embeff
BETTER EMBEDDED.

Was wird nur aus meinem Code?

Software-Performance endlich fundiert bewerten

@DanielPenning
www.embeff.com

Weltkarte
(1459)

Weltkarte (1525)

embeff
BETTER EMBEDDED.

Menschen haben ihr Unwissen eingestanden.

Wissenschaftliche Revolution

1. Ignoranz eingestehen

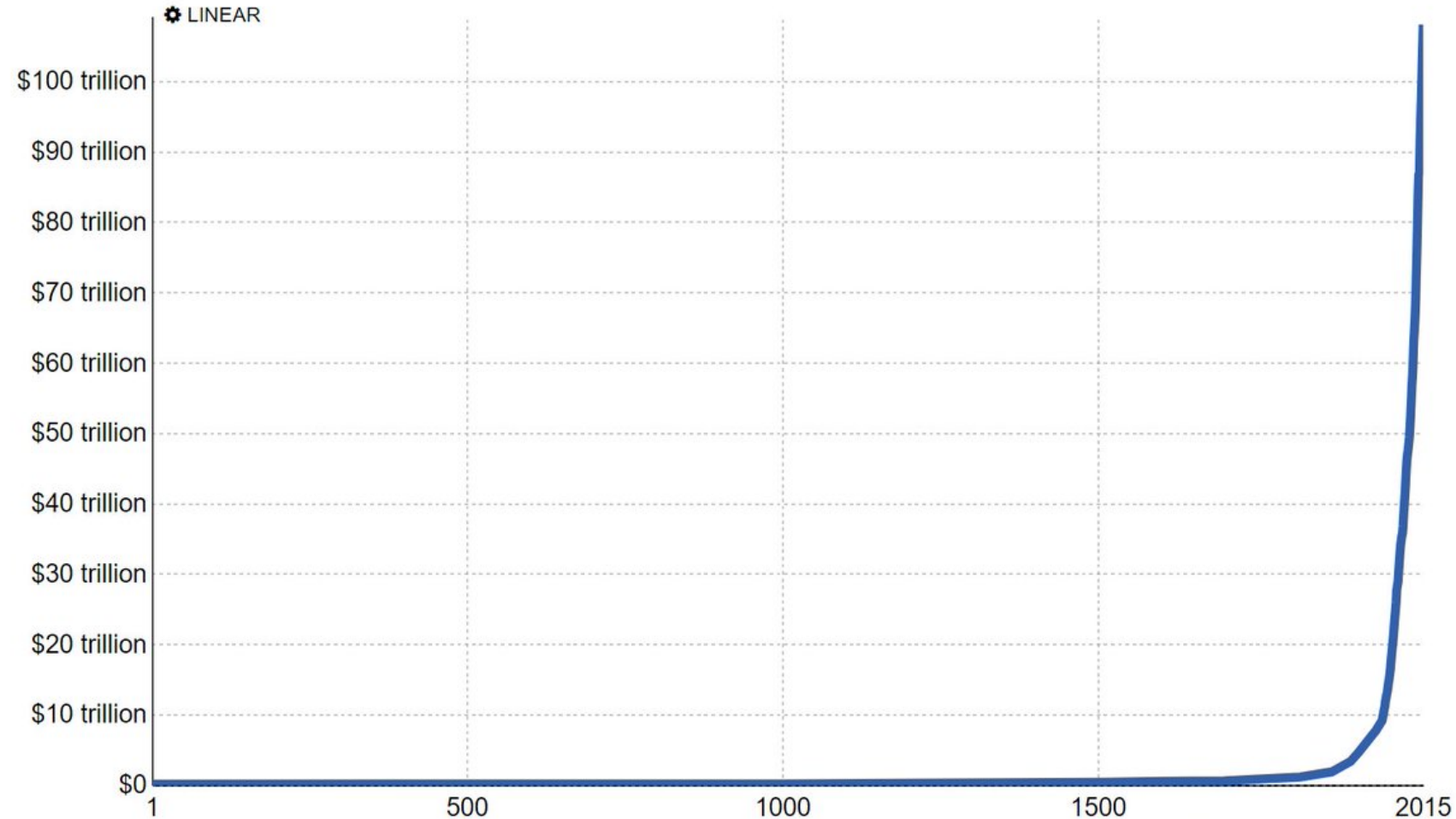
> **ignorance** | ˈɪgn(ə)r(ə)ns |
>
> noun *[mass noun]*
>
> lack of knowledge or information: *he acted in **ignorance of** basic procedures*.

2. Messungen
   - Daten sammeln.
   - Aus Daten Hypothesen ableiten.

@DanielPenning
www.embeff.com

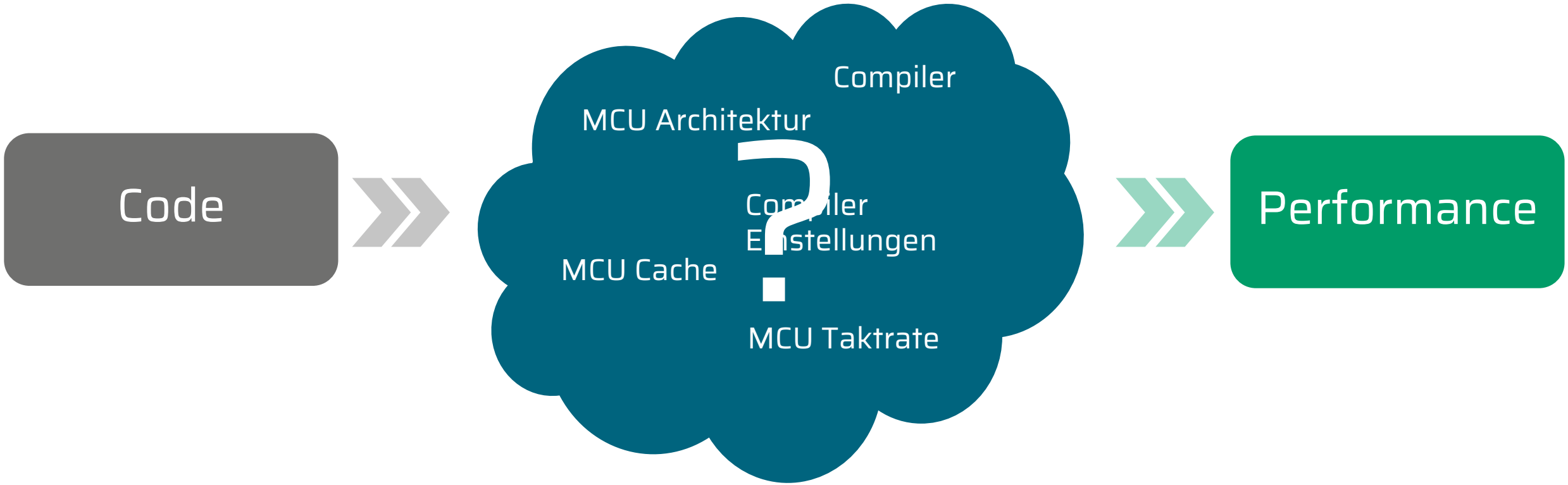
World GDP over the last two millennia

Total output of the world economy; adjusted for inflation and expressed in 2011 international dollars.



Source: World GDP (Our World In Data based on World Bank & Maddison)

Embedded & Ignoranz



Konsequenzen

Vorurteile

Misstrauen gegenüber jeglicher Abstraktion

Niedrige Softwarequalität

Schlechte Performance

embeff
BETTER EMBEDDED.

Unwissenheit eingestehen!

Embedded & Messungen

Profiling

- Top Down Prozess.
- Gut um Flaschenhälse zu lokalisieren.
- Schlecht um spezifisches Verständnis aufzubauen.

Wissen Bottom-Up aufbauen

- Mit kleinen Code-Blöcken beginnen.
- Performance messen.
- Hypothesen aufstellen.

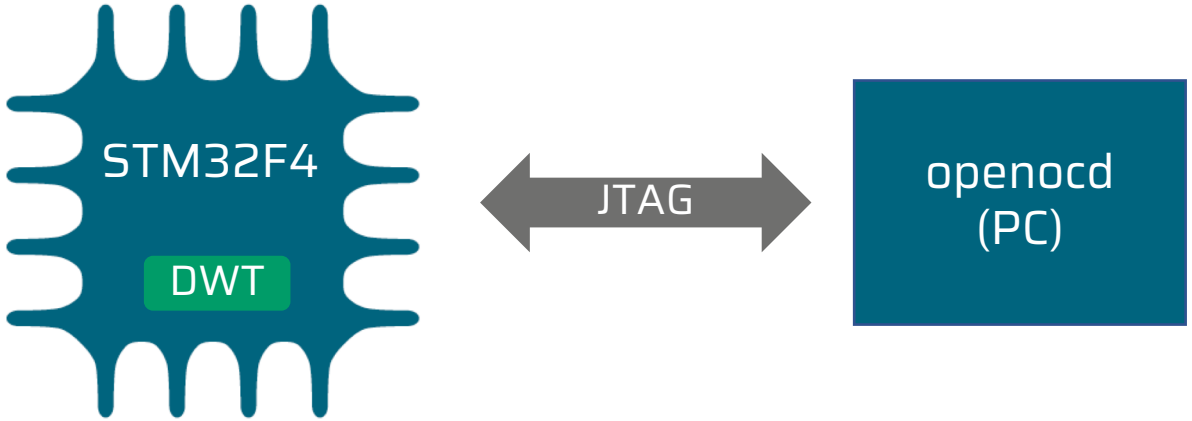
Code Performance für armv7m

Architektur weit verbreitet (Cortex-M3/M4)

Definiert **D**ata **W**atchpoint and **T**race Unit

CMSIS Register	Beschreibung
DWT_CYCCNT	Cycle Count Register
DWT_CPICNT	CPI Count Register
DWT_EXCCNT	Exception Overhead Count Register
DWT_SLEEPCNT	Sleep Count Register
DWT_LSUCNT	LSU Count Register
DWT_FOLDCNT	Folded-instruction Count Register

Taktzyklen messen



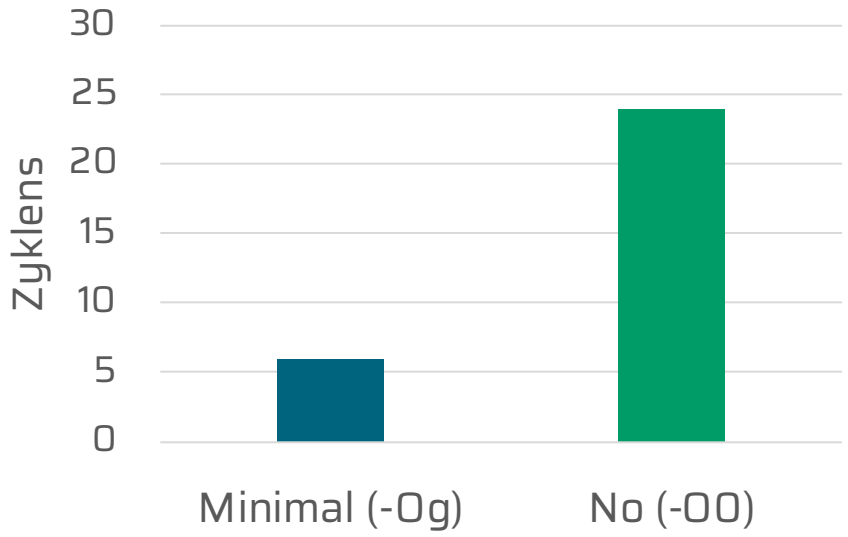

```
BKPT //< CYCCNT lesen
CodeUnderTest(<Parameter>)
BKPT //< CYCCNT lesen
```

embeff
BETTER EMBEDDED.

Messungen durchführen.

Beispiel 1: Optimierung

```
int square(int x) {
    return x*x;
}
```




```
square(int):
        mul     r0, r0, r0
        bx      lr
```

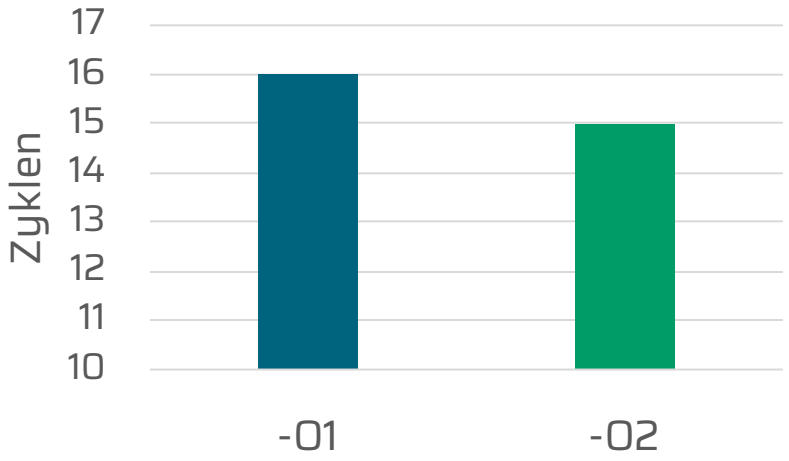
```
square(int):
        push    {r7}
        sub     sp, sp, #12
        add     r7, sp, #0
        str     r0, [r7, #4]
        ldr     r3, [r7, #4]
        ldr     r2, [r7, #4]
        mul     r3, r2, r3
        mov     r0, r3
        adds    r7, r7, #12
        mov     sp, r7
        ldr     r7, [sp], #4
        bx      lr
```

embeff
BETTER EMBEDDED.

Hypothese #1
Der Unterschied zwischen minimaler und keiner Optimierung ist riesig.

Beispiel 2: Pipeline

```
int DependentOps(int x) {
    int tmp = x/3;
    int tmp2 = x/7;
    return tmp+tmp2;
}
```



DependentOps_O1(int):

```
  ldr     r3, .L2
  smull   r2, r3, r3, r0
  asrs    r1, r0, #31
  subs    r3, r3, r1
  ldr     r2, .L2+4
  smull   ip, r2, r2, r0
  add     r0, r0, r2
  rsb     r0, r1, r0, asr #2
  add     r0, r0, r3
  bx      lr
.L2:
  .word   1431655766
  .word   -1840700269
```

DependentOps_O2(int):

```
  ldr     r3, .L3
  ldr     r1, .L3+4
  smull   r2, r3, r3, r0
  add     r3, r3, r0
  asrs    r2, r0, #31
  smull   r1, r0, r1, r0
  rsb     r3, r2, r3, asr #2
  subs    r0, r0, r2
  add     r0, r0, r3
  bx      lr
.L3:
  .word   -1840700269
  .word   1431655766
```

embeff
BETTER EMBEDDED.

Hypothese #2
In der Umsetzung zwischen einzelnen Statements und Assembly ist der Compiler besser als Sie.

Beispiel 3: FPU vs Soft-FPU

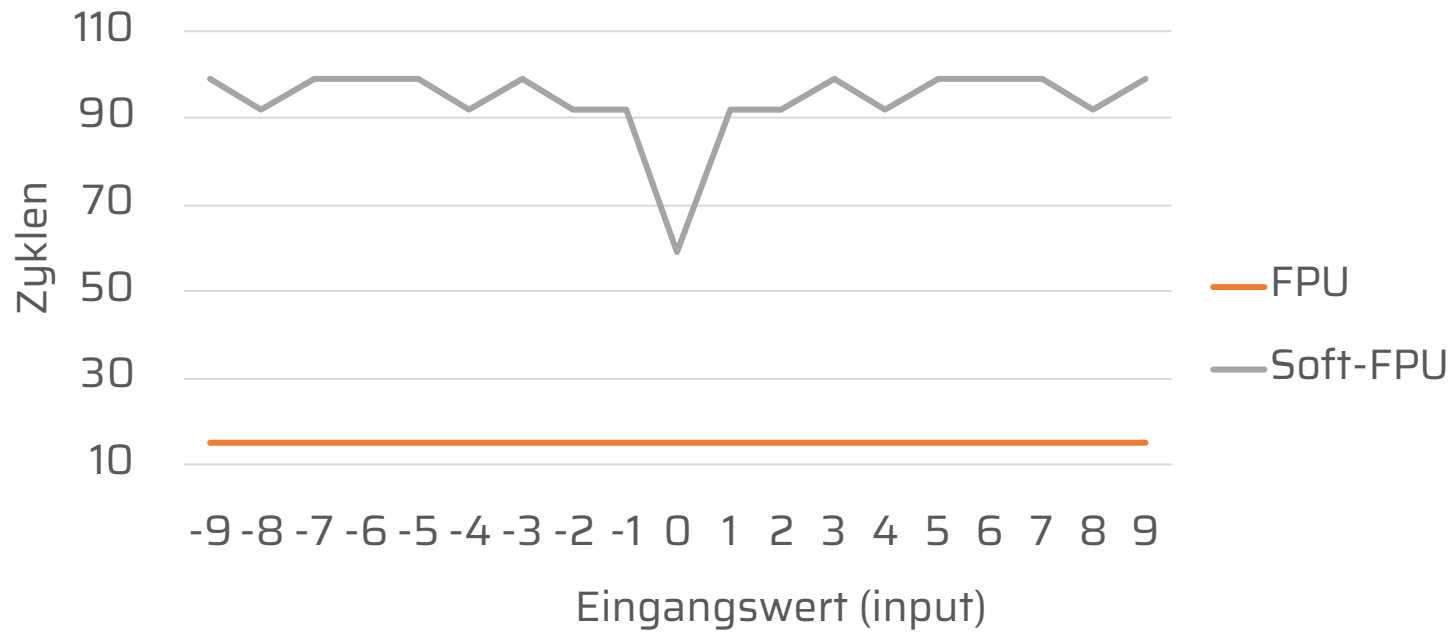
```
int MultiplyWithPi(int input) {
    return input * 3.14159265359f;
}
```

```
MultiplyWithPi_FPU(int):
  vmov    s15, r0 @ int
  vldr.32 s14, .L3
  vcvt.f32.s32    s15, s15
  vmul.f32        s15, s15, s14
  vcvt.s32.f32    s15, s15
  vmov    r0, s15 @ int
  bx      lr
.L3:
  .word   1078530011
```

```
MultiplyWithPi_SoftFPU(int):
  push    {r3, lr}
  bl      __aeabi_i2f
  ldr     r1, .L4
  bl      __aeabi_fmul
  bl      __aeabi_f2iz
  pop     {r3, pc}
.L4:
  .word   1078530011
```

Beispiel 3: FPU vs Soft-FPU

```
int MultiplyWithPi(int input) {
    return input * 3.14159265359f;
}
```



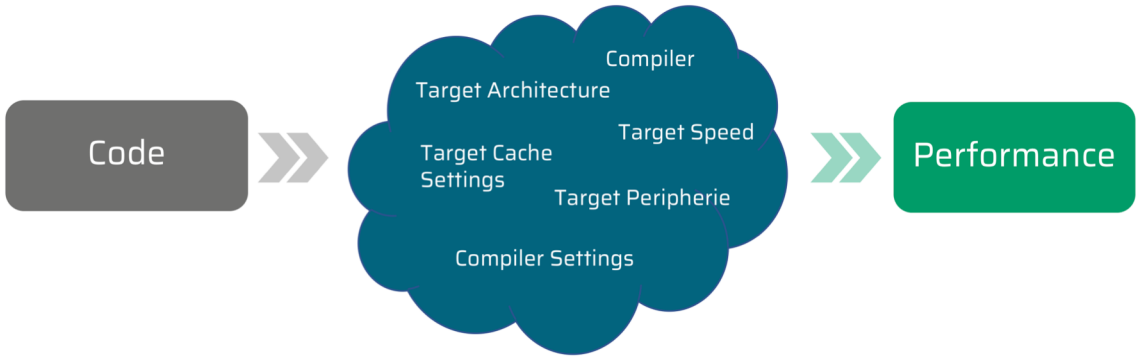
embeff
BETTER EMBEDDED.

Hypothese #3
Software-FPU ~ 6x langsamer und nicht deterministisch.

Example 4: CRC Berechnung

Cyclic Redundancy Check

- Direkte Berechnung
- Lookup-Tabelle
- Hardware-Unterstützung



Online Benchmarking

- Code auf echter Hardware ausführen und messen.
- Kostenfrei nutzbar.
- https://barebench.com

embeff
BETTER EMBEDDED.

barebench.com
- Demo -

embeff
BETTER EMBEDDED.

Hypothese #4
Performance *kann* von der Taktrate abhängen.

embeff
BETTER EMBEDDED.

Hypothese #5
Caching ist für hohe Taktraten essentiell.

Zusammenfassung

Unwissenheit eingestehen.

Performance messen.

Aus Messungen Hypothesen ableiten.

Hypothesen diskutieren und verbreiten.

Vorurteile daten-basiert widerlegen.

embeff
BETTER EMBEDDED.

Lassen Sie uns Embedded-Systeme besser machen!

Wir unterstützen Sie dabei mit Training und Beratung.