Echtzeitanforderung und Linux

Pengutronix.

Definition Harte Echtzeit I

- Was zeichnet ein Echtzeitsystem aus?
  - Zeitverhalten ist Teil der System-Spezifikation!
  - Bei Embedded-Systemen heißt das meist:
    Zeit für Reaktion auf Interrupt (Timer oder Extern)
  - Wichtig: Echtzeit heißt nicht "schnell", sondern "rechtzeitig"!

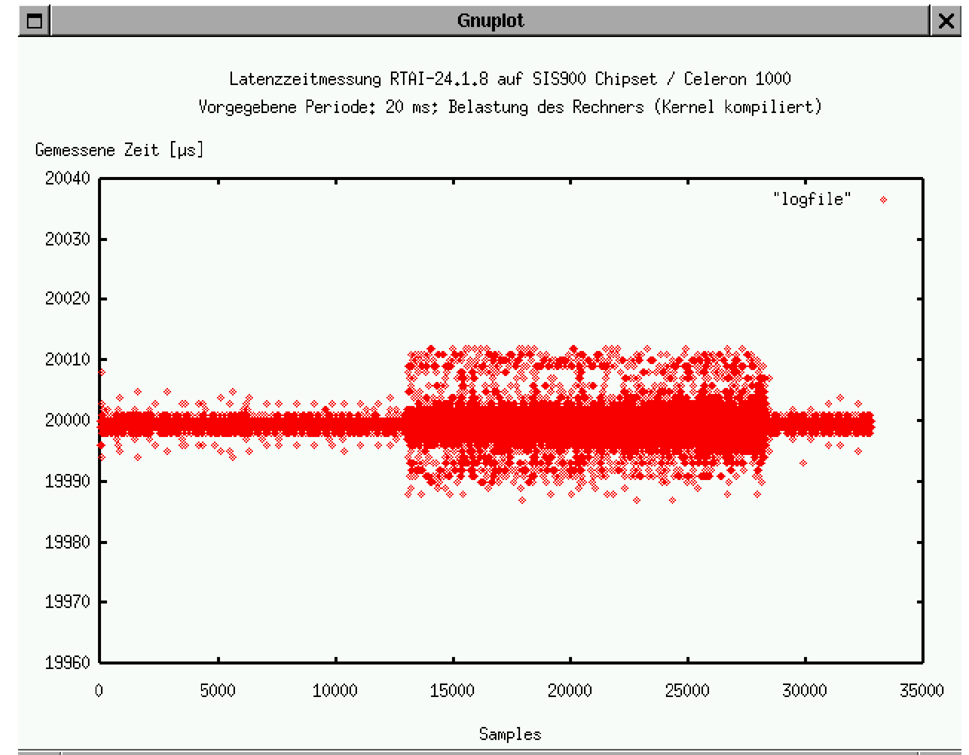
Definition Harte Echtzeit II

- Deterministische Antwortzeit auf Ereignisse

- Häufige Anwendung:

  Zyklische Tasks mit vorgegebener Zyklus-zeit

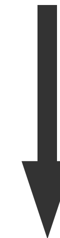
- ... aber warum sind "General Purpose" Betriebssysteme nicht echtzeitfähig?



Pengutronix.

Ursachen für Latenz bei Microcontroller

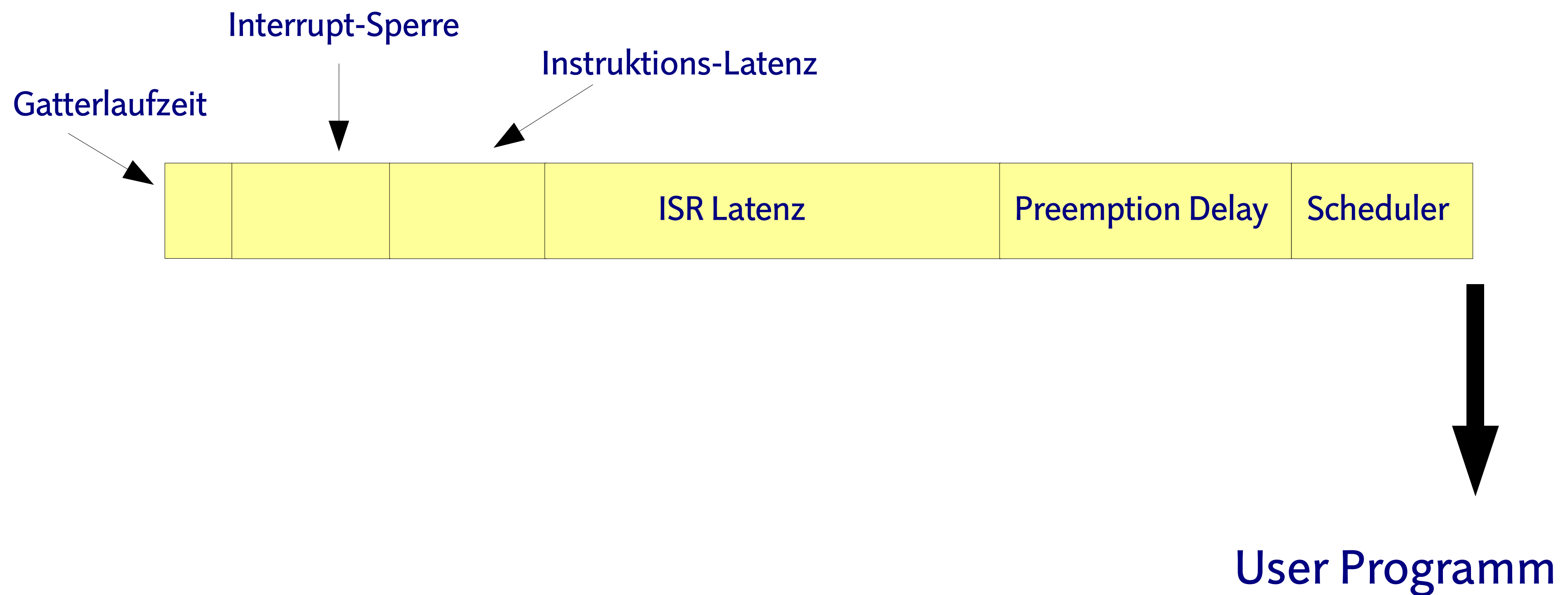
- Single Threaded Context:



User Programm

Ursachen für Latenz bei Prozessor + OS

- Multi Threaded Context:



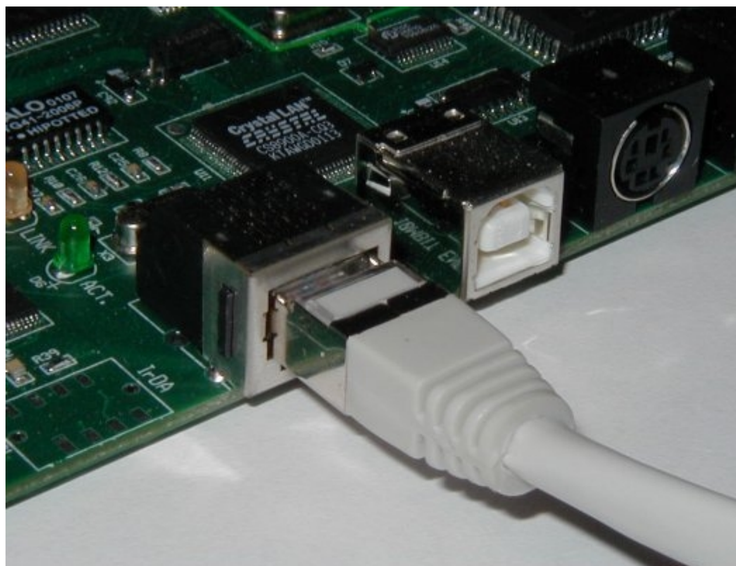
Pengutronix

Warum ist die Latenz nicht deterministisch?

- Standard-Linux 2.4 und 2.6 (mit Ausnahmen) sind nicht preemptiv!
- Interrupt kommt "irgendwann"
- Wenn System gerade im Kernel läuft (System Call), wird der gerade laufende Context abgearbeitet, bis er abgeschlossen ist.
- Das kann, je nach Systemzustand, lange dauern.

Warum dann nicht Controller?

- Moderne Embedded Systeme haben oft nicht nur Echtzeitanforderungen!
- Kommunikation, Feldbusse, Netzwerk-Protokolle, Visualisierung, Web-Schnittstelle usw.



Pengutronix.

Echtzeit findet auf vielen Ebenen statt

- Rechenleistungs-Zuteilung (Scheduling):
  - Weiche Echtzeit
  - Harte Echtzeit
- Resourcen-Verwaltung
  - Speicher
  - Locks
- Kommunikation zwischen Systemen

Pengutronix.

Vergabe der Rechenleistung an Prozesse

UNIX-artige Betriebssysteme verwenden einen allgemeinen Ansatz dafür:

- Fair Queueing
  - Faire Verteilung der Rechenleistung auf alle Prozesse.
  - Kein Prozess „verhungert“
  - „Sanfte“ Vergabe von Prioritäten
  - Prozesse mit hoher Priorität erhalten zwar insgesamt mehr Rechenleistung zugeteilt, aber sie nehmen diese anderen aktiven Prozessen nicht weg. D.h. auch ein hoch priorisierter Prozess muss ggf. warten, bis alle anderen abgearbeitet sind

Vergabe der Rechenleistung an Prozesse

Reine Echtzeitbetriebssysteme verwenden einen anderen Ansatz:

- Priority Queueing
  - Vergabe der Rechenleistung ausschließlich über die Priorität
  - Hochpriore Prozesse laufen immer auf Kosten niederpriorer Prozesse
  - Niederpriore Prozesse erhalten im schlechtesten Fall niemals Rechenleistung
  - Hochpriore Prozesse laufen immer, wenn sie im Zustand „Bereit“ sind

Vergabe der Rechenleistung an Prozesse

Wie wird Linux 2.6.x diesen unterschiedlichen Anforderungen gerecht?

- Es verwendet zwei unterschiedliche Scheduler-Verhalten:
  - Standard-Verhalten für „Fair-Queueing“
  - Echtzeit-Verhalten für „Priority-Queueing“
- Alle Prozesse im System können einem der beiden Scheduler-Verhalten zugeordnet werden
- Allen Prozessen im System kann eine Priorität zugeordnet werden

Beispiel: Kamera steuert Bandlauf

Aufgabe:

- Zu bearbeitendes Gut kommt als Bandware in die Verarbeitungs-Maschine
- Das Gut muss auf den Laufrollen auf einer bestimmten Position gehalten werden, um die nachfolgende Bearbeitung zu ermöglichen

Umsetzung:

- Eine Zeilenkamera nimmt die aktuelle Position des Guts auf der Rolle auf
- Ein Regelungsprozess steuert die Lage der Laufrolle, um das Gut auf Position zu halten

Voraussetzung:

- Für die Regelungsstrecke muss eine äquidistante Abtastrate sichergestellt werden

Beispiel: Kamera steuert Bandlauf

Ablauf:

- Regelungsprozess nimmt eine „Probe“ der Kamera
- Berechnet die neuen Vorgaben für die Stellmotoren der Laufrollen
- Geht in den Zustand „Warten“ bis zum Beginn der nächsten Abtastung

Problem:

- Einstellung der Periodendauer bzw. Wartezeit bis zum Beginn der nächsten Abtastung:
  - Standard Kernel 2.4/2.6 verwendet nur Timer-Ticks und deren Raster (beispielsweise 10 ms oder 8 ms)
- Sicherstellen, daß zu Beginn der neuen Abtastung der Regelungsprozess Rechenleistung zugeteilt bekommt:
  - Standard Kernel 2.4 kann das nicht garantieren!

Beispiel: Kamera steuert Bandlauf

Lösungsansatz:

- Verwendung eines Standard Kernel 2.6
- Zuordnen des Regelungsprozesses an den Echtzeit-Scheduler
- Verwendung einer höheren Timer-Tick-Rate

In diesem Beispiel reichen die verbesserten Soft Realtime Eigenschaften des 2.6er Kernels gegenüber 2.4 aus.

Harte Echtzeit

- Harte Echtzeit stellt weitere Anforderungen an das System, welche durch reine Prioritätssteuerung und gesonderte Prozess-Scheduler nicht zu erreichen sind
- Die Anforderung, eine laufende Aktivität zugunsten einer höher priorisierten Aktivität zu unterbrechen, kann zu jedem beliebigen Zeitpunkt eintreffen
- Die gerade laufende Aktivität kann ein Userspace-Prozess, ein Betriebssystemaufruf eines Userspace-Prozesses, aber auch ein Kernel-Thread oder eine Interrupt-Routine sein
- Voraussetzung für harte Echtzeit ist, dass alle genannten Aktivitäten unterbrochen und durch eine höher priorisierte Aktivität ersetzt werden können (Preemption)

RTAI

- Standard-Linux 2.4 und 2.6 (mit Ausnahmen) sind nicht preemptiv!
- Interrupt kommt "irgendwann"
- Wenn System gerade im Kernel läuft (System Call), wird der gerade laufende Context abgearbeitet, bis er abgeschlossen ist.
- Das kann, je nach Systemzustand, lange dauern.

RTAI Systemaufbau

Userspace Programme

Linux 2.4

RTAI-Programme im Kernel

RTAI Infrastruktur für
Threads & Kommunikation

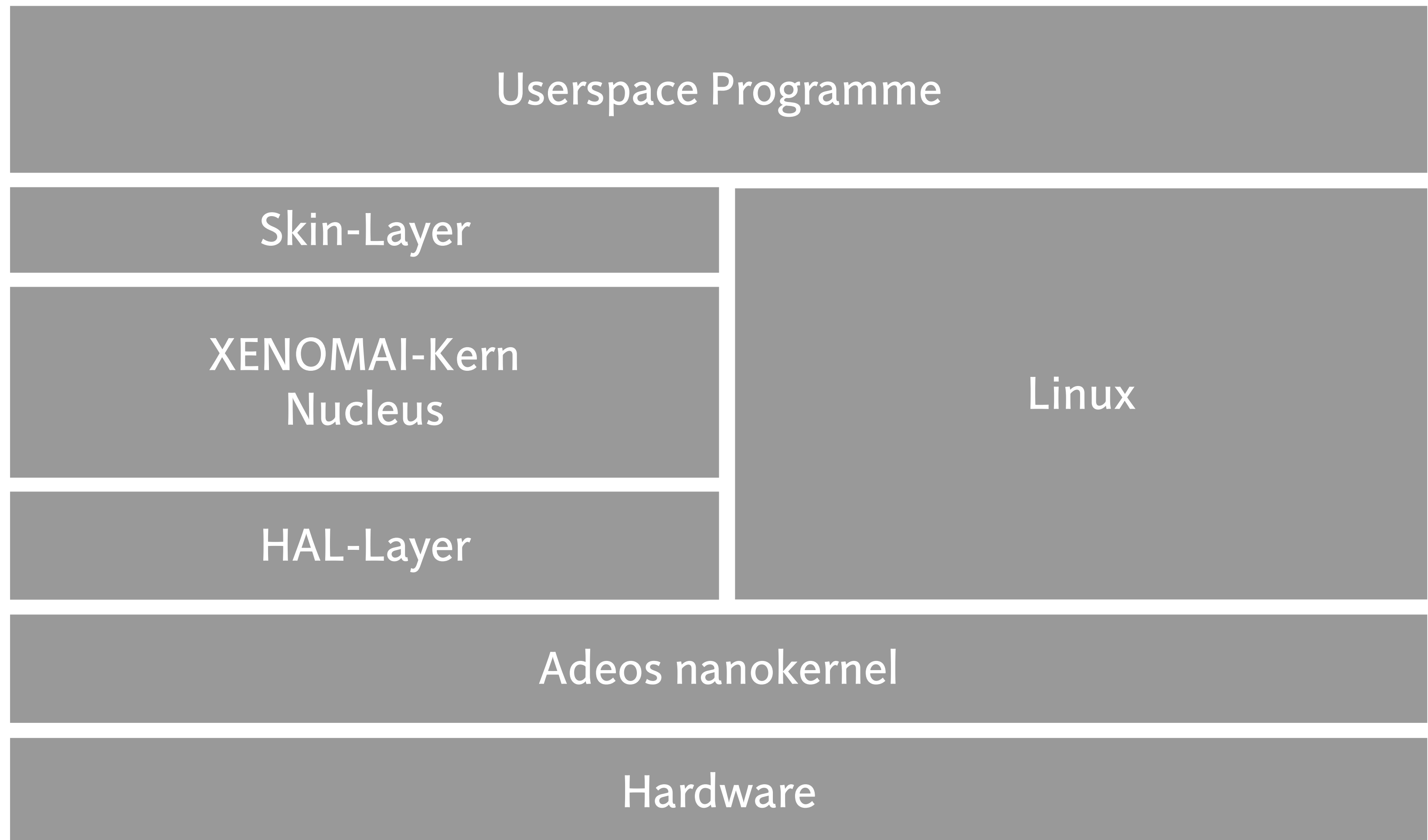
RTAI HAL Schicht (fängt alle Interrupts ab)

Hardware

Xenomai

- Dual Kernel Ansatz
- Echtzeit-Programme binden gegen Xenomai
- Echtzeit-Programme können auch den Linux-Kernel verwenden, verlieren dabei aber Performance

Xenomai Systemaufbau



Pengutronix.

Mainline HRT: Preempt RT im Kernel

Seit ca. 2002 gibt es konstante Aktivitäten, den Linux Kernel selbst hart echtzeitfähig zu machen, und zwar ohne Dual Kernel Ansatz!

Änderungen am Kernel (Mainline ab 2.6.17 ff.):

- Generic Interrupt Layer
  Behandlung von Interrupts über alle Architekturen hinweg
- Interrupt-Service-Routinen unterliegen dem Scheduler
- Abstrakte, architekturunabhängige Timer, Separierte Timeouts
- Neues Framework für hochauflösende Timer mit geringem Overhead
- Priority Inherritance Mutexe (PI-Mutexe)
- Ersetzen von Spinlocks durch PI-Mutexe
- Vollständig Preemptiv

Entwickeln mit RT-Preempt

- Echtzeit wird als Spezialfall angesehen:
  - Entwicklung startet wie gewohnt im Userspace
  - "Einschalten" der Echtzeiteigenschaften, wenn nötig
- Es soll möglichst keinen Unterschied machen, ob man für Echtzeit oder Nicht-Echtzeit programmiert
  - Vereinfachtes Debugging mit vorhandenen Tools
  - Einfache Portierung bestehender Software

Beispiel: Servo-Steuerung mittels PWM

Aufgabe:

- Über einen in Software kontrollieren GPIO wird ein PWM-Signal erzeugt, welches darüber die Position eines Servo-Motors vorgibt
- Der Motor muss die Position stabil halten, darf nicht „jittern“

Umsetzung:

- Eine Prozess kontrolliert den GPIO mittels einer Zeitschleife
- Die meisten Servo-Motoren verlangen eine bestimmte Periodendauer, daran muss sich das Programm anpassen lassen

Voraussetzung:

- Beliebige Wartezeiten müssen realisierbar sein

Beispiel: Servo-Steuerung mittels PWM

- Prozess bekommt die Vorgabe für das Puls-Pausenverhältnis
- Die Periodendauer wird in 100 Teile aufgeteilt, der Prozess muss 100 mal pro Periode geweckt werden
- Ist das Puls-Pausenverhältnis erreicht, wird der Pegel am GPIO geändert

Problem:

- Der Kernel muss die mitunter kurzen Wartezeiten umsetzen können
- Der Kernel muss innerhalb einer Toleranz sicherstellen, dass der Prozess nach dieser Wartezeit auch tatsächlich Rechenleistung erhält

Beispiel: Servo-Steuerung mittels PWM

Lösungsansatz:

- Verwendung eines RT-Preempt-gepatchten Kernel 2.6
- Verwendung der hochauflösenden Timer
- Zuordnen des Steuerungsprozesses an den Echtzeit-Scheduler

Resourcenverwaltung unter Echtzeit

- Echtzeitapplikationen sind praktisch nie homogene Applikationen
  - In der Regel hat nur ein kleiner Teil Echtzeitanforderungen
  - Der Rest der Applikation ist zeitunkritisch
- Durch diesen Aufbau ergeben sich Berührungspunkte beider Teile
  - Gemeinsame Nutzung von Resourcen wie Speicher zum Zwecke der Kommunikation bzw. Datenaustausch
- Gemeinsam genutzte Resourcen müssen vor konkurrierenden Zugriffen geschützt werden!
- Dieser Schutz darf aber die Prioritätssteuerung nicht sabotieren

Beispiel: GPIO-Verwaltung

Aufgabe:

- In einem Register ist die Kontrolle über mehrere GPIO realisiert
- Die meisten GPIO darin unterliegen langsamen Kontrollen
- Einige GPIO werden aber von einem Echtzeitprozess kontrolliert

Umsetzung:

- Eine Mutex schützt den Zugriff auf das gemeinsame Register (beispielsweise in einer gemeinsam benutzten Bibliotheksfunktion)
- Alle Prozesse verwenden die Bibliotheksfunktion, um ihnen zugeordnete GPIO zu manipulieren
- Die Mutex verhindert das Chaos beim gemeinsamen Zugriff

Beispiel: GPIO-Verwaltung

Problem:

- Besitzt gerade ein niederpriorer Prozess die Mutex, blockiert er einen hochprioren Prozess, der ebenfalls zugreifen möchte
- Wird nun der niederpriore Prozess seinerseits unterbrochen, besteht diese Blockade für eine beliebige Zeit

Lösungsansatz:

- Verwendung der PI-Mutexe im Kernel 2.6

Echtzeit im Userspace

- Wie die vorangegangenen Beispiele andeuten, ist Echtzeit nicht nur innerhalb des Kernels möglich, sondern auch im Userspace
- Die umgesetzten Maßnahmen im Kernel sind für weite Teile des Kernels selbst transparent und vollständig transparent für den Userspace
- POSIX enthält bereits seit langer Zeit die dafür notwendigen Spezifikationen und APIs
- Mit den aktuellen Maßnahmen im Linux 2.6 Kernel wurden diese nur umgesetzt

Grenzen von Software-HRT

- Optimale Voraussetzung: Linux mit Echtzeit-Erweiterung, unterstützt durch Hardware:
  - FPGAs
  - Microcontroller
  - DSPs
- Für Datenprozessing: Framework-Basierte Abstraktionssysteme, z.B. freeSP

Zusammenfassung

- Der Standard-Linux-2.6-Kernel erfüllt bereits einen Teil der Standard-Echtzeitanforderungen (Soft-Realtime)
- Darüber hinausgehende Forderungen können vom RT-Preempt-Patch erfüllt werden (Hard-Realtime)
- Echtzeit ist nicht auf den Kernel beschränkt
- Echtzeit ist Architektur-Unabhängig
- Über die normale POSIX-API kann jeder Prozess diese Echtzeiteigenschaften verwenden
- Entwicklung von Echtzeitapplikationen erfordert keine besonderen Werkzeuge mehr