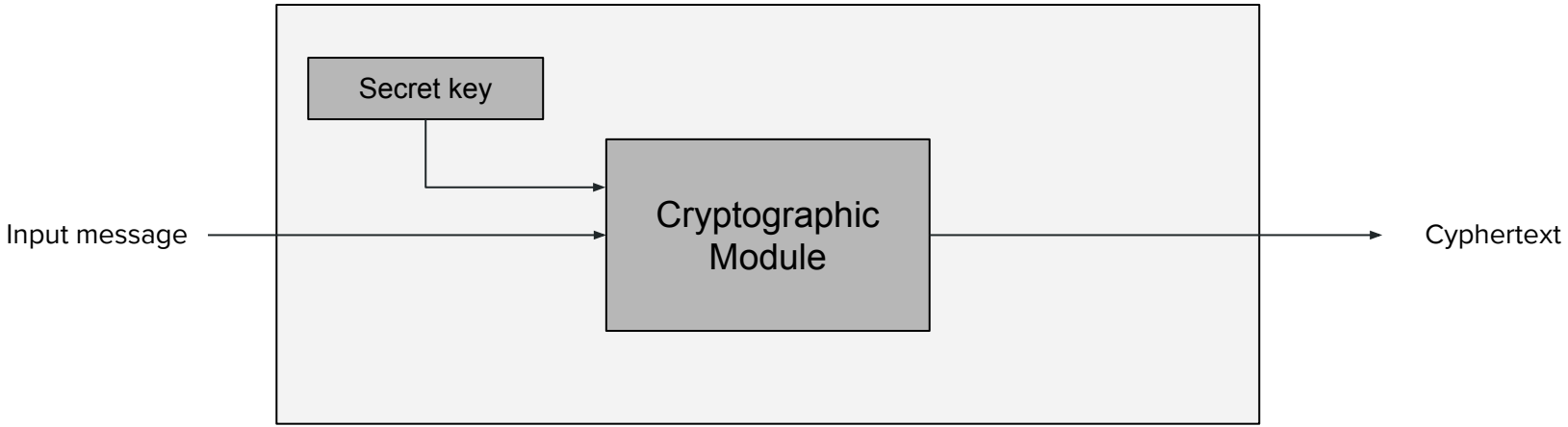
- Odczyt/zapis do konkretnego adresu na magistrali,
- Sekwencja wartości na różnych sygnałach/rejestrach,
- Upłynięcie danej ilości czasu.

Payload - czynność następująca po triggerze, np.:

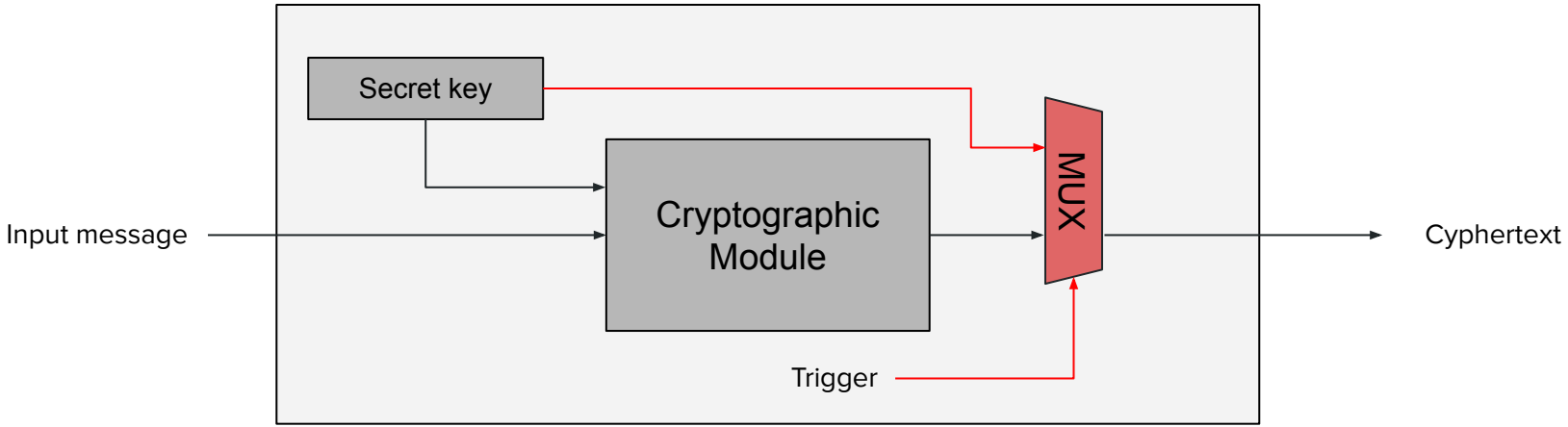
- Zmiana poziomu uprawnień do obszaru pamięci,
- Zapis ukrytych danych na wyjściu,
- Wyłączenie systemu.



# TROJAN SPRZĘTOWY - PRZYKŁAD



# TROJAN SPRZĘTOWY - PRZYKŁAD



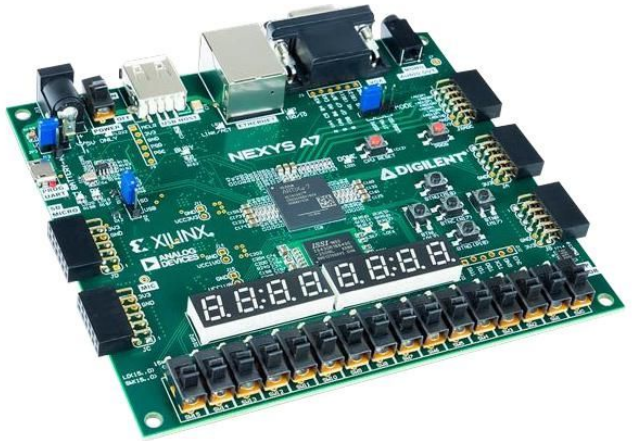
# FPGA

# FPGA

Field Programmable Gate Array - programowalny układ logiczny. Modyfikowalny odpowiednik układów scalonych.

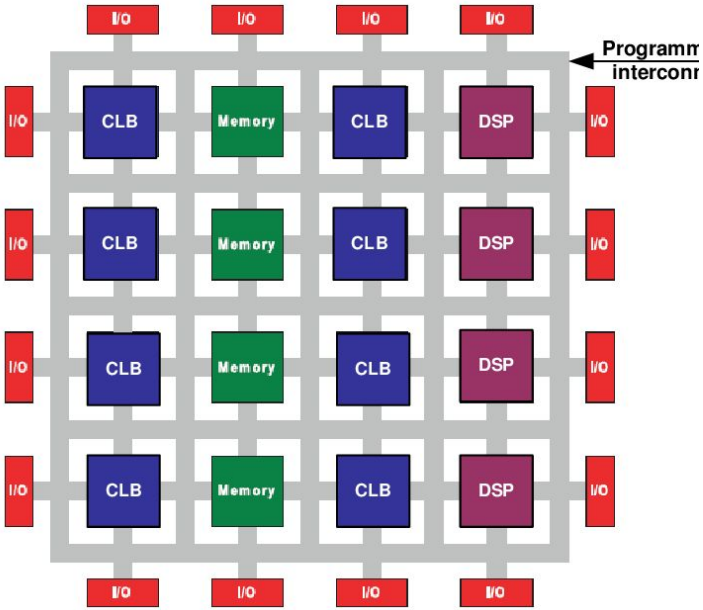
Zazwyczaj wolniejsze od ASIC jednak ze względu na elastyczność stanowią dobre rozwiązania do takich dziedzin jak:

- Układy, które muszą być zdalnie modyfikowalne (np. w kosmosie),
- Układy na liniach produkcyjnych,
- Testowanie projektów na układy scalone.



# FPGA - Budowa

- CLB - Configurable Logic Block
- DSP - Digital Signal Processor
- BRAM - Block RAM
- I/O - wyprowadzenia wejścia/wyjścia



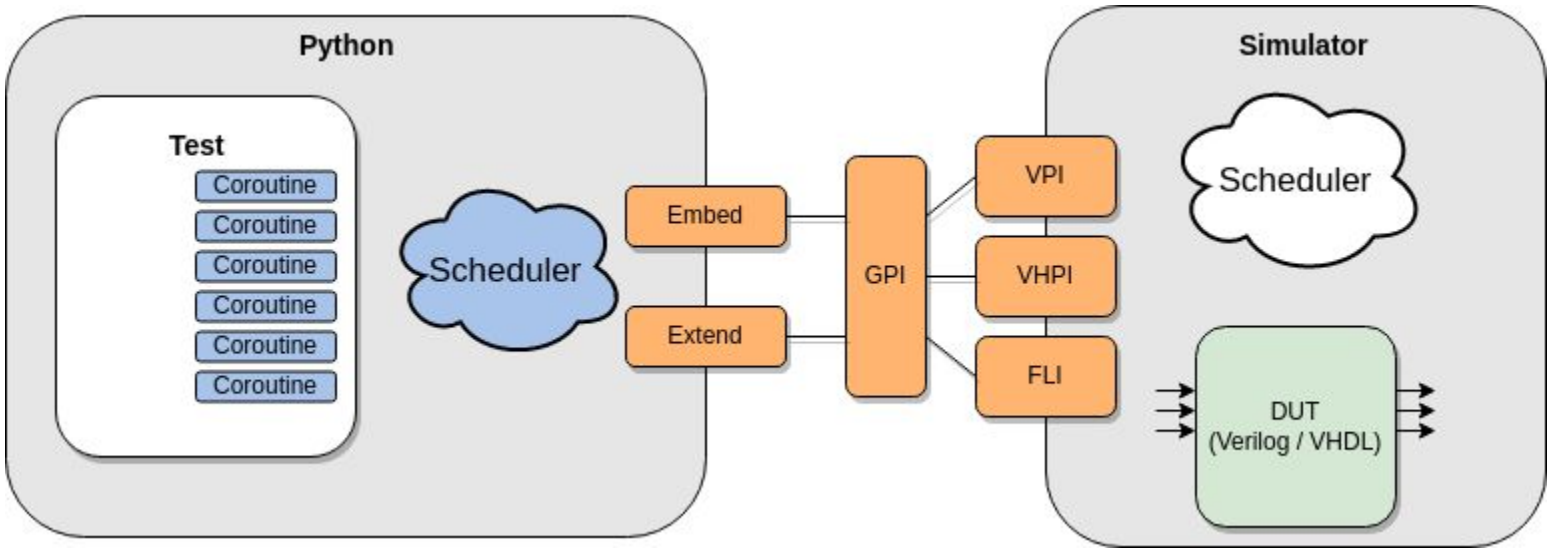
# СОСОТВ

# COCOTB

*Cocotb is a **CO**routine based **CO**simulation **T**est**B**ench environment for verifying VHDL and SystemVerilog RTL using Python.*

Środowisko, do testowania projektów opisu sprzętu, w języku Python.

# COCOTB


Python
Test
Coroutine
Coroutine
Coroutine
Coroutine
Coroutine
Coroutine
Scheduler
Embed
Extend
GPI
VPI
VHPI
FLI
Simulator
Scheduler
DUT
(Verilog / VHDL)

# COCOTB - Projekt

```
$ cat sample.sv

`timescale 1us/1us

module fake_crypto (
  input logic clk,
  input logic [7:0] input_,
  output logic [7:0] output_
);

always @(posedge clk) begin
  output_ <= ~input_;
end

// Dump waves
initial begin
  $dumpfile("dump.vcd");
  $dumpvars(1, fake_crypto);
end

endmodule
```

# COCOTB - Słowa kluczowe

- `@cocotb.test()` - dekorator, informacja dla cocotb, że funkcja jest testbenchem dla DUT,
- `async` - wywołanie polecenia asynchronicznie jako nowe coroutine
- `await` - lokalne oczekiwanie na zakończenie polecenia
- `assert` - sprawdzenie czy dany warunek jest spełniony

# COCOTB - Dostępy

- `x = dut.x`
- `dut.x.value = 1`
- `dut.x[5].value = 1`
- `dut.x.value = Force(1)`
- `dut.x.value = Freeze()`
- `dut.x.value = Release()`

- `await Edge(dut.x)`
- `await RisingEdge(dut.x)`
- `await FallingEdge(dut.x)`
- `await ClockCycles(dut.x, num_cycles=10, rising=True)`
- `await Time(100, ‘us’)`

# COCOTB - Python test

*$ cat test_sample.py*

```
import cocotb
from cocotb.triggers import Timer, FallingEdge
from cocotb.clock import Clock


@cocotb.test()
async def sample_simple_test(dut):
    clock = Clock(dut.clk, 10, units="us")  # Create a 10us period clock on port clk
    cocotb.start_soon(clock.start())        # Start the clock
    await FallingEdge(dut.clk)              # Synchronize with the clock

    dut.input_.value = 0b11100111

    await Timer(100, 'us')
    dut.input_.value = 0x21
    await Timer(100, 'us')
    assert dut.output_.value == (~0x21 & 0xff)

    dut._log.info("out is %s", dut.output_.value)
```

# COCOTB - Makefile

```
$ cat Makefile

TOPLEVEL_LANG ?= verilog
SIM ?= verilator

VERILOG_SOURCES = $(shell pwd)/sample.sv

MODULE = test_sample
TOPLEVEL = fake_crypto

include $(shell cocotb-config --makefiles)/Makefile.sim
```

# COCOTB - Uruchamianie

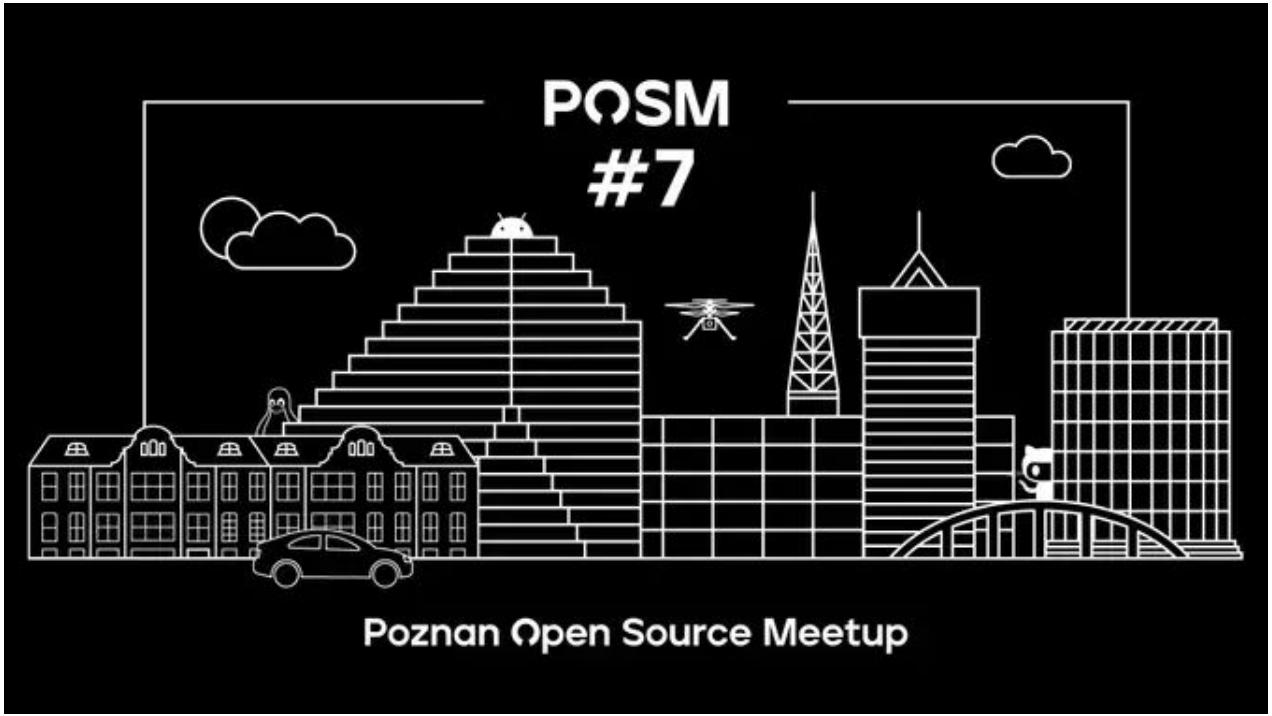
Cocotb korzysta z systemu budowania `Make`, uruchamianego komendą `make`.
Ważne flagi:

- `EXTRA_ARGS` - dodatkowe parametry dla symulatora, muszą być przekazane przed make,
- `SIM` - wybór symulatora (np. `verilator`),
- `TOPLEVEL` - nazwa głównego modułu.

Przykład uruchomienia z wykorzystaniem Verilatora z włączonym tracingiem:

```
EXTRA_ARGS=”--trace –trace-structs” make SIM=verilator
```

# POSM #7



# POSM #7

1. Taking photos without a camera - photorealistic device renders using Blender API; Maciej Zientara, Antmicro
2. **Efficient digital design co-simulation using cocotb; Maciej Dudek, Antmicro**
    a. Implementacja magistrali Tilelink i AHB wykorzystana do projektu ASIC.

github.com/putrequest/embedded-cocotb_1