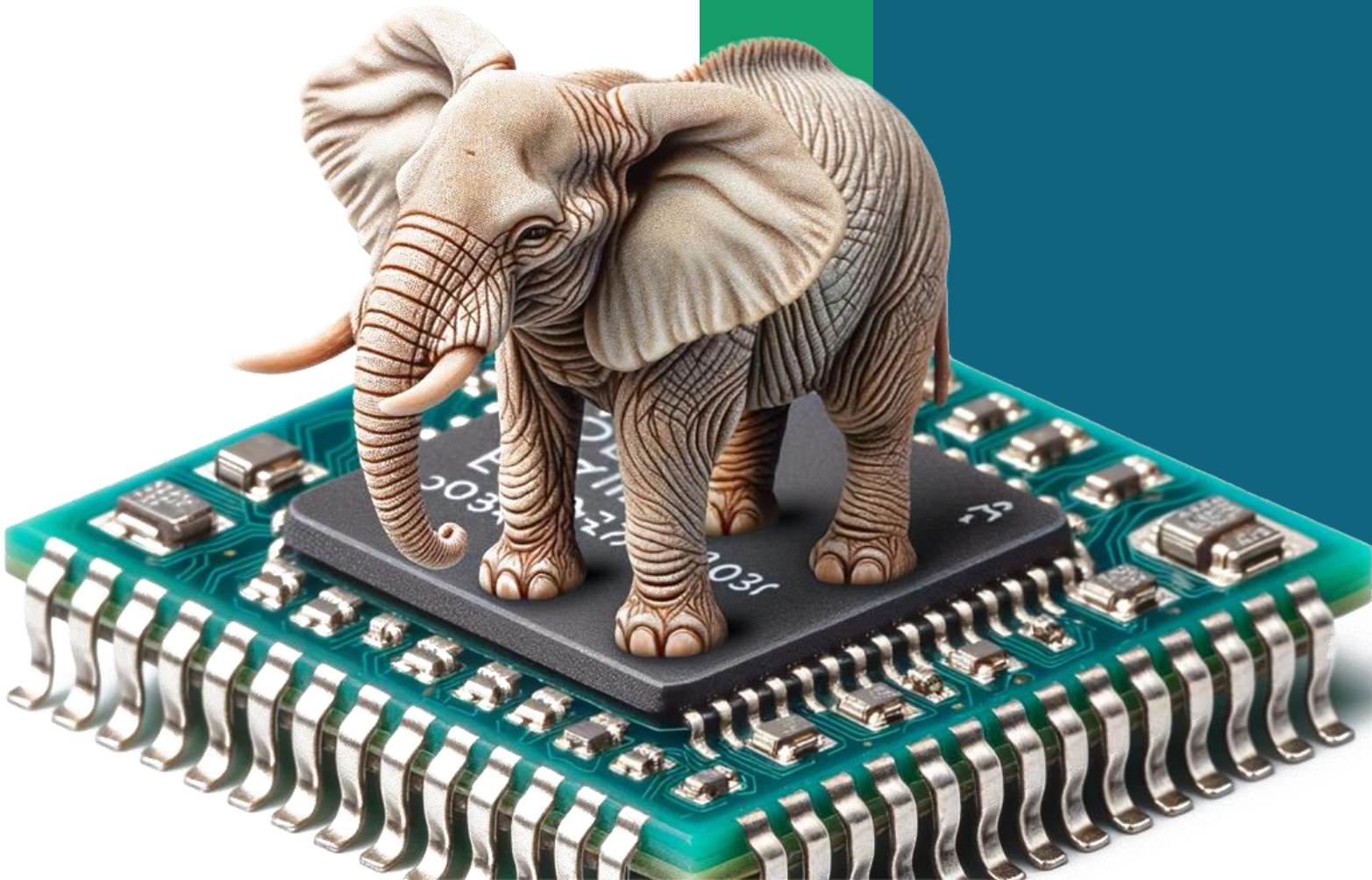
embeff

BETTER EMBEDDED.

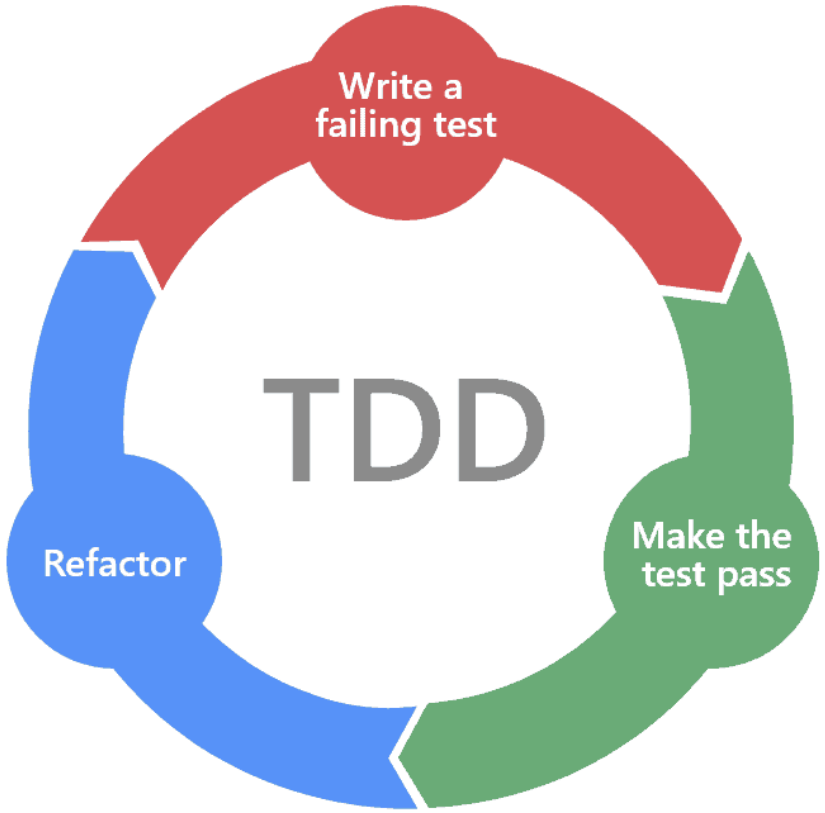
TDD und Mikrocontroller

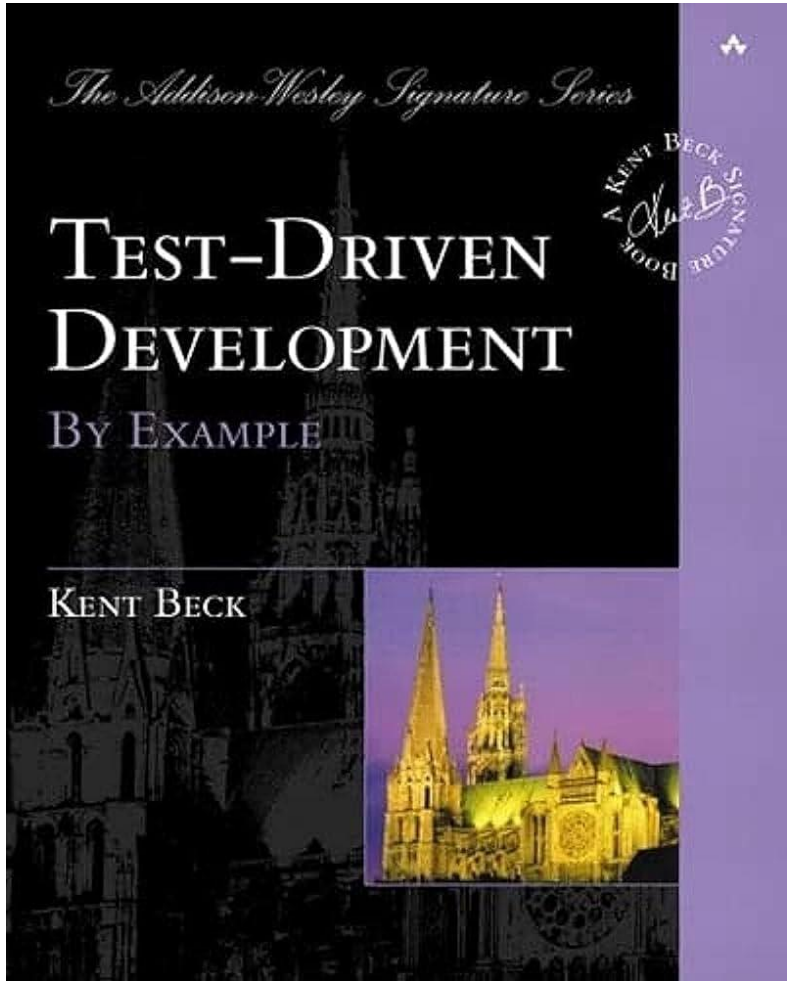


Ein praxistauglicher
Test-First-Ansatz

Daniel Penning, embeff GmbH

Der Kern von TDD





TDD benötigt schnelles und automatisiertes Feedback.

Wie bekommt man Feedback?

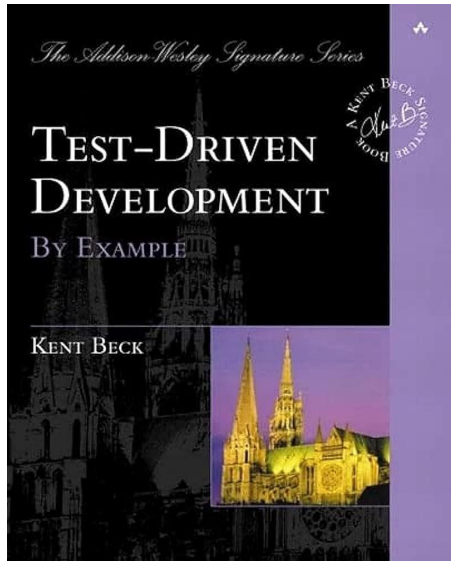
Klasische Software

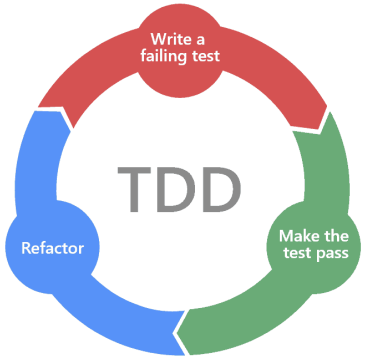
- Programm ausführen
- Code isoliert testen
  - Abhängigkeiten entkoppeln
  - Unit Tests

Embedded

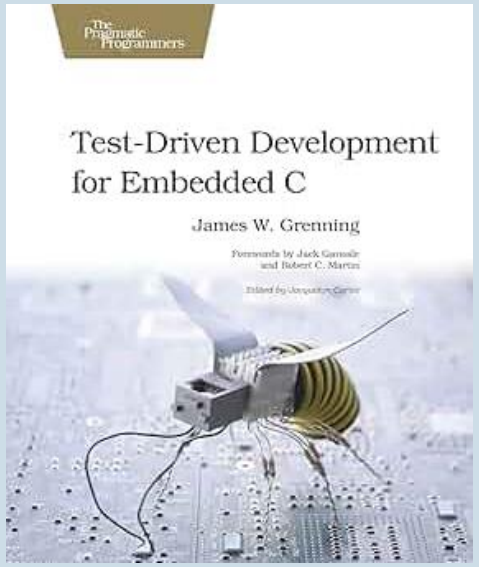
- Das Binary ausführen?
  - Mikrocontroller flashen
  - Sensoren / Geräte anschließen
- Code isoliert testen?

Der Weg zu TDD für Mikrocontroller





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. Code+Test entwickeln
2. Cross-Kompilieren
4. Resultate anschauen

Target

3. Tests ausführen

Host

1. Code+Test entwickeln
2. Kompilieren
3. Tests ausführen
4. Resultate anschauen

“On-Target“ Tests

- „langsam“
- schwer zu automatisieren

“Off-Target“ Tests

- schnell
- einfach zu automatisieren

Off-Target Testing: Code isolieren (1/2)

```
#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: Code isolieren (2/2)

```cpp
#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;
}
```

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

Off-Target Testing

Host

1. Test+Code entwickeln
2. Kompilieren
3. Tests ausführen
4. Resultate anschauen

Einrichtung

1. Separate Toolchain für Entwickler-PC aufsetzen. (gcc, clang, MSVC).
2. Ein übliches Unit Test Framework integrieren. (Googletest, Catch2, Unity).

Schritte für jeden Test

1. Code von allen Hardware-Abhängigkeiten entkoppeln.
2. Test schreiben und Test+Code für Entwickler-PC kompilieren.
3. Test ausführen.

TDD Anforderung	Off-Target Testing
Automatisches Feedback	✓
Schnelles Feedback	✓ (<3s, Build)

Off-Target: Risiken

```
#include <iostream>

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

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

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 (Fortsetzung)

Programmiersprache

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-abhängiger 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);
}
```

Können wir dafür Tests schreiben?

On-Target Testing

Host

1. Code und Tests entwickeln
2. Cross-kompilieren
4. Resultate anschauen

Target

3. Tests ausführen

Einrichtung

- Ein Unit Test Framework (Googletest, Catch2, Unity).
- Für Mikrocontroller adaptieren. Resultate ausgeben (bspw. per UART).
- Ausgaben am PC einlesen.

Schritte für jeden Test

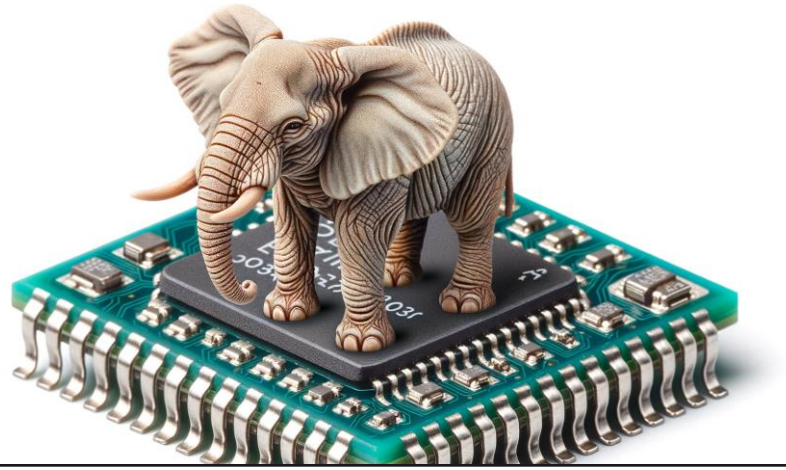
- Unit Test schreiben und Test+Code kompilieren.
- Mikrocontroller flashen.
- Binary ausführen (= Tests starten).

TDD Anforderung	On-Target Testing
Automatisches Feedback	✅ (spezielle Einrichtung nötig)
Schnelles Feedback	✅ (<10s, Build + Flashen)

Pin-Verhalten testen: “State of the art” (2023)

HAL und Treiber-Code arbeitet auf Pins.
Wie kann solcher Code getestet werden?

- Manuelles Testen.
- Register-basiertes Testen.



```
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);
```

embeff
BETTER EMBEDDED.

Register-basiertes Testen

Kern-Idee:

- Reference Manual ist Spezifikation zwischen SFR und Pin-Verhalten.
- Statt Pin-Ebene zu testen, prüfen ob HAL korrekte SFR-Zugriffe durchführt.

```
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, /* ???? */);
}
```

Register-basiertes Testen: Erwartete Werte finden

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.

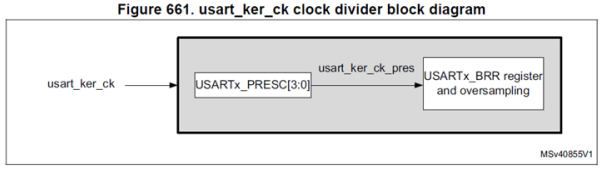
BRR[3] must be kept cleared.

2

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

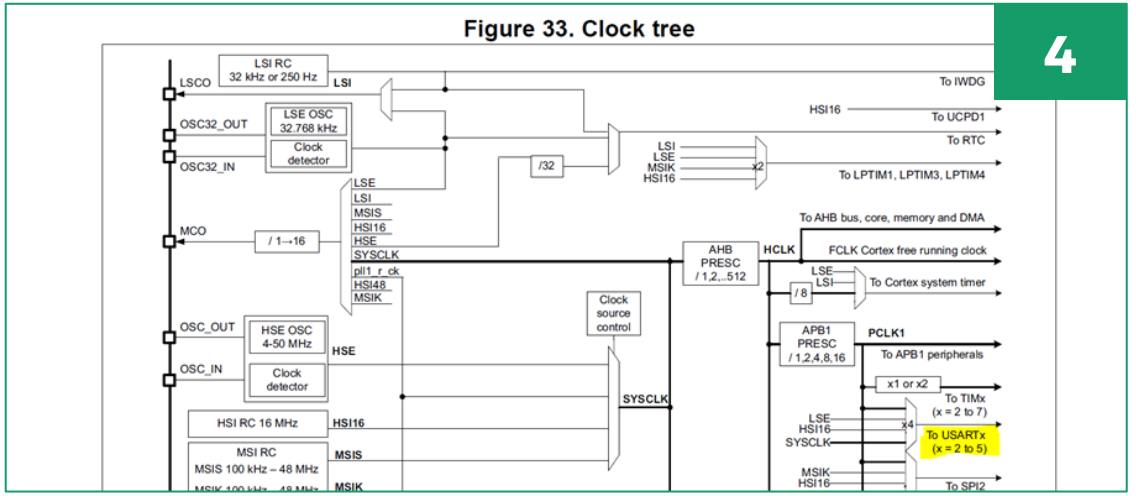
3

Figure 661. usart_ker_ck clock divider block diagram



4

Figure 33. Clock tree



...

embeff
BETTER EMBEDDED.

Register-basiertes Testen

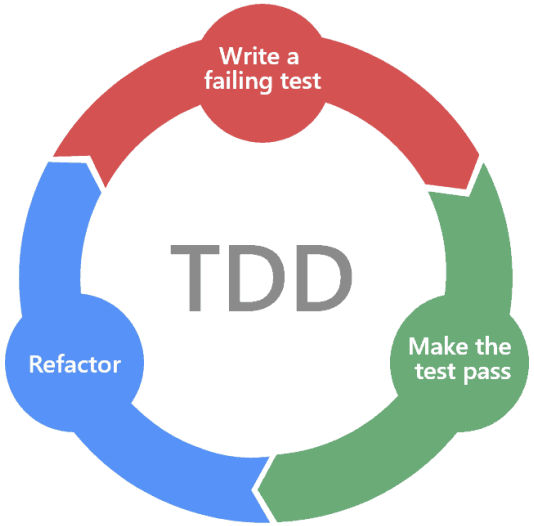
Kern-Idee:

- Reference Manual ist Spezifikation zwischen SFR und Pin-Verhalten.
- Statt Pin-Ebene zu testen, prüfen ob HAL korrekte SFR-Zugriffe durchführt.

Wie es TDD verhindert

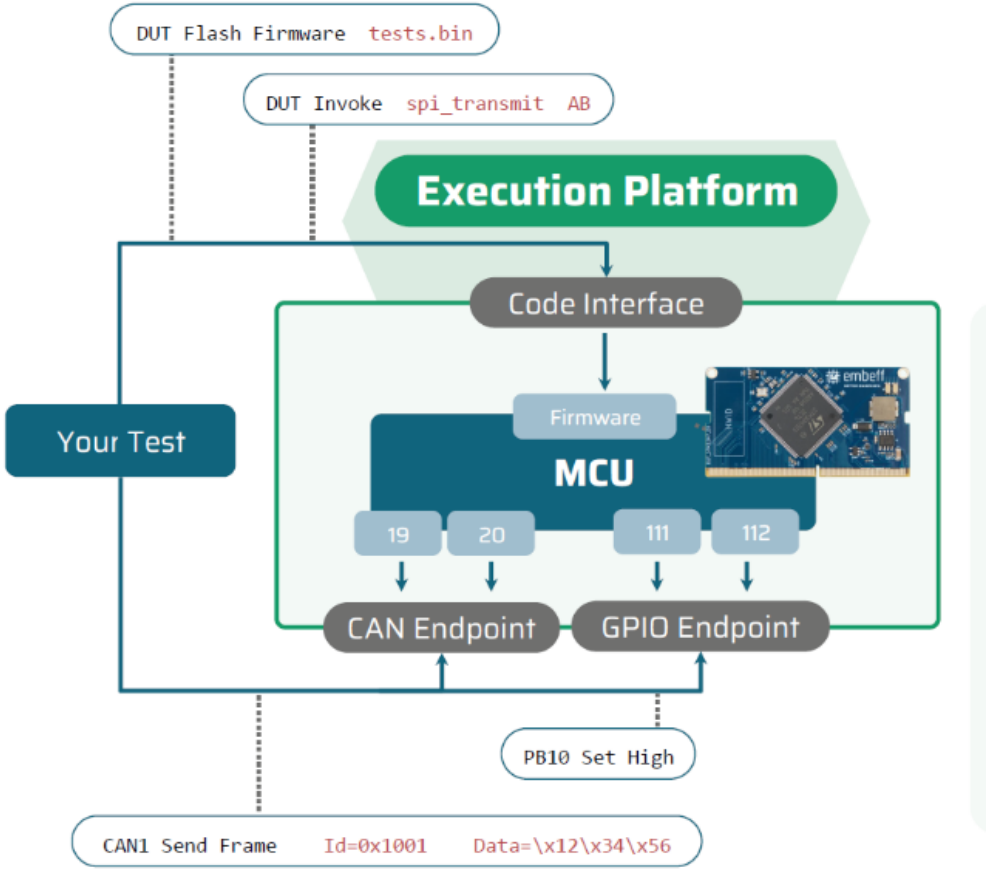
1. **Was sind die “korrekten” Register-Werte?**
   - Heutige Mikrocontroller sind enorm komplex.
   - Abhängigkeiten zwischen Peripherien.

2. **Whitebox**
   - Prüft eine spezifische Implementierung.
   - TDD-Refactoring wird Implementierung häufig ändern.



Pin-Verhalten mit einbeziehen: Open Loop Tests

1. **Pins verfügbar machen**
   - Exakten Mikrocontroller verwenden.
   - Gleiche Grundbeschaltung verwenden.
   - Relevante Pins nach außen verfügbar machen.
2. **Code zugreifbar machen**
   - Firmware automatisch flashen.
   - Funktionen in der laufendne Firmware aufrufen.
3. **Pins zugreifbar machen**
   - Schnittstellen überwachen und Kommunikation auslesbar machen.
   - Aktives Senden auf Schnittstellen ermöglichen.

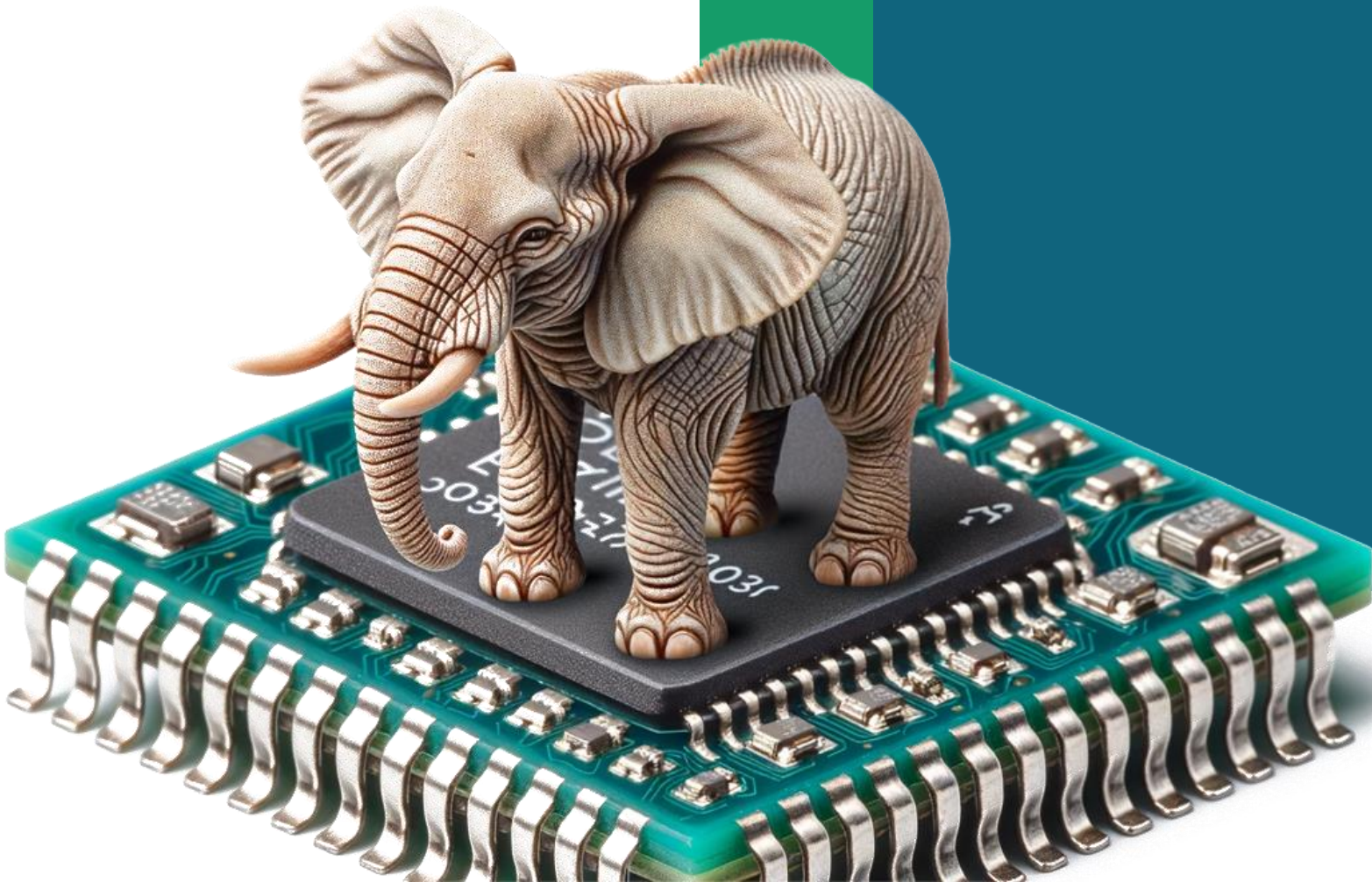



```
*** Test Cases ***
MCU UART can receive 7-Bit Maximum + 1
    UART2 Start         Bitrate=115.2kbit/s    Parity=Odd    StopBits=1    DataBits=8
    UART2 Transmit      Data=\x80
    Dut Invoke          rx_buf_print
    ${received} =       Dut Get Output
    Should Be Equal     RX (hex): 80     ${received[0]}
```

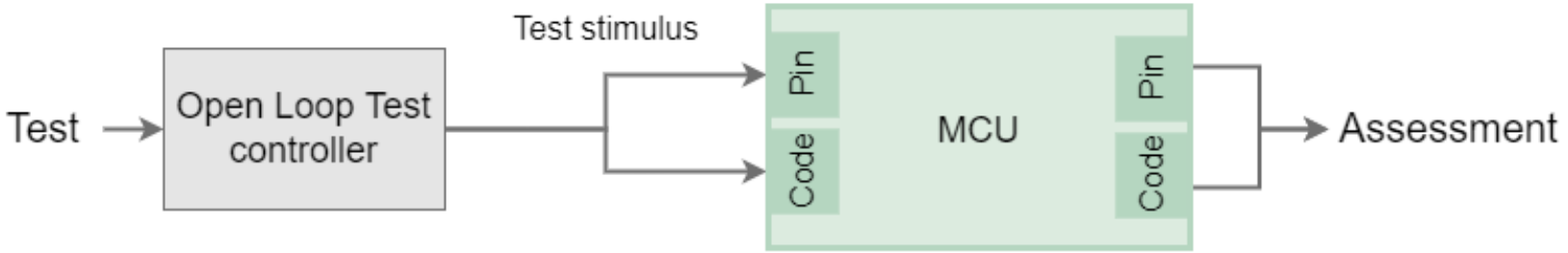

embeff
BETTER EMBEDDED.

Live Demo

TDD für UART Empfang



Open-Loop: Beispiel mit der ExecutionPlatform



Pin

Code

```
*** Test Cases ***
Receive 7-Bit Maximum + 1 Value
    UART2 Start    Bitrate=115.2kbit/s    Parity=Odd  StopBits=1  DataBits=8
    # Transmit a single byte 0x80
    UART2 Transmit    Data=\x80

    # Call function on microcontroller that outputs all received data
    Dut Invoke    rx_buf_print
    ${received} =    Dut Get Output
    Should Be Equal    RX (hex): 80    ${received[0]}
```

Open Loop Testing

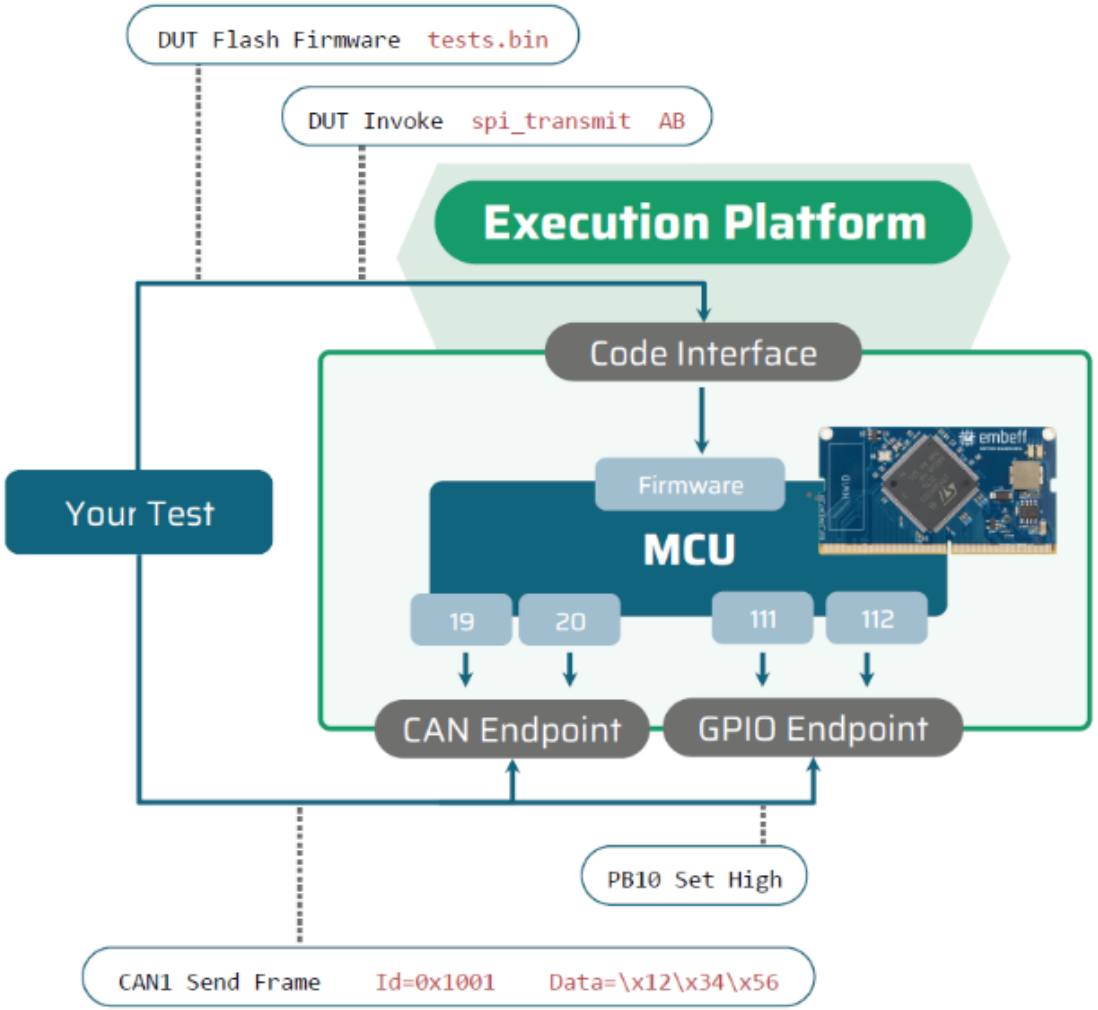
Setup

- Pins verfügbar machen.
- Code zugreifbar machen.
- Pins zugreifbar machen.

Schritte für jeden Test

- Code schreiben und für Mikrocontroller kompilieren.
- Test auf dem Entwickler-PC schreiben.
- Test auf dem Entwickler-PC ausführen.

TDD Anforderung	Open Loop Testing
Automatisches Feedback	✅ (spezielles Setup erforderlich)
Schnelles Feedback	✅ (<10s, Build+Flash)



embeff
BETTER EMBEDDED.

Danke für die Aufmerksamkeit!

Empfohlene Literatur

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

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

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

[4] Kommerzielles Open Loop Testsystem: ExecutionPlatform. https://embeff.com

Direkter Kontakt
daniel.penning@embeff.com
Phone +49 (451) 16088698

Credits: Some images were created with the assistance of DALL·E 3.

TDD for µController ist möglich!

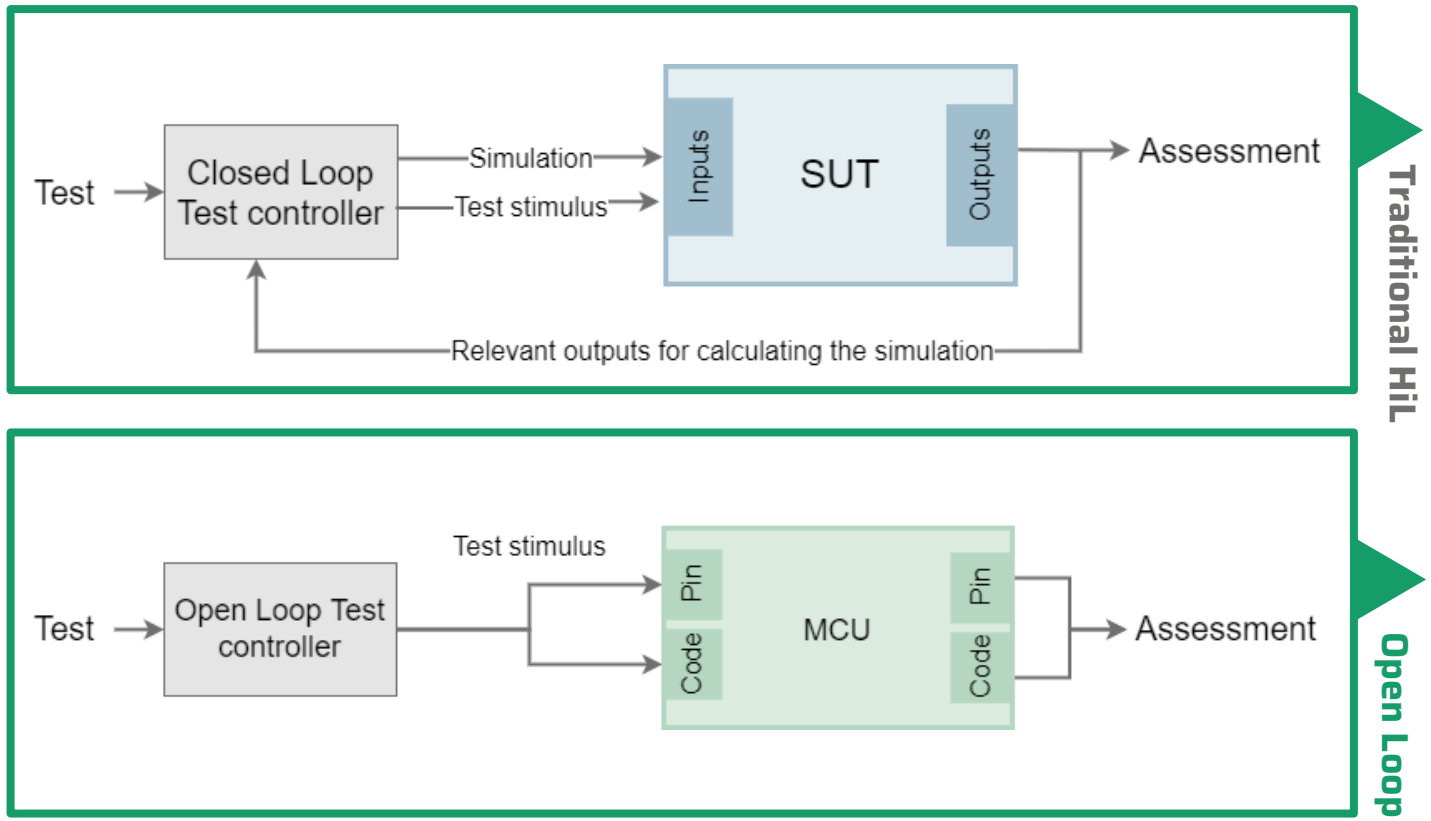
TDD Anforderung	Off-Target	On-Target	Open Loop
Automatisiert	ja	spezieller Aufbau	spezieller Aufbau
Schnelles Feedback	<3s (Build)	<10s (Build+Flash)	<10s, (Build+Flash)
Code-Rahmen	nur generisch	interne Peripherie	externe Peripherie

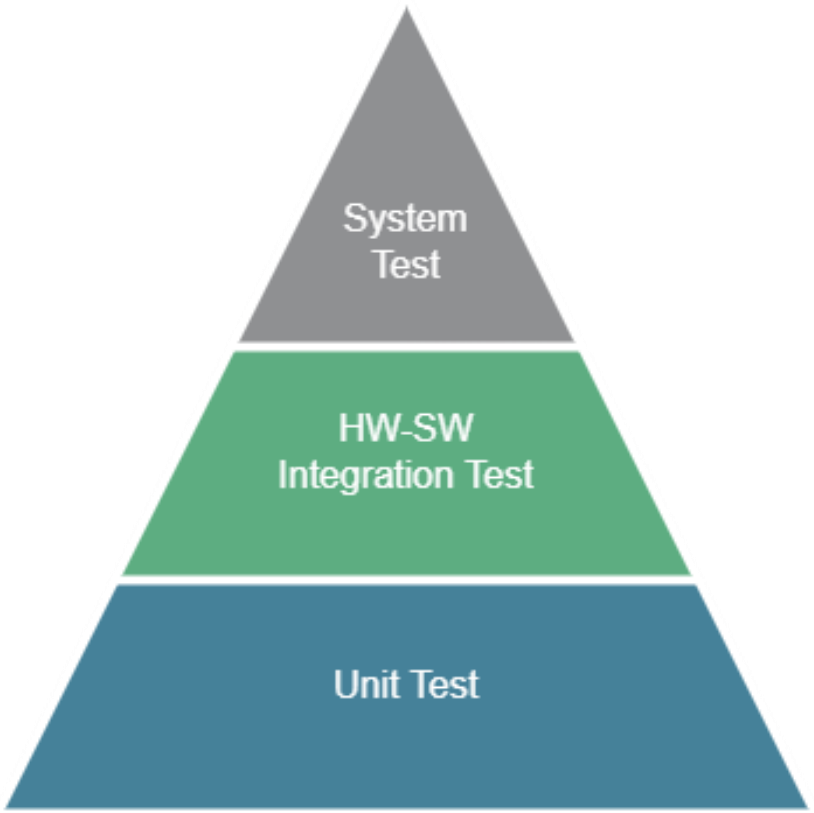
Folien

Demo starten

embeff
BETTER EMBEDDED.

Überblick: Open Loop Tests






```
*** Test Cases ***
MCU UART can receive 7-Bit Maximum + 1
    UART2 Start         Bitrate=115.2kbit/s    Parity=Odd    StopBits=1    DataBits=8
    UART2 Transmit      Data=\x80
    Dut Invoke          rx_buf_print
    ${received} =       Dut Get Output
    Should Be Equal     RX (hex): 80    ${received[0]}
```

embeff
BETTER EMBEDDED.