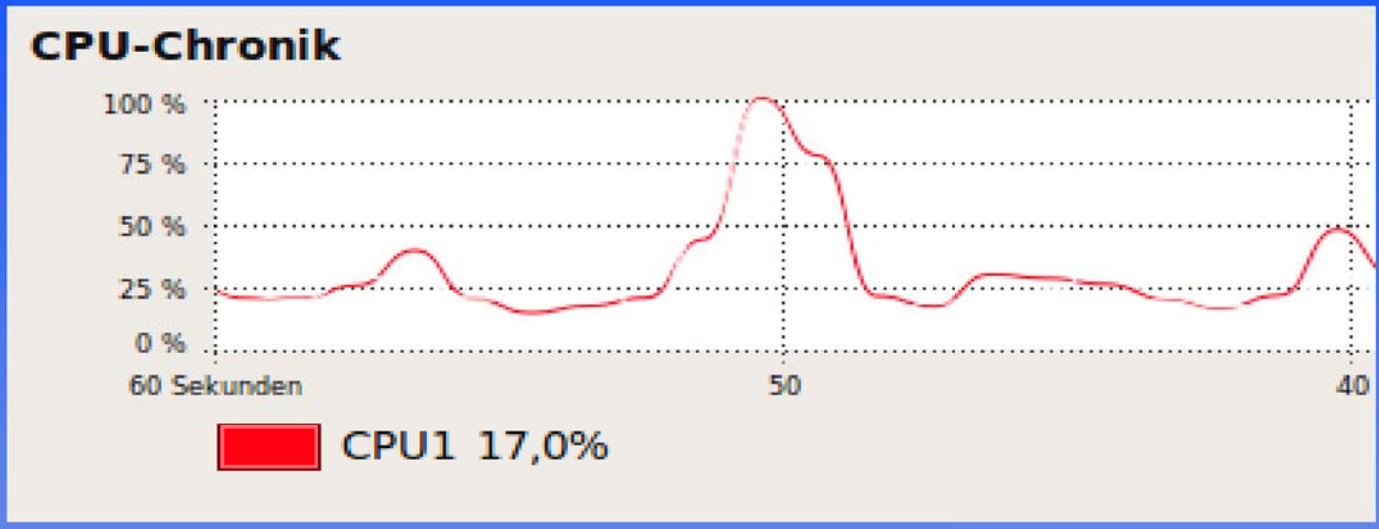
Performancemessungen und - Optimierungen an GNU/Linux Systemen

Wolfram Luithardt, Daniel Gachet und Olivier Nasrallah

Hochschule für Technik und Architektur, Fribourg, Schweiz

Embedded Systems

Motivation und einige wichtige Fragen



- Normalerweise benötigen Mikroprozessorsysteme nur sehr selten ihre volle Leistung. Dies geschieht hauptsächlich bei Eintreten von bestimmten Ereignissen (Events). Wie kann also erreicht werden, dass, wenn die Performance benötigt wird, diese auch vorhanden ist?

- Modernes event-getriggertes Programmdesign sollte monolithische Programme vermeiden und Multiprocessing oder zumindest Multithreading verwenden. In solchen Systemen kommt dem Scheduling natürlich eine massgebliche Rolle zu. Wann wird welcher Prozess aufgerufen? Wer bekommt wann welche Ressourcen zugeteilt ?

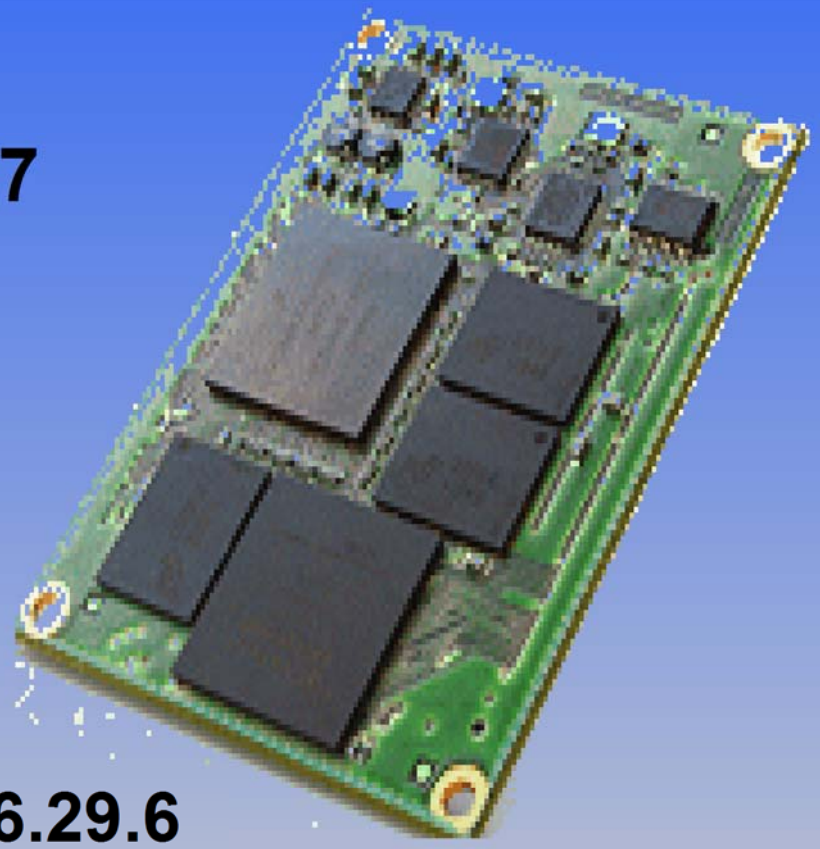
Das verwendete System

Plattform APF-27

www.armadeus.org

- Prozessor Freescale i.mx27
- Takt bis 400MHz
- 64 MB RAM
- 16 MB Flash
- Ethernet, USB etc...
- FPGA

....



Kernel 2.6.29.6

Wie kann man die Zeit messen?

DO rücksetzen

Routine

DOsetzen

Unix Zeitsystem

clock_t times(struct tms *buf);

```
struct tms {
    clock_t tms_utime;  /* user time */
    clock_t tms_stime;  /* system time */
    clock_t tms_cutime; /* user time of children */
    clock_t tms_cstime; /* system time of children */
};
```

```
typedef long clock_t;
```

Auflösung 10ms ist natürlich nicht ausreichend :-(

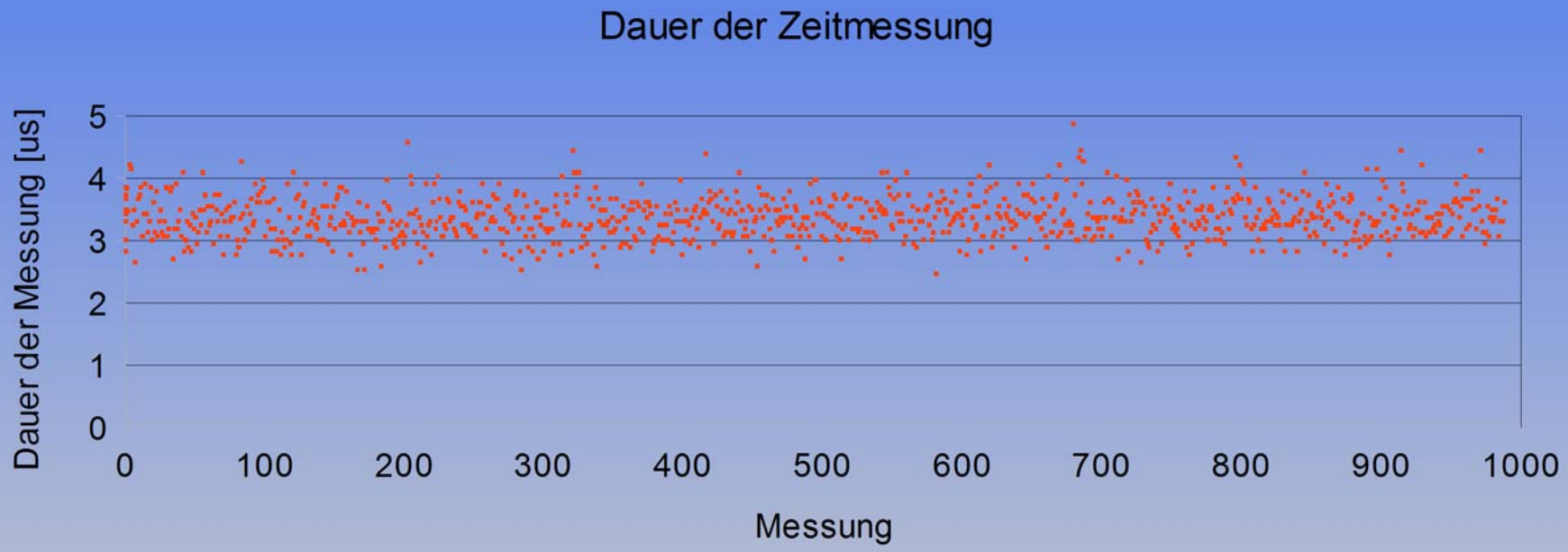
Ein moderneres Interface

```
int clock_gettime( clockid_t clock_id, struct timespec *ts);


cklockid_t:         CLOCK_REALTIME
                    CLOCK_MONOTONIC
                    CLOCK_PROCESS_CPUTIME_ID
                    CLOCK_THREAD_CPUTIME_ID

struct timespec {
        time_t    tv_sec;           /* seconds */
        long      tv_nsec;      /* nanoseconds */
}
```

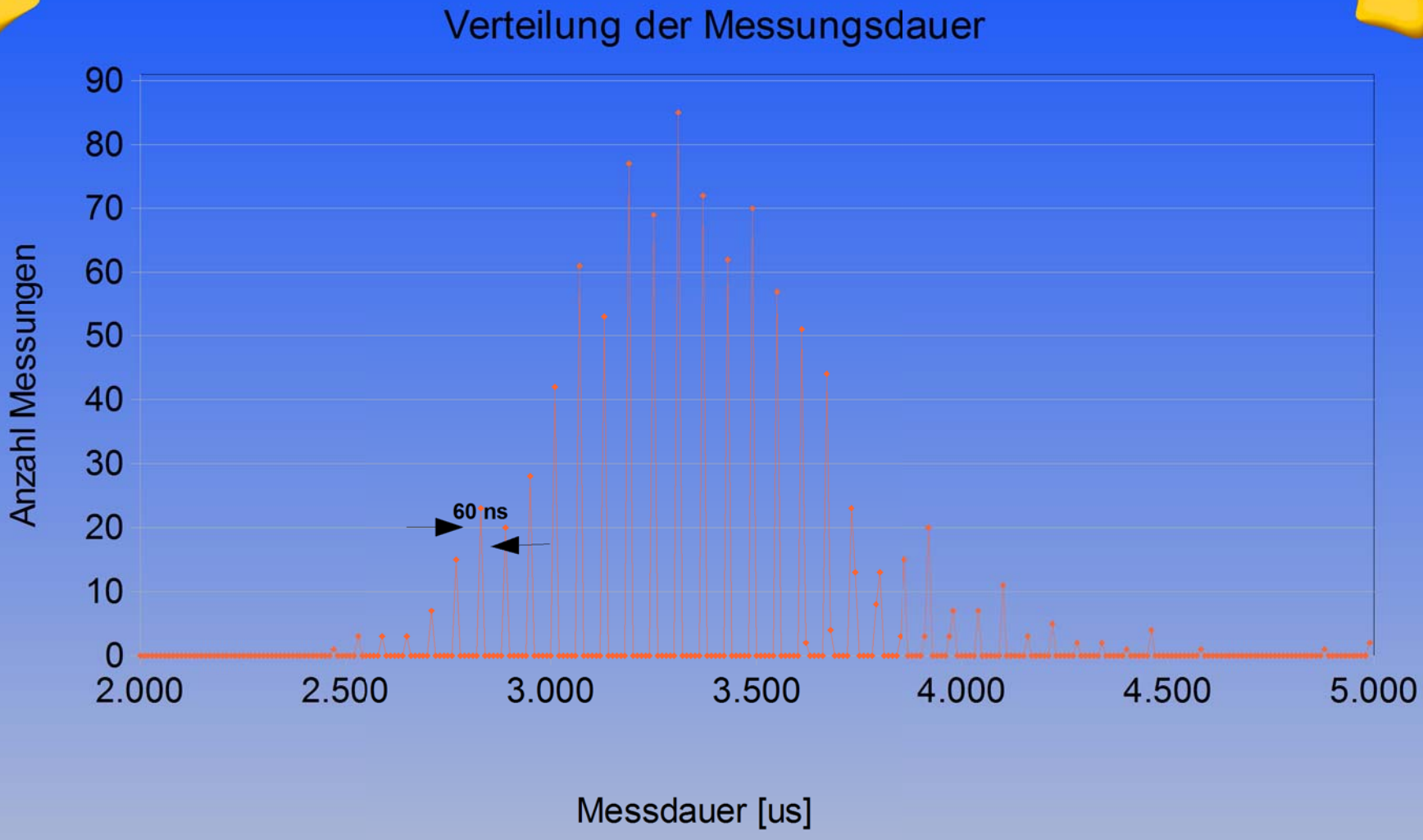
Achtung: Kompilieren mit der Realtime Library (-lrt) !

Wie lange benötigt die Zeitmessung?

```
struct timespec rt1, rt2;
long long time1;
clock_gettime(CLOCK_MONOTONIC, &rt1);
clock_gettime(CLOCK_MONOTONIC, &rt2);
time1 = (long long int)(rt2.tv_sec - rt1.tv_sec)*1000000000 + (rt2.tv_nsec - rt1.tv_nsec);
```



Wie lange benötigt die Zeitmessung?



Reale Auflösung: ca. 60 ns

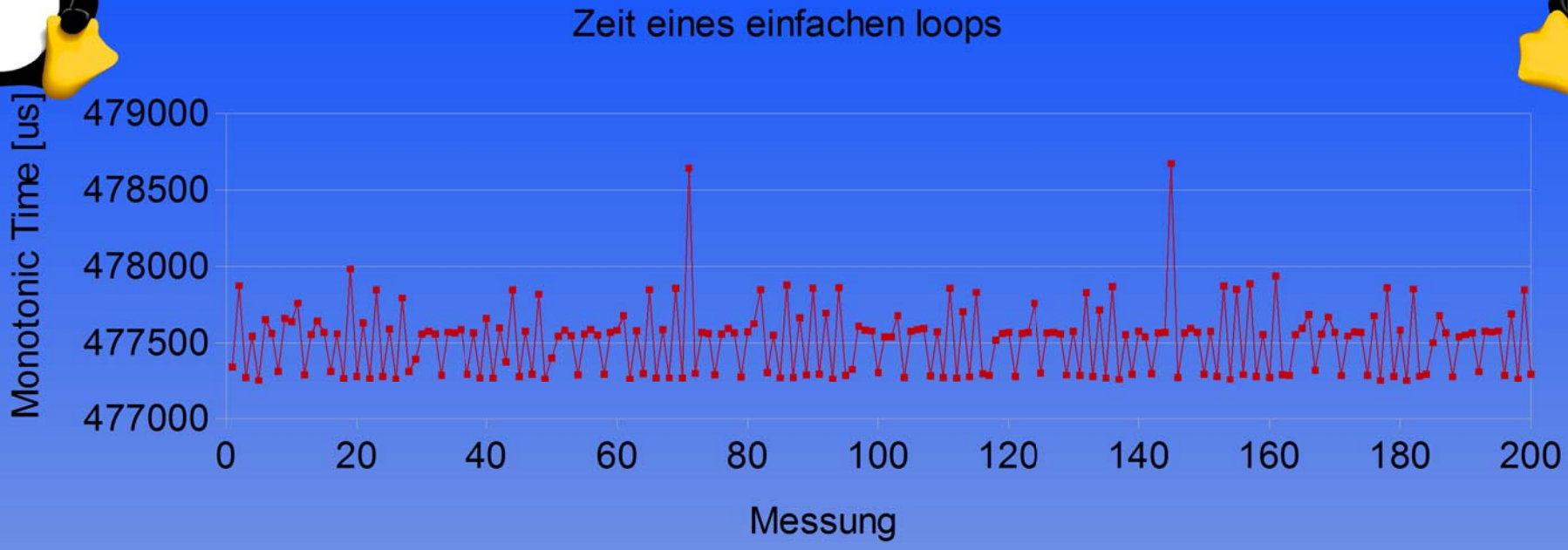
Einfacher Loop

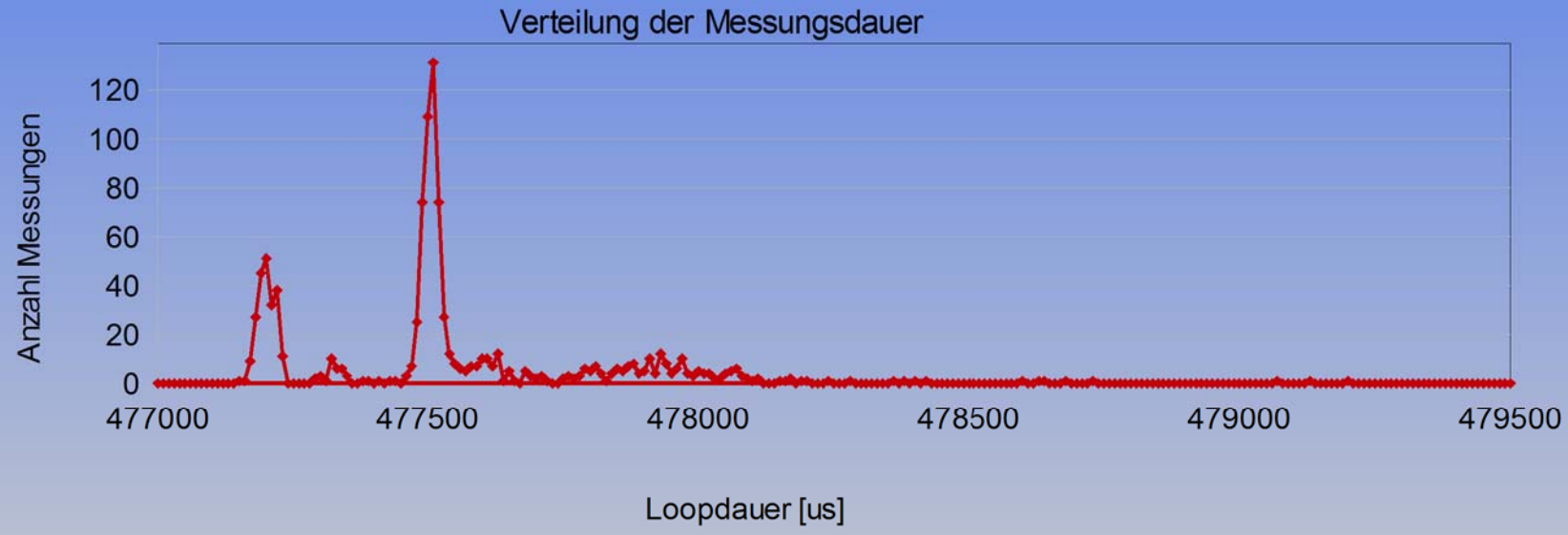
```
int main(){
static int counter = 0;
struct timespec rt1, rt2;
long long time1;
clock_gettime(CLOCK_REALTIME, &rt1);
while(counter<10000000){
    counter++;
}
clock_gettime(CLOCK_REALTIME, &rt2);
time1 = (long long int)(rt2.tv_sec - rt1.tv_sec)*1000000000 + (rt2.tv_nsec - rt1.tv_nsec);
printf("%.3f\n", (double)(time1)/1000);
exit(0);
}
```

Resultat: 477.5 ±0.5 ms ⟶ 45 ns/loop

bei 400 MHz Taktfrequenz: 45 ns / 2.5 ns / Taktzyklus = 18 Taktzyklen

Verteilung der Dauer



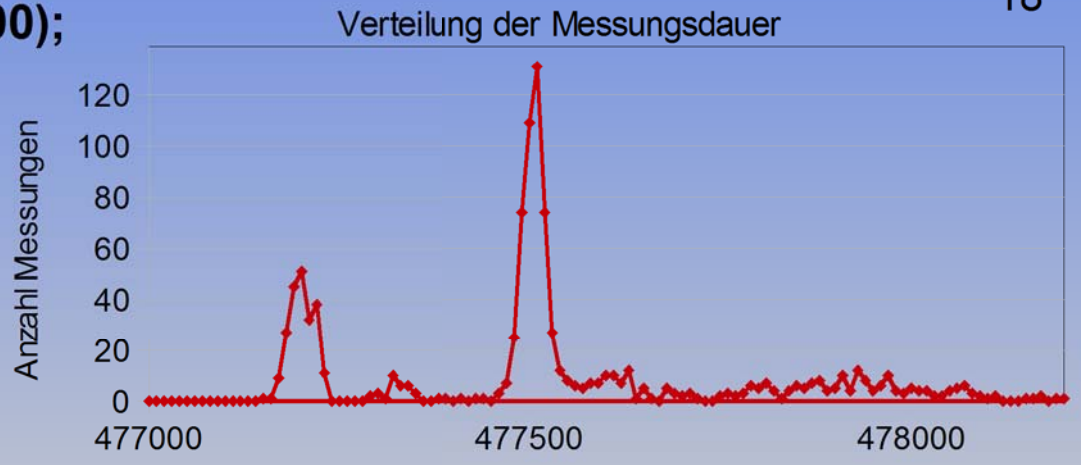


Einfacher Loop

```
int main(){
static int counter = 0;
struct timespec rt1, rt2;
long long time1;
clock_gettime(CLOCK_REALTIME, &rt1);
while(counter<10000000){
   counter++;
}
clock_gettime(CLOCK_REALTIME, &rt2);
time1 = (long long int)(rt2.tv_sec - rt1.tv_sec)*1000000000
+ (rt2.tv_nsec - rt1.tv_nsec);
printf("%.3f\n", (double)(time1)/1000);
exit(0);
}
```

Resultat: 477.5 ±0.5 ms

```
        b       .L2
.L3:
        ldr     r3, .L6+8       2       1-3
        ldr     r3, [r3, #0]    2       1-3
        add     r2, r3, #1      1       1
        ldr     r3, .L6+8       2       1-3
        str     r2, [r3, #0]    1       1-3
.L2:
        ldr     r3, .L6+8       2       1-3
        ldr     r2, [r3, #0]    2       1-3
        ldr     r3, .L6+12      2       1-3
        cmp     r2, r3          1       1
        ble     .L3             3       3
                                18      12- 26
```



Besser!

```
int main(){
static int counter = 0;
struct timespec rt1, rt2;
long long time1;
clock_gettime(CLOCK_REALTIME, &rt1);
while(counter<10000000){
   counter++;
}
clock_gettime(CLOCK_REALTIME, &rt2);
time1 = (long long int)(rt2.tv_sec - rt1.tv_sec)*1000000000
+ (rt2.tv_nsec - rt1.tv_nsec);
printf("%.3f\n", (double)(time1)/1000);
exit(0);
}
```

```
        b       .L2
.L3:
        ldr     r3, [fp, #-32]
        add     r3, r3, #1
        str     r3, [fp, #-32]
.L2:
        ldr     r2, [fp, #-32]
        ldr     r3, .L6+8
        cmp     r2, r3
        ble     .L3
```

~~Resultat: 477.5 ±0.5 ms~~ **Resultat: 276.4±0.2 ms**

...und noch besser!!!

```
        b       .L2
.L3:
        ldr     r3, [fp, #-32]
        add     r3, r3, #1
        str     r3, [fp, #-32]
.L2:
        ldr     r2, [fp, #-32]
        ldr     r3, .L6+8
        cmp     r2, r3
        ble     .L3
```

276.4±0.2 ms

```
        ldr     r3, [fp, #-32]
        ldr     r2, .L6+8
        b       .L2
.L3:
        ldr     r3, [fp, #-32]
        add     r3, r3, #1
        str     r3, [fp, #-32]
.L2:
        cmp     r3, r2
        ble     .L3
```

202.0±0.6 ms

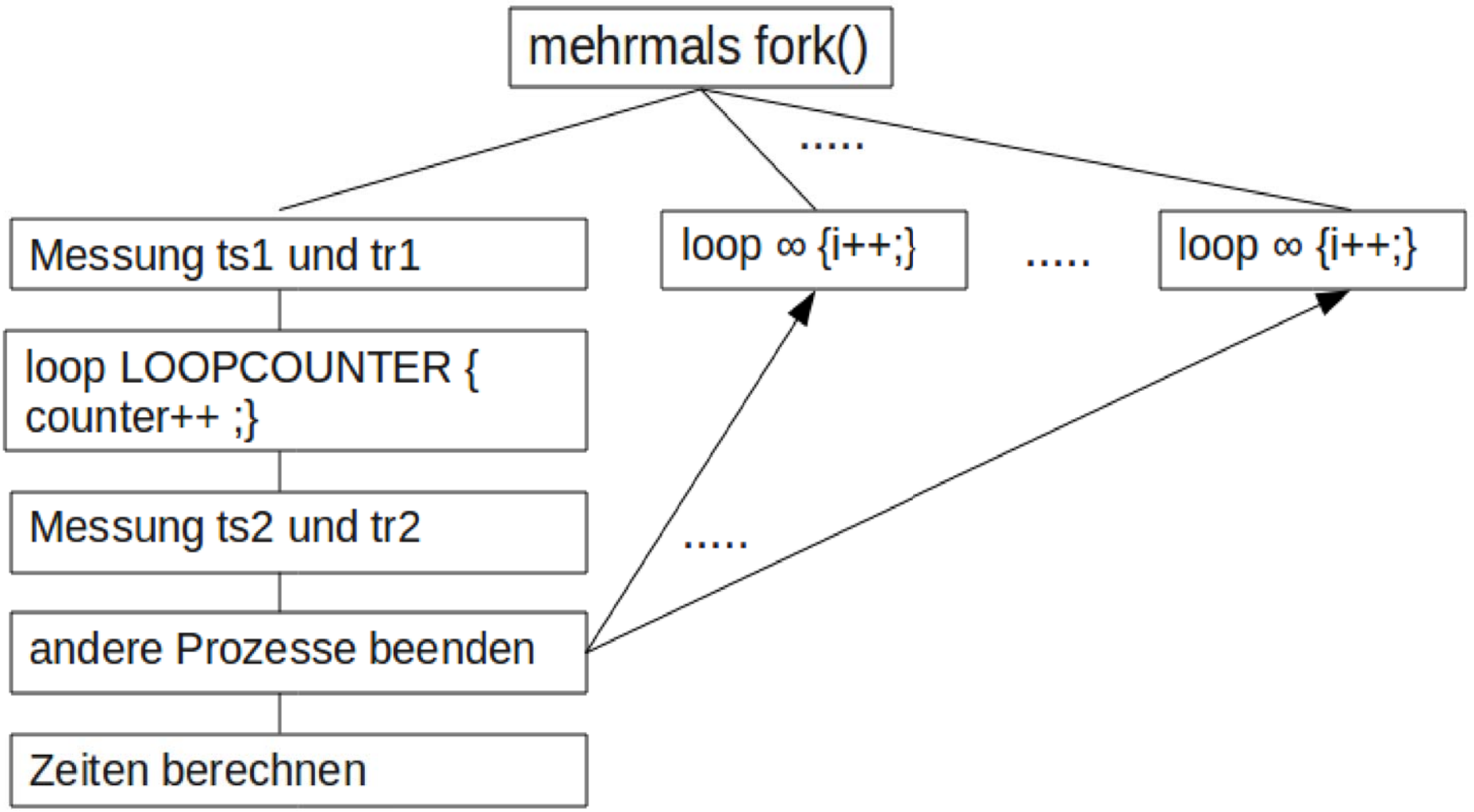
```
int main(){
register counter = 0;
struct timespec rt1, rt2;
long long time1;
clock_gettime(CLOCK_MONOTONIC, &rt1);
while(counter<10000000){
   counter++;
}
clock_gettime(CLOCK_MONOTONIC, &rt2);
.....
```

```
        ldr     r2, .L6+8
        b       .L2
.L3:
        add     r3, r3, #1
.L2:
        cmp     r3, r2
        ble     .L3
```

126.6±0.5 ms

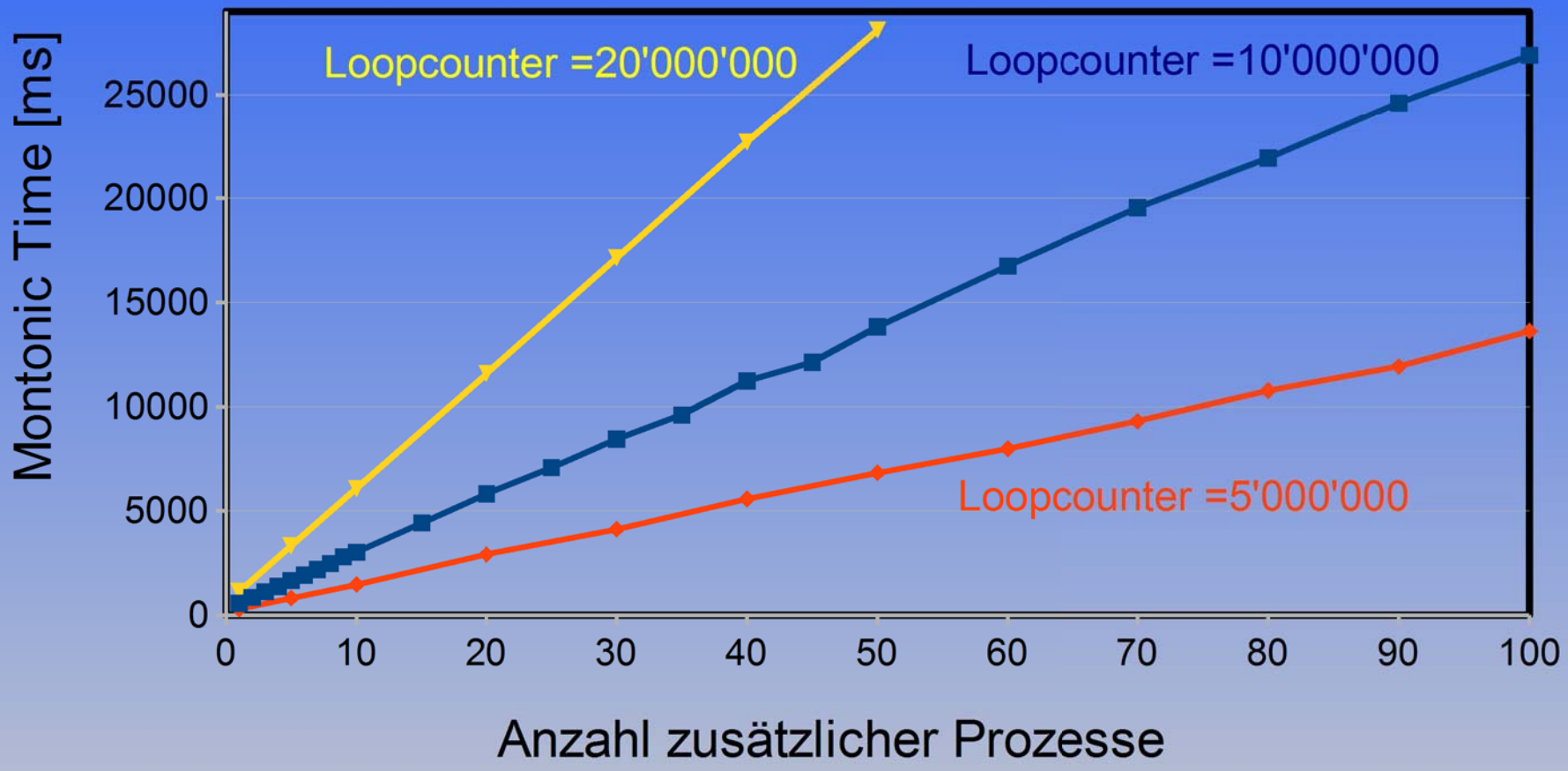
kompiliert mit O1, 02 oder O3: 6.4 ms (Loop wird nicht mehr ausgeführt...)

Ein anderer Versuch

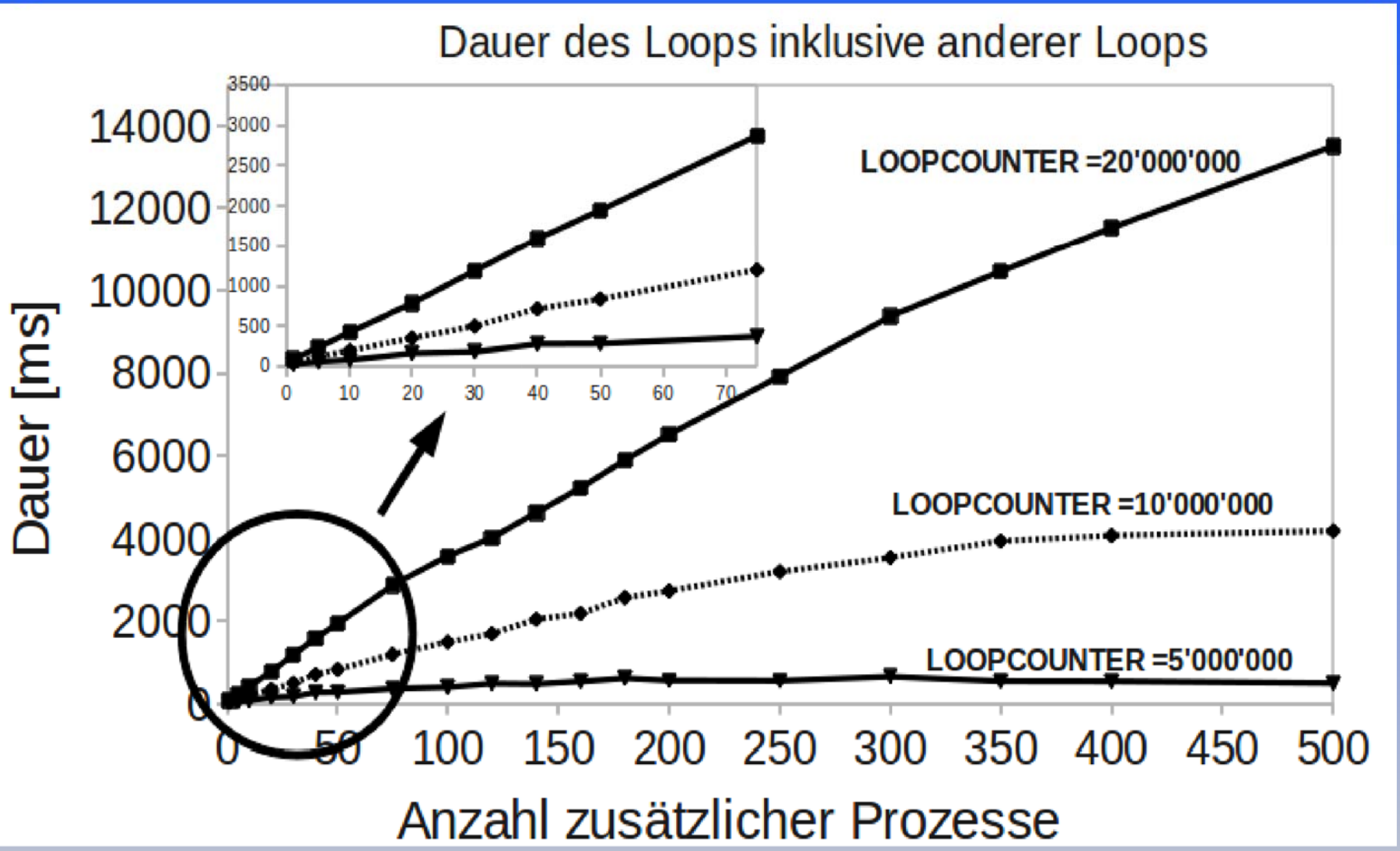


Ergebnisse APF27

Loopzeit bei mehreren Prozessen



Ergebnisse Laptop Intel-Dualcore



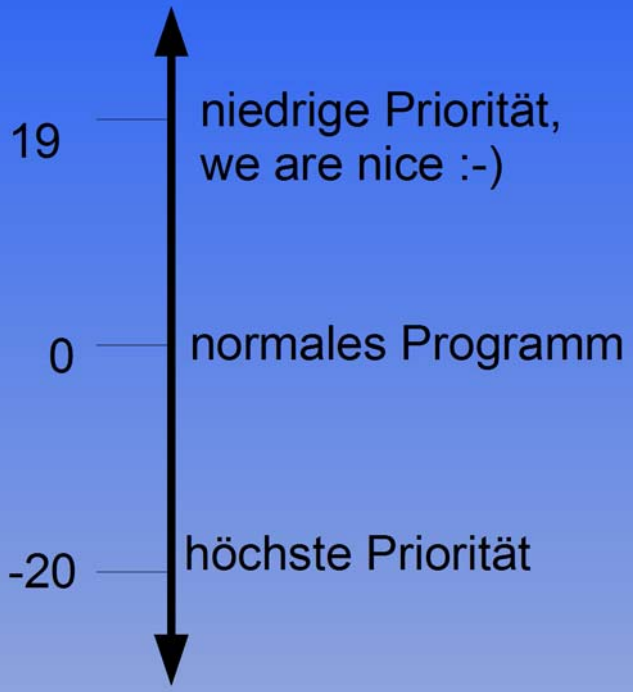
Bevorzugung bestimmter Prozesse

Sind wir nett?

Nice-Werte von -20 bis 19!

<u>Kommandozeile</u>: nice programm
renice inc pid

<u>Systemfunktion</u>: int nice(int inc);

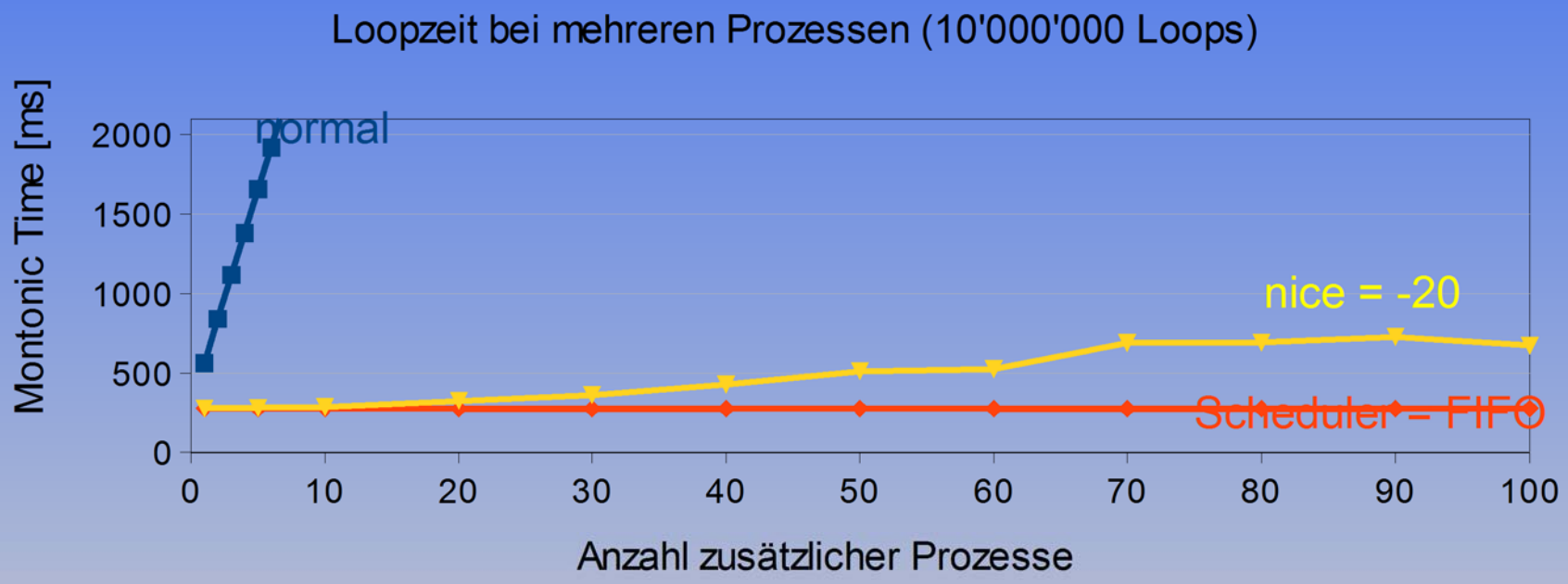


mit dem Programm <u>top</u>

Ändern des Schedulers

- Normal: SCHED_OTHER (preemptives Scheduling)
- Round Robin: SCHED_RR (Gleiche Prioritäten werden preemptiv abgearbeitet)
- strikt: SCHED_FIFO (First come, first serve)
- ganz nett: SCHED_BATCH (Nur wenn wirklich nichts anderes los ist...)

```
int sched_setscheduler(pid_t pid, int policy, const struct sched_param *sp);
```



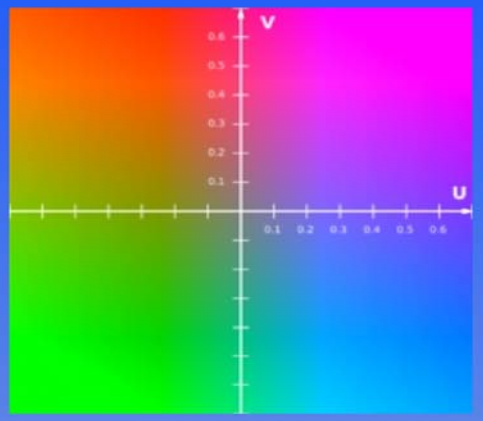
Ein reales Beispiel: Webcam mit APF-27

- Treiber für Kamera: uvcvideo
- Treiber für LCD
- Programm zum lesen der Kamera und Anzeigen auf LCD: capture

```
Mem: 9416K used, 50660K free, 0K shrd, 0K buff, 1932K cached
CPU:  99% usr   0% sys   0% nice   0% idle   0% io   0% irq
0% softirq
Load average: 1.00 0.85 0.45
  PID   PPID USER     STAT   VSZ %MEM  %CPU COMMAND
  468    421 root     R     3080   5%  99% capture
  472    421 root     R     1208   2%   1% top
  421      1 root     S     1244   2%   0% -sh
```

Analyse und Optimierungen

```
process_image(buffers[buf.index].start, buf.length);

yuv422_to_rgb565(void *yuv, int size, void *rgb);
```




```
R = YUV2R(Y, U, V);
G = YUV2G(Y, U, V);
B = YUV2B(Y, U, V);
```

```
R = Y + 1.4026 * (V-128);
G = Y - 0.3444 * (U-128) - 0.7144 * (V-128);
B = Y + 1.7730 * (U-128);
```

```
R = (Y<<10) + 1436*(V-128);
R = R>>10;
```

```
B = (Y<<10) +  1815*(U-128);
B = B>>10;
```

```
G = (Y<<10) -  352*(U-128) - 732*(V-128);
G = G>>10;
```

und noch ein usleep(30000); zwischen den Aufrufen zum Lesen eines Bildes

Und wenn andere Aufgaben zu erledigen sind?

```
Mem: 9348K used, 50728K free, 0K shrd, 0K buff, 1916K cached
CPU:  41% usr   0% sys   0% nice  57% idle   0% io   0% irq   0% softirq
Load average: 0.16 0.04 0.01
  PID  PPID USER     STAT   VSZ %MEM  %CPU COMMAND
  608   593 root     S     2968   5%  34% /home/capture
  612   593 root     R     1208   2%   0% top
  593     1 root     S     1236   2%   0% -sh
    1     0 root     S     1220   2%   0% init
```

Starten von vielen Prozessen 'heizung', die nichts anderes machen, als in einem Endlosloop drehen....

```
Mem: 10104K used, 49972K free, 0K shrd, 0K buff, 1944K cached
CPU:  99% usr   0% sys   0% nice   0% idle   0% io   0% irq   0% softirq
Load average: 5.99 1.96 0.76
  PID  PPID USER     STAT   VSZ %MEM %CPU COMMAND
  608   593 root     R     2968   5%  10% /home/capture
  615   593 root     R      516   1%  10% ./heizung
  616   593 root     R      516   1%  10% ./heizung
  620   593 root     R      516   1%  10% ./heizung
  621   593 root     R      516   1%  10% ./heizung
  622   593 root     R      516   1%  10% ./heizung
  624   593 root     R      516   1%  10% ./heizung
  617   593 root     R      516   1%  10% ./heizung
  618   593 root     R      516   1%  10% ./heizung
  619   593 root     R      516   1%  10% ./heizung
```

Control Groups

Control Groups provide a mechanism for aggregating/partitioning sets of tasks, and all their future children, into hierarchical groups with specialized behaviour.

Mounten der C-groups:

```
mkdir /cgroups
mount -t cgroup -o cpu groups /cgroups
```

```
# pwd
/cgroups
# ls
cpu.rt_period_us    cpu.shares          release_agent
cpu.rt_runtime_us   notify_on_release   tasks
#
```

```
mkdir /cgroups/camera
mkdir /cgroups/low

echo PID(capture) >
/cgroups/camera/tasks
echo PID(heizung) > /cgroups/low/tasks
```

root-Gruppe hat cpu.shares von 1024

```
echo 1024 > /groups/camera/cpu.shares
echo 1024 > /groups/low/cpu.shares
```

Control Groups

```
echo 608 > /cgroups/camera/tasks
echo 615 > /cgroups/low/tasks
echo 616 > /cgroups/low/tasks
echo 617 > /cgroups/low/tasks
echo 618 > /cgroups/low/tasks
```

```
echo 619 > /cgroups/low/tasks
echo 620 > /cgroups/low/tasks
echo 621 > /cgroups/low/tasks
echo 622 > /cgroups/low/tasks
echo 624 > /cgroups/low/tasks
```

```
Mem: 10136K used, 49940K free, 0K shrd, 0K buff, 1952K cached
CPU:  99% usr   0% sys   0% nice   0% idle   0% io   0% irq   0% softirq
Load average: 9.35 8.82 5.44
  PID  PPID USER     STAT       VSZ %MEM %CPU COMMAND
  608   593 root     R         2968   5%  33% /home/capture
  619   593 root     R          516   1%   8% ./heizung
  617   593 root     R          516   1%   8% ./heizung
  620   593 root     R          516   1%   7% ./heizung
  624   593 root     R          516   1%   7% ./heizung
  622   593 root     R          516   1%   7% ./heizung
  621   593 root     R          516   1%   7% ./heizung
  615   593 root     R          516   1%   7% ./heizung
  618   593 root     R          516   1%   7% ./heizung
  616   593 root     R          516   1%   7% ./heizung
  647   593 root     R         1208   2%   1% top
```

cgroups-Subsysteme

blkio this subsystem sets limits on input/output access to and from block devices such as physical drives (disk, solid state, USB, etc.).

cpu this subsystem uses the scheduler to provide cgroup tasks access to the CPU.

cpuacct this subsystem generates automatic reports on CPU resources used by tasks in a cgroup.

cpuset this subsystem assigns individual CPUs (on a multicore system) and memory nodes to tasks in a cgroup.

devices this subsystem allows or denies access to devices by tasks in a cgroup.

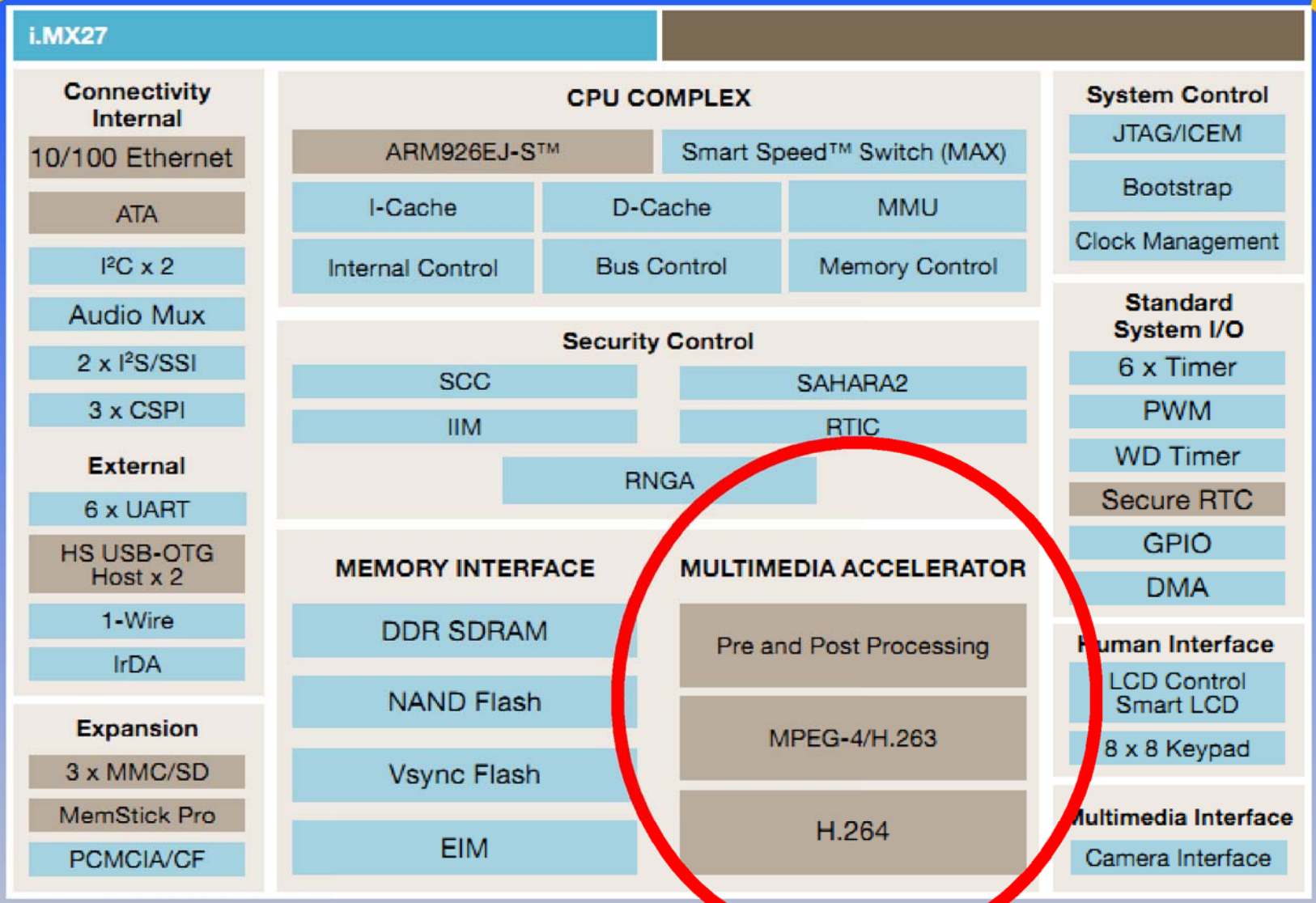
freezer this subsystem suspends or resumes tasks in a cgroup.

memory this subsystem sets limits on memory use by tasks in a cgroup, and generates automatic reports on memory resources used by those tasks.

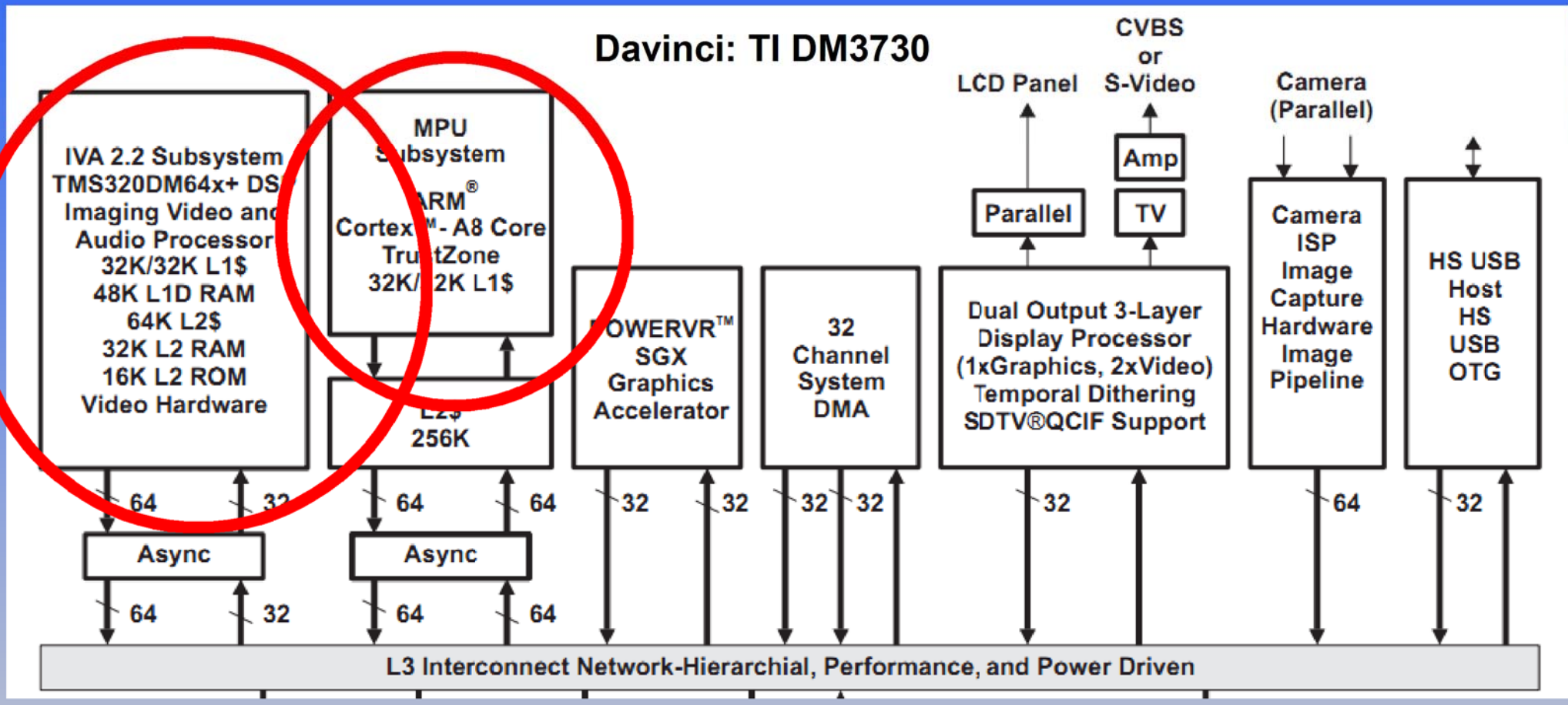
net_cls this subsystem tags network packets with a class identifier (classid) that allows the Linux traffic controller (tc) to identify packets originating from a particular cgroup task.

ns the namespace subsystem.

Gibt es noch andere Möglichkeiten?



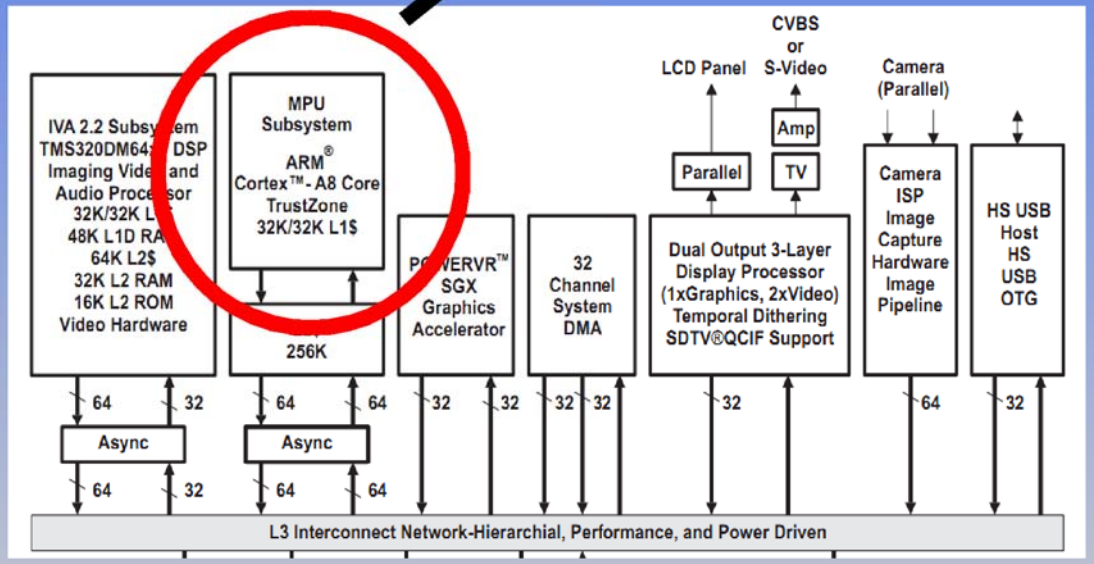
Unser Projekt SOSoC: System Optimization using a System on Chip

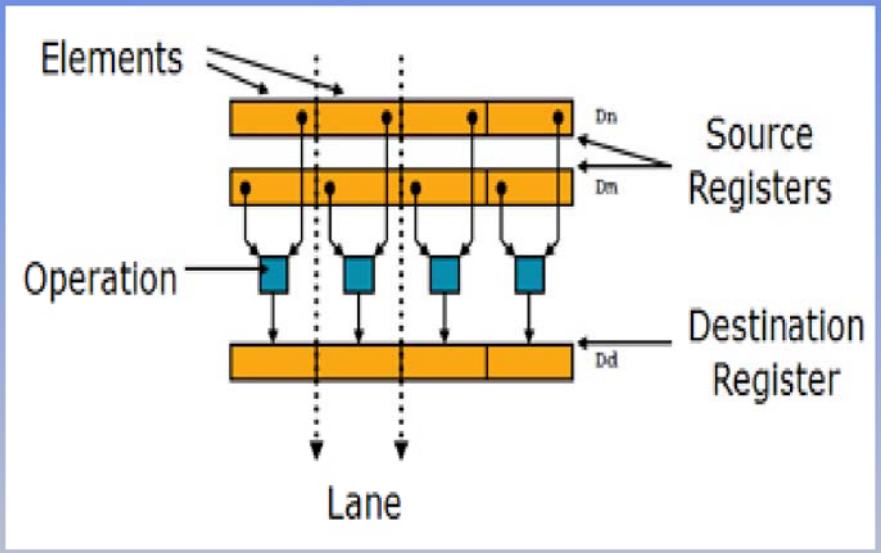


Co-Prozessor NEON: SIMD

MPU Subsystem

ARM®
Cortex™- A8 Core
TrustZone
32K/32K L1$





NEON: ein Beispiel

```
void NeonTest(short int * __restrict a, short int * __restrict b, short int * __restrict z)
{
    int i;
    for (i = 0; i < 200; i++) {
        z[i] = a[i] * b[i];
    }
}
```

```
000000  e3a03019    MOV       r3,#0x19
|L1.4|
000004  f4200a4d    VLD1.16   {d0,d1},[r0]!
000008  e2533001    SUBS      r3,r3,#1
00000c  f4212a4d    VLD1.16   {d2,d3},[r1]!
000010  f2100952    VMUL.I16  q0,q0,q1
000014  f4020a4d    VST1.16   {d0,d1},[r2]!
000018  1afffff9    BNE       |L1.4|
00001c  e12fff1e    BX        lr
```

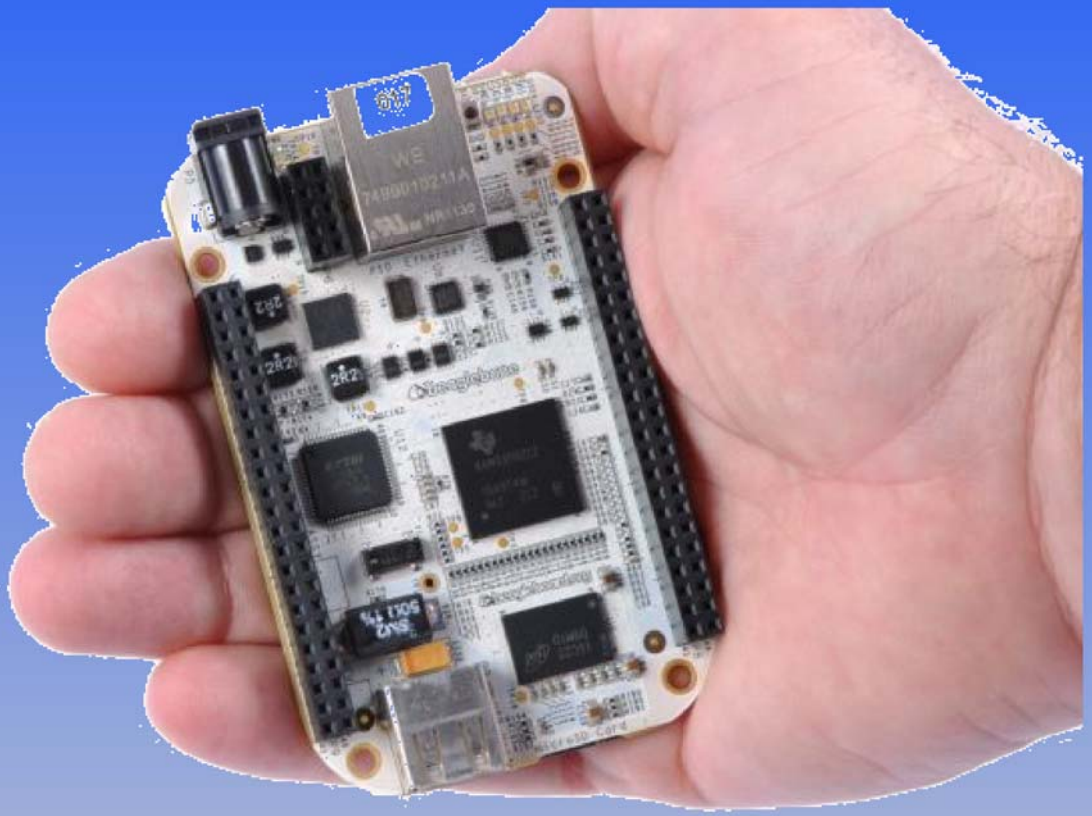
25 loops

200 loops

```
000000  e92d4030  PUSH      {r4,r5,lr}
000004  e3a03000  MOV       r3,#0
|L1.8|
000008  e0804083  ADD       r4,r0,r3,LSL #1
00000c  e0815083  ADD       r5,r1,r3,LSL #1
000010  e1d440b0  LDRH      r4,[r4,#0]
000014  e1d550b0  LDRH      r5,[r5,#0]
000018  e1640584  SMULBB    r4,r4,r5
00001c  e0825083  ADD       r5,r2,r3,LSL #1
000020  e2833001  ADD       r3,r3,#1
000024  e35300c8  CMP       r3,#0xc8
000028  e1c540b0  STRH      r4,[r5,#0]
00002c  bafffff5  BLT       |L1.8|
000030  e8bd8030  POP       {r4,r5,pc}
```

Das Beagleboard

~ 120 Euro, mit DSP und NEON



~ 70 Euro, kein DSP aber NEON

beagleboard.org

Dank an die beteiligten Personen

Vielen Dank an:

Davide Doninelli,
Guy Morand,
Silvain Décastel,
Xavier Menoud,
Andéol Demierre,
Igor Zanetti

Philipp Zaugg
Samuel Tâche
Cédric Attallah

Jean-Roland Schuler
Nicolas Schroeter

Vielen Dank auch an Euch für die Aufmerksamkeit !

wolfram.luithardt@hefr.ch