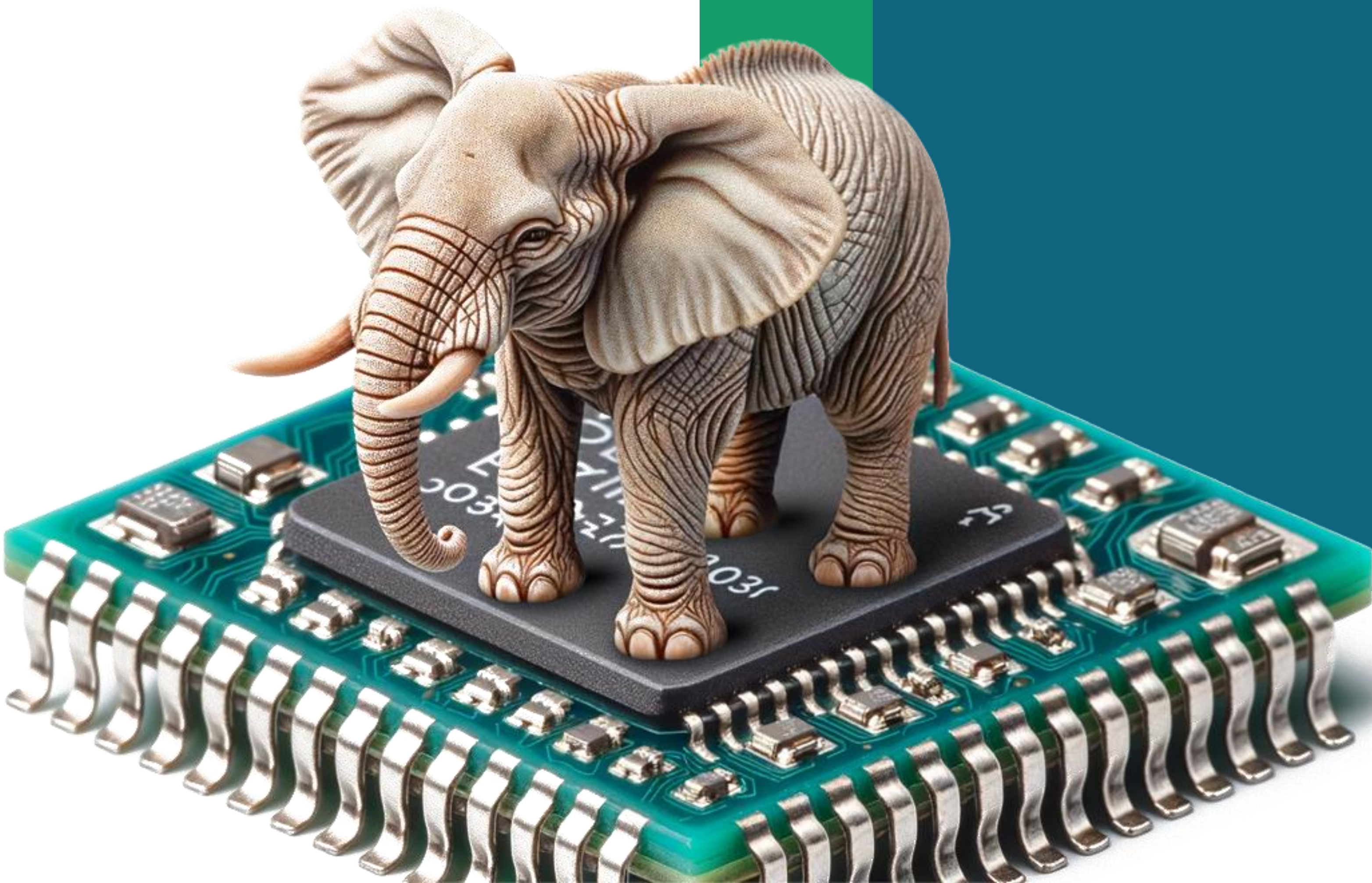
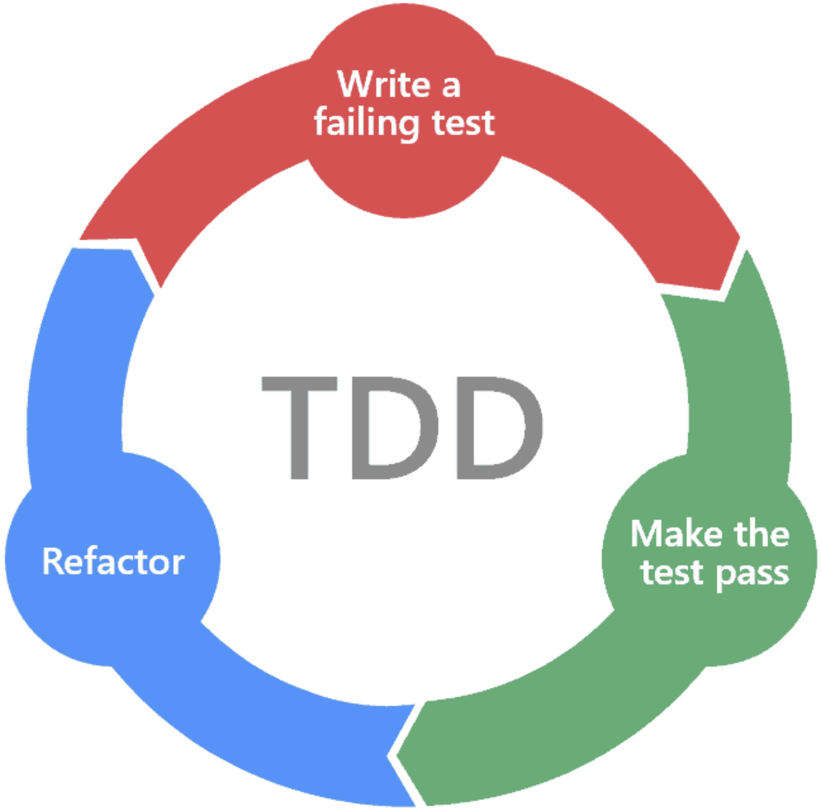
TDD for Microcontroller

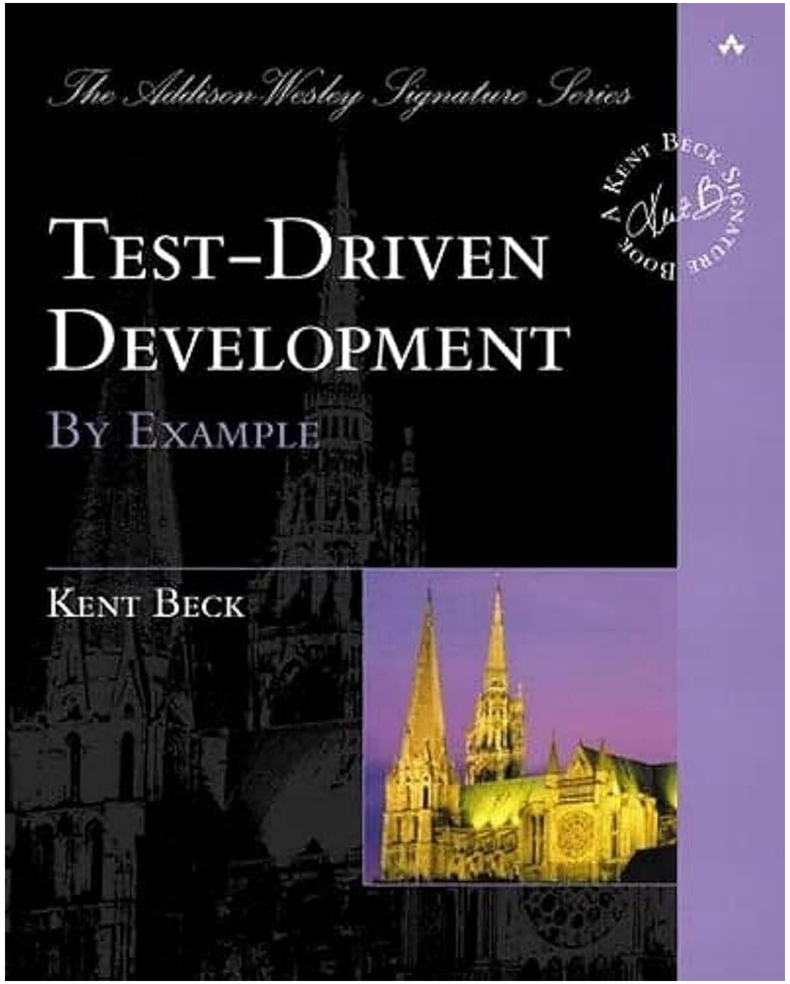
A practical test-first approach

Daniel Penning, embeff GmbH



TDD in a nutshell





TDD requires quick and automated feedback.

How to get feedback

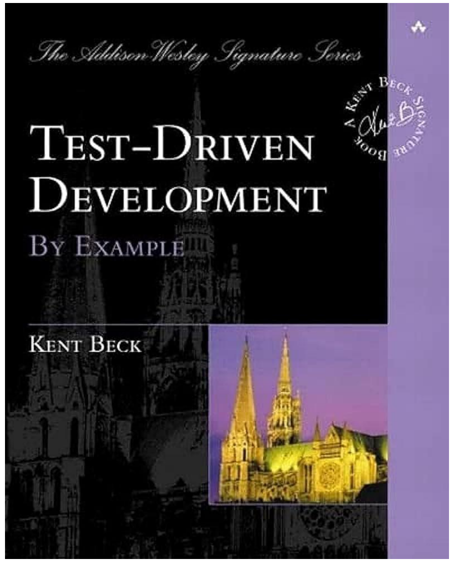
Traditional software

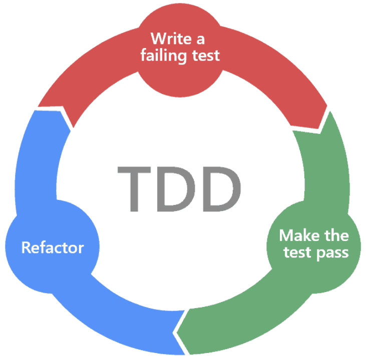
- Run the program
- Test isolated parts of the code
  - Decouple dependencies
  - Write unit test

Embedded

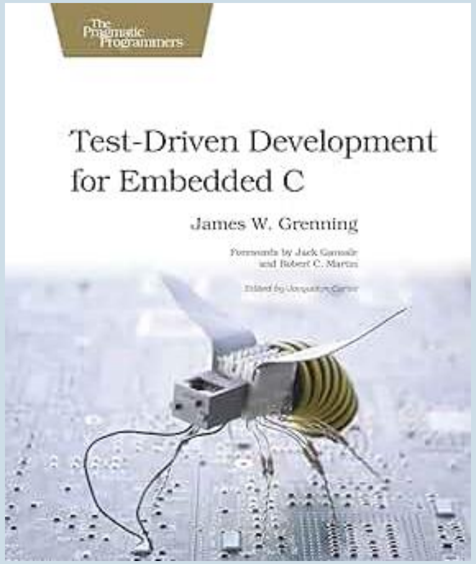
- Run the binary?
  - Flash microcontroller
  - Connect sensor / other devices
- Test isolated parts of the code?

The road to get TDD for microcontroller





2002



Host

1. Develop
2. Cross Compile
4. View Results

Target

3. Run Tests

Host

1. Develop
2. Compile
3. Run Tests
4. View Results

2011

embeff BETTER EMBEDDED.

Dual Targeting

Host

1. Develop Code+Test
2. Cross Compile
4. View Results

Target

3. Run Tests

Host

1. Develop Code+Test
2. Compile
3. Run Tests
4. View Results

"On-Target" Tests

- „slow"
- difficult to automate

"Off-Target" Tests

- fast
- easy to automate

Off-Target Testing: Decouple code (1/2)

```c
#include <def/cortex_m33.h>

OUR_UINT32 crc32(uint8_t const* buffer, size_t len) {
    const OUR_UINT32 POLY = 0x04C11DB7;
    OUR_UINT32 crc = -1;

    while( len-- ) {
        crc = crc ^ (*buffer++ << 24);
        for( int bit = 0; bit < 8; bit++ ) {
            if( crc & (1L << 31)) crc = (crc << 1) ^ POLY;
            else                  crc = (crc << 1);
        }
    }
    return crc;
}
```

Off-Target Testing: Decouple code (2/2)

```
#include <cstdint>

uint32_t crc32(uint8_t const* buffer, size_t len) {
    const uint32_t POLY = 0x04C11DB7;
    uint32 crc = -1;

    while( len-- ) {
        crc = crc ^ (*buffer++ << 24);
        for( int bit = 0; bit < 8; bit++ ) {
            if( crc & (1L << 31)) crc = (crc << 1) ^ POLY;
            else                  crc = (crc << 1);
        }
    }
    return crc;
}
```

```
TEST(TestSoftwareCrc, SampleData16Bytes_ReturnPreCalculatedValue) {
    uint8_t const SampleData[16] = {0x34,0x7c,0x1b,0x18, /* ... */ };
    EXPECT_EQ(crc32(SampleData, 16), 0x0547A4CCu);
}
```

Off-Target Testing

Host

1. Develop Test+Code
2. Compile
3. Run Tests
4. View Results

Setup

1. Setup a different toolchain hat compiles for the Host Machine (gcc, clang, MSVC).
2. Integrate a Unit Test Framework (Googletest, Catch2, Unity).

Steps for each test

1. Decouple the relevant code from <u>all</u> hardware dendencies.
2. Write a unit test and compile the test+code for your Host.
3. Run the test.

TDD Requirement	Off-Target Testing
Automated feedback	✅ (straight-forward)
Quick feedback	✅ (<3s, build)

Off-target: Risks

```
#include <iostream>

int fun1() { printf("fun1() \n"); return 0; }
int fun2() { printf("fun2() \n"); return 0; }

void foo(int x, int y) { printf("foo() \n"); }

int main() {
    foo(fun1(), fun2());
}
```

x86-64 with gcc 9.2

```
fun2()
fun1()
foo()
```

Cortex-M4 with arm-gcc-none-eabi 8

```
fun1()
fun2()
foo()
```

8. The order of evaluation of arguments is unspecified. All side effects of argument expression evaluations take effect before the function is entered.

C++98 Standard, 5.2.2 Function call [1]

unspecified behavior

behavior, for a well-formed program construct and correct data, that depends on the implementation.

Off-target: Risks (cont.)

Language

implementation-defined behaviour

behaviour, for a well-formed program construct and correct data, that depends on the implementation and which is documented at each implementation.

Undefined behaviour

behaviour for which this International Standard imposes no requirements

Hardware

Fused MAC instructions give incorrect results for rare data combinations

The Cortex-M4 processor includes optional floating-point logic which supports the fused MAC instructions (VFNMA, VFNMS, VFMA, VFMS). This erratum causes fused MAC operations on certain combinations of operands to result in either or both of the following:

- A result being generated which is one Unit of Least Precision (ULP) greater than the correct result.
- Inexact or underflow flags written to the Floating-point Status Control Register, FPSCR, incorrectly.

Arm Errata 839676

Toolchain

4.9 series reproducibly corrupts register R7

Code below does not exhibit problem when compiled with 4.8 2014q3. Code trashes register R7 with 4.9 2015q1, q2, or q3 and the flagged braces are uncommented. Commenting out the flagged braces removes the problem with the 4.9 series.

GNU Arm gcc #1527413

Off-target: Hardware-dependent code?

```cpp
#include <stm32u575xx.h>

uint32_t crc32(uint8_t const* buffer, size_t len) {
    uint32_t const* arr = reinterpret_cast<uint32_t const*>(buffer);
    uint32_t const* const end = arr + (len/4);
    CRC->CR = CRC_CR_RESET;

    while (arr < end) {
        uint32_t val = *arr++;
        CRC->DR = __builtin_bswap32(val);
    }
    return CRC->DR;
}
```

```cpp
TEST(TestHardwareCrc, SampleData16Bytes_ReturnPreCalculatedValue) {
    uint8_t const SampleData[16] = {0x34,0x7c,0x1b,0x18, /* ... */ };
    EXPECT_EQ(crc32(SampleData, 16), 0x0547A4CCu);
}
```

Can we write tests like that?

On-Target Testing

Host

3. Run Tests

Target

1. Develop Code + Tests
2. Cross Compile
4. View Results

Setup

- Integrate a Unit Test Framework (Googletest, Catch2, Unity).
- Adapt it so it compiles for your target. Stream output (e.g. via UART).
- Collect output for PC.

Steps for each test

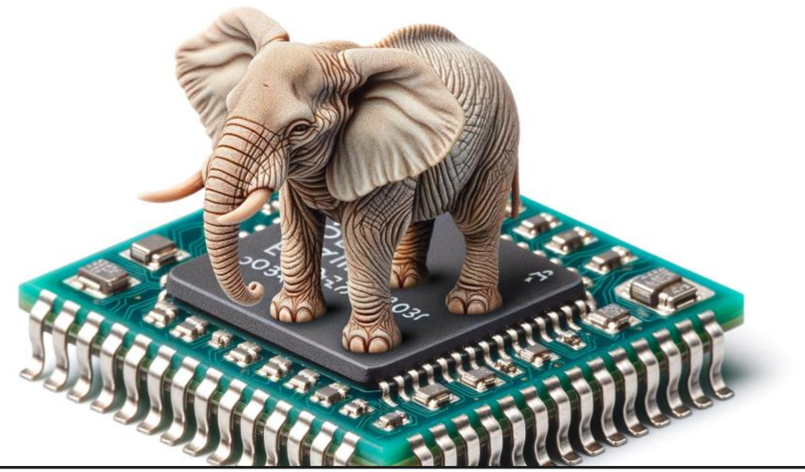
- Write a unit test and compile test + code for your Target.
- Flash the Target.
- Run the binary (= running the tests).

TDD Requirement	On-Target Testing
Automated feedback	✅ (special setup required)
Quick feedback	✅ (<10s, build+flash)

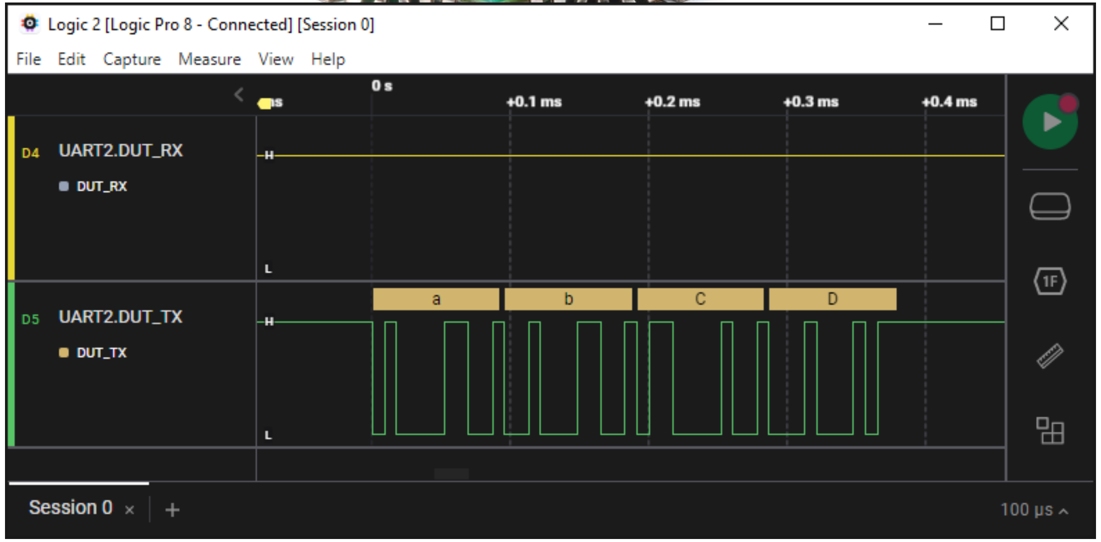
Testing pin behavior: “State of the art” (2023)

Most HAL code affects the external pins. How to test?

- Manual testing while developing.
- Register-based testing.



```
UART_HandleTypeDef huart2{};
huart2.Instance = USART2;
huart2.Init.BaudRate = 115200;
huart2.Init.WordLength = UART_WORDLENGTH_9B;
huart2.Init.StopBits = UART_STOPBITS_1;
huart2.Init.Parity = UART_PARITY_ODD;
/* More init settings...   */
HAL_UART_Init(&huart2);
uint8_t transmitData[] = "abCD";
HAL_UART_Transmit(&huart2, transmitData, 4, 0);
```



Register based testing

Core idea:

- Reference Manual is binding contract between SFR and Pin behavior.
- Instead of Pin behavior we check if HAL is doing correct SFR-access.

```
UART_HandleTypeDef huart2{};
huart2.Instance = USART2;
huart2.Init.BaudRate = 115200;
huart2.Init.WordLength = UART_WORDLENGTH_9B;
huart2.Init.StopBits = UART_STOPBITS_1;
huart2.Init.Parity = UART_PARITY_ODD;
/* More init settings...   */
HAL_UART_Init(&huart2);
uint8_t transmitData[] = "abCD";
HAL_UART_Transmit(&huart2, transmitData, 4, 0);
```

```
TEST(TestUartHal, Init_Baudrate_Correct) {
    UART_HandleTypeDef huart2{};
    huart2.Instance = USART2;
    huart2.Init.BaudRate = 115200;
    HAL_UART_Init(&huart2);
    EXPECT_EQ(USART2->BRR, /* ???? */);
}
```


embeff
BETTER EMBEDDED.

Register-Based-Testing: Find the correct values

1

57.8.5 USART baud rate register (USART_BRR)

This register can only be written when the USART is disabled (UE=0). It may be automatically updated by hardware in auto baud rate detection mode.

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
BRR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **BRR[15:0]:** USART baud rate

BRR[15:4]

BRR[15:4] correspond to USARTDIV[15:4]

BRR[3:0]

When OVER8 = 0, BRR[3:0] = USARTDIV[3:0].

When OVER8 = 1:

BRR[2:0] = USARTDIV[3:0] shifted 1 bit to the right.

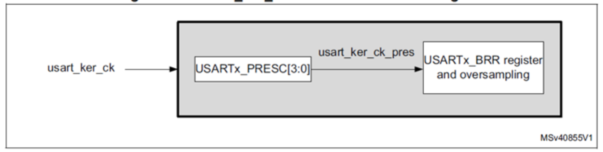
BRR[3] must be kept cleared.

2

$$\text{Tx/Rx baud} = \frac{\text{usart\_ker\_ck\_pres}}{\text{USARTDIV}}$$

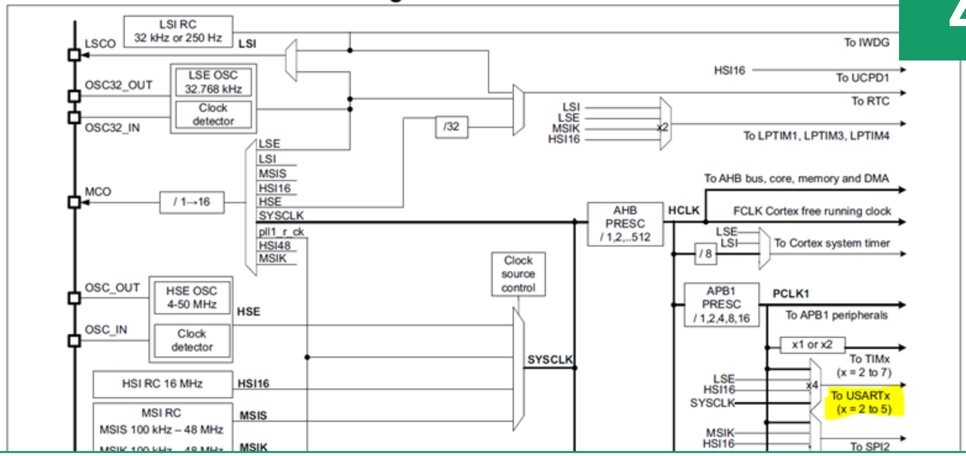
3

Figure 661. usart_ker_ck clock divider block diagram



4

Figure 33. Clock tree



...

Register-based testing

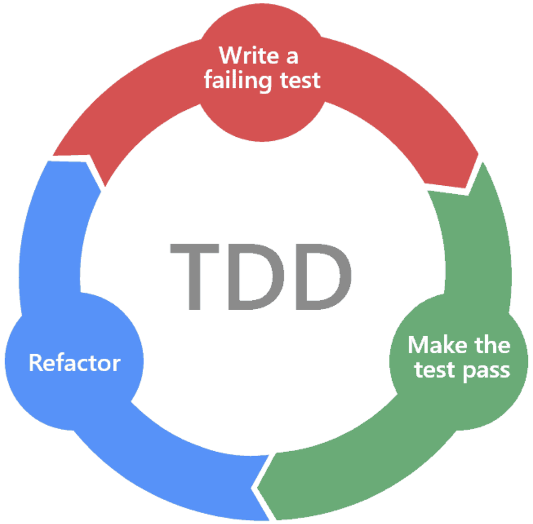
Core idea:

- Reference Manual is binding contract between SFR and Pin behavior.
- Instead of Pin behavior we check if HAL is doing correct SFR-access.

How it prevents TDD

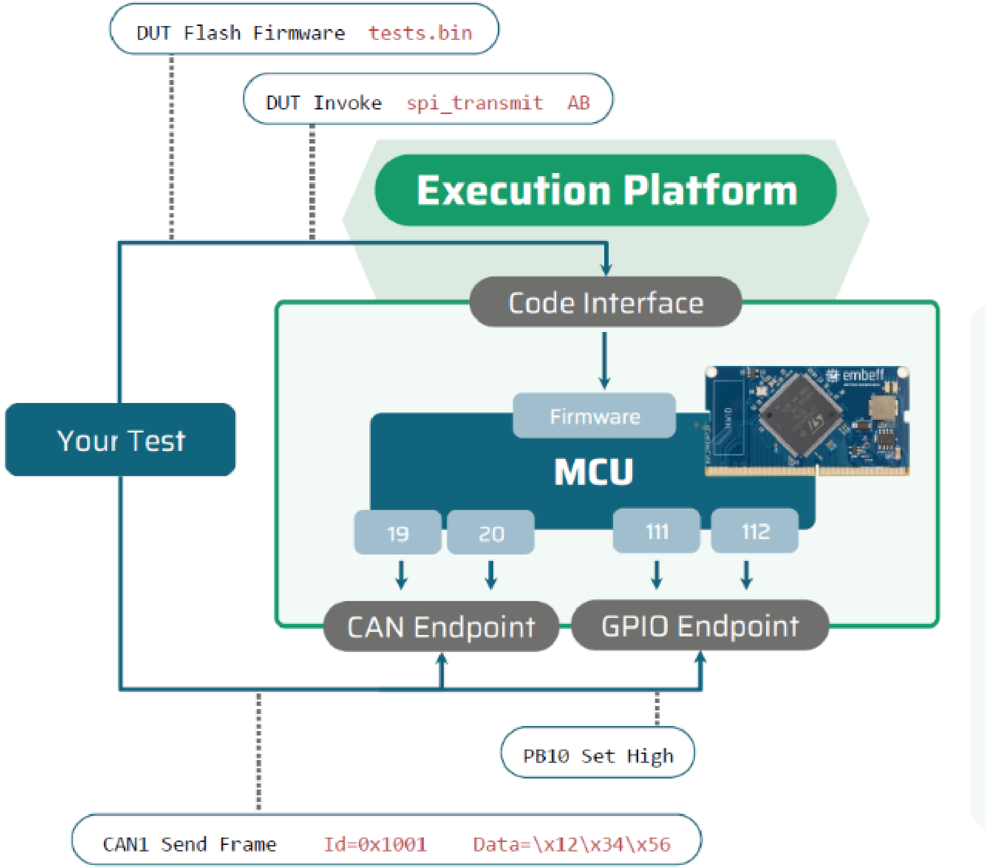
1. **What are the “correct” register values?**
   - Current microcontrollers are complex.
   - Dependencies between many peripherals.

2. **Whitebox-nature**
   - Checks against a specific implementation.
   - Refactoring will often change this implementation.



Involve pin behavior: Open Loop Tests

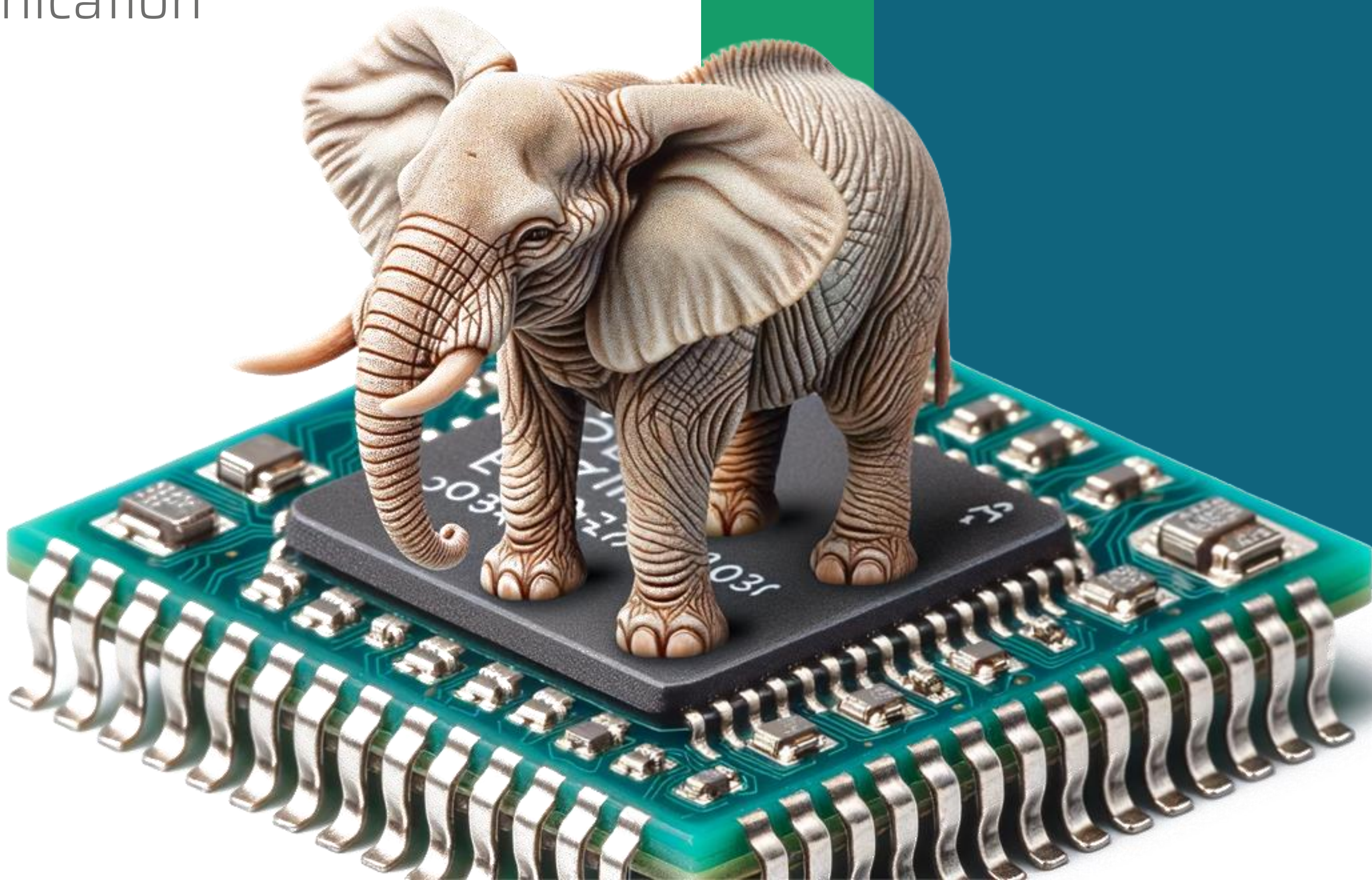
1. **Make pins accessible**
   - Use the exact microcontroller
   - Use same basic wiring (power/clock)
   - Relevant pins on a connector
2. **Make code accessible**
   - Provide a way to flash the new/updated firmware
   - Allow to call functions running on the microcontroller.
3. **Make periphery accessible**
   - Observe a peripheral and provide functions to read back the results.
   - Provide functions to actively transmit on UART etc.



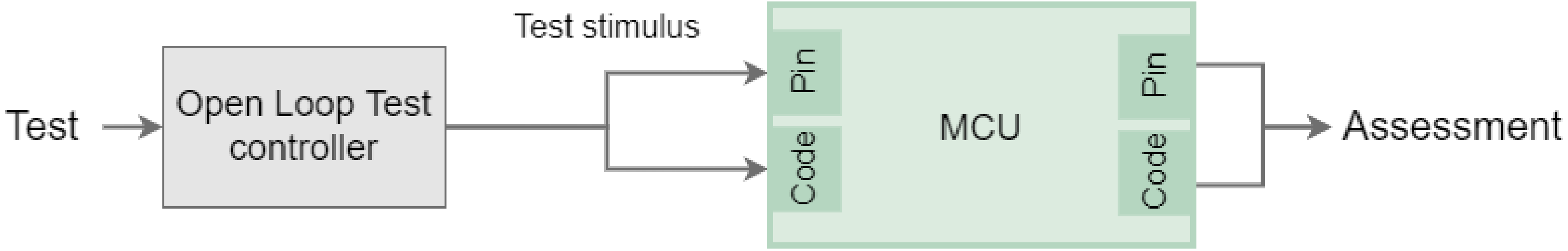

```
*** Test Cases ***
Write Transfer Contains Correct Data
    Dut Invoke    i2c_write    123
    ${writeTransfers} =    I2C0 Get Write Transfers
    Should Be Equal As Strings    ${writeTransfers[0].data}    123
```

Live Demo

TDD for using a I2C communication



Recall Example (ExecutionPlatform)



Code

Pin

```
*** Test Cases ***
Change sys_clk, ensure that i2c freq is still ~200kHz
    Dut Invoke    halve_clk_sys
    Dut Invoke     write    b
    ${clock_kHz} =     Get Observed I2C Clock in kHz
    Should Be True     ${clock_kHz} > 180 and ${clock_kHz} < 220
```

Open Loop Testing

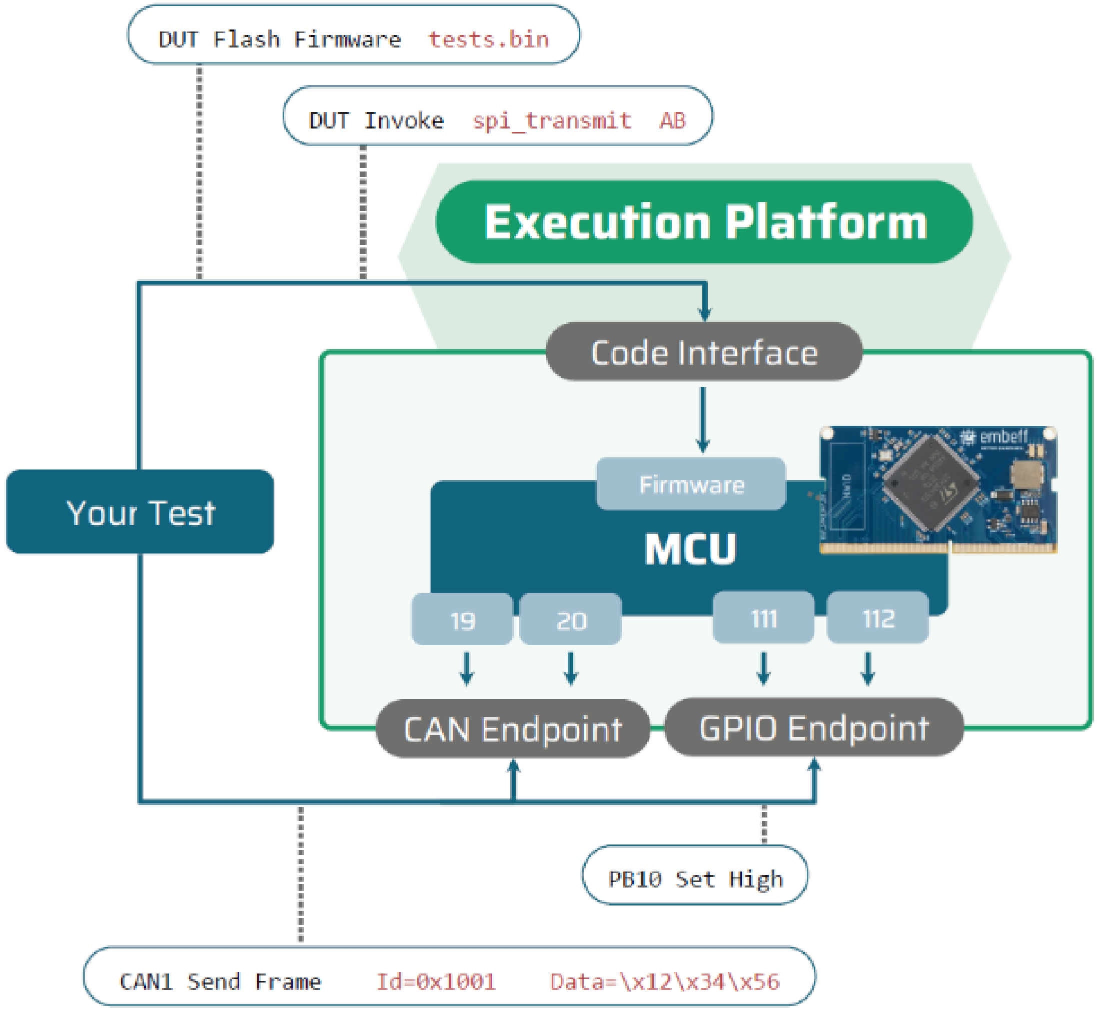
Setup

- Make pins accessible.
- Make code accessible.
- Make periphery accessible.

Steps for each test

- Write code and compile it for your Target.
- Write a test on your Machine.
- Run the test on your Machine.

TDD Requirement	Open Loop Testing
Automated feedback	✅ (special setup required)
Quick feedback	✅ (<10s, build+flash)



embeff BETTER EMBEDDED.

Thanks for listening!

Recommended reading

[1] Test Driven Development: By Example. Kent Beck, 2002

[2] Test Driven Development for Embedded C. James W. Grenning, 2011

[3] (German) Versteckte Risiken beim Kompilieren von Embedded Software. Heise Online

[4] Commercial Open Loop Tests: ExecutionPlatform. https://embeff.com

We want to support interesting embedded open source projects. You know one that could benefit from automatic testing & live browser exploring? Let us know.

Contact me
daniel.penning@embeff.com
Phone +49 (451) 16088698

TDD for MCUs is possible!

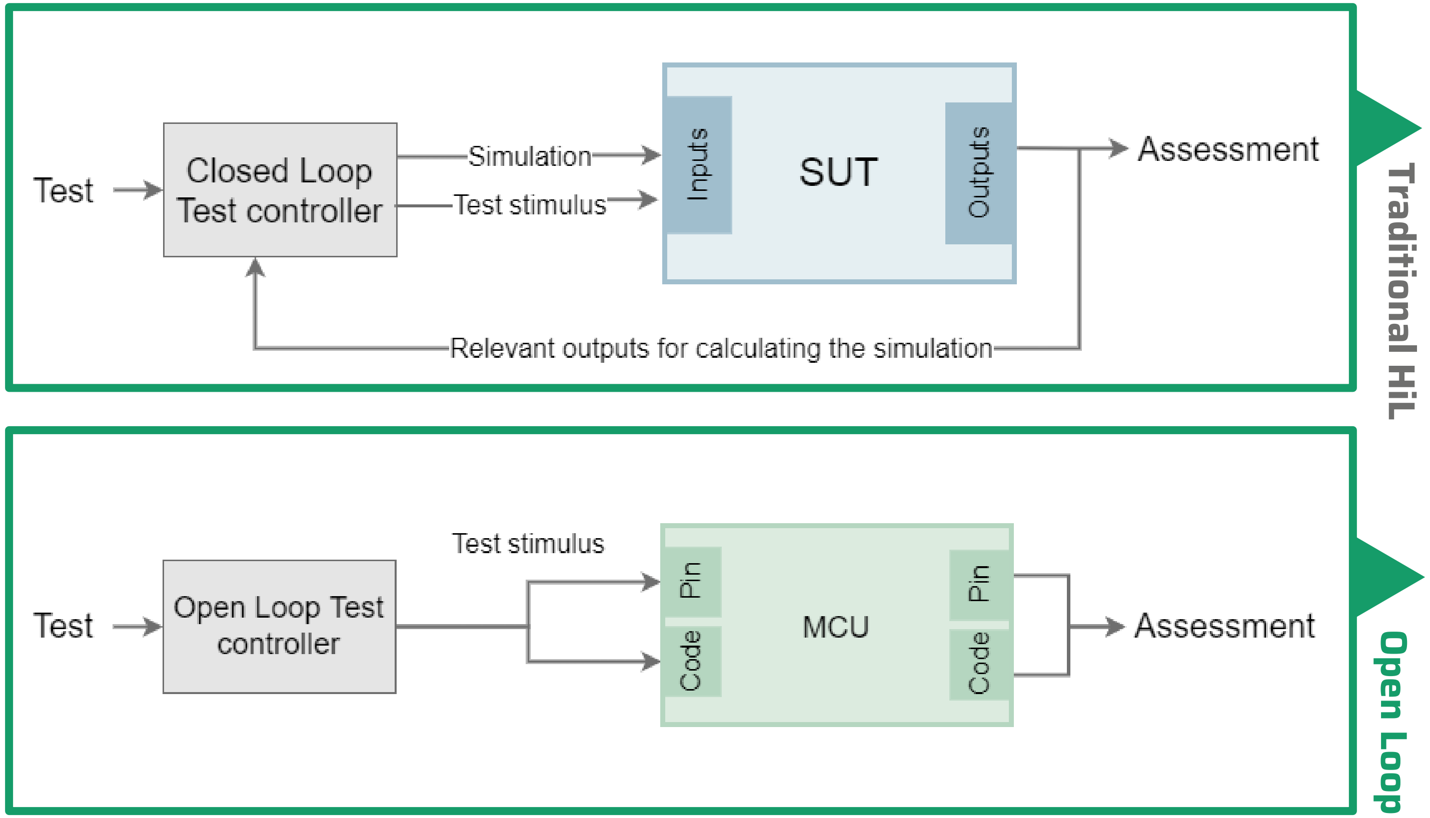
TDD Requirement	Off-Target	On-Target	Open Loop
Automated	yes	special setup	special setup
Quick feedback	<3s (build)	<10s (build+flash)	<10s, (build+flash)
Scope	Generic code	Internal periphery	External periphery

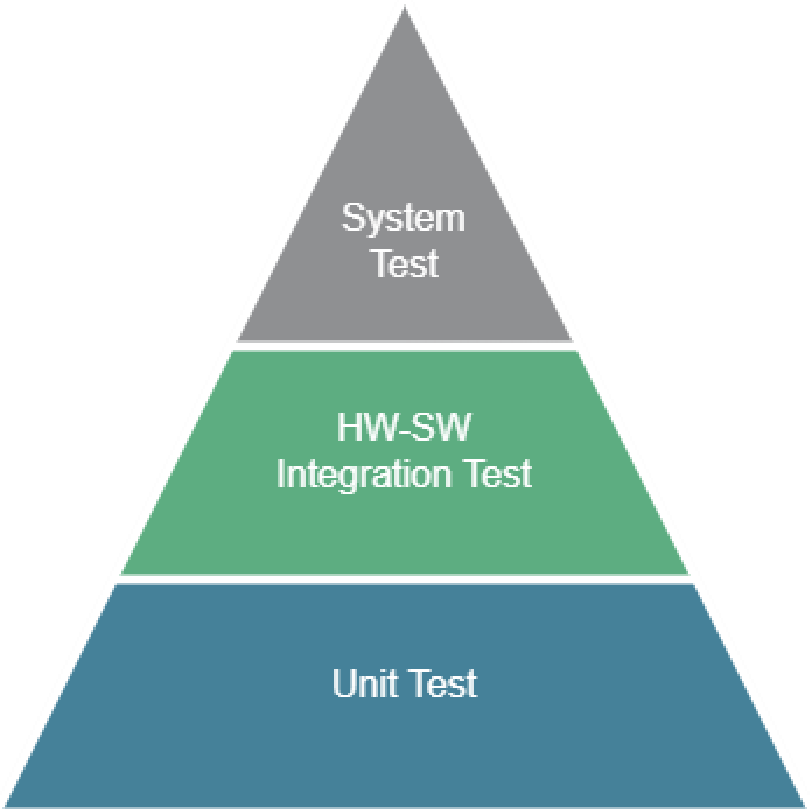
Slides

Start demo

embeff BETTER EMBEDDED.

Overview: Open Loop Tests






```
*** Test Cases ***
Write Transfer Contains Correct Data
    Dut Invoke    i2c_write    123
    ${writeTransfers} =    I2C0 Get Write Transfers
    Should Be Equal As Strings    ${writeTransfers[0].data}    123
```

embeff
BETTER EMBEDDED.