Forth in Echtzeit durch

Zeit-gesteuerte Architektur

Ulrich Hoffmann <uho@forth-ev.de>

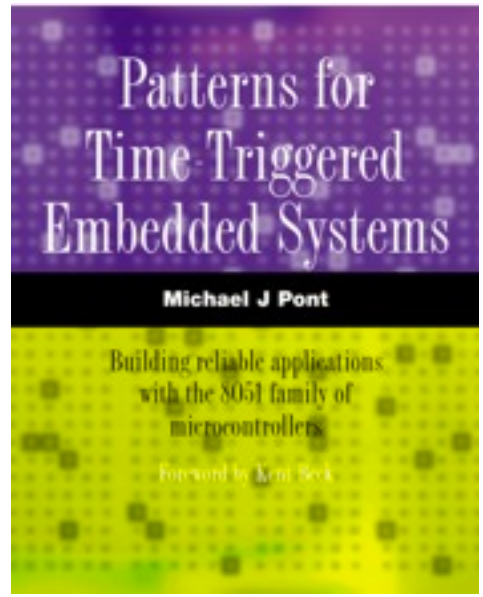
Überblick

- Zeit-gesteuerte Architektur
- Implementierung auf Mecrisp-Stellaris
- Demo
- Bewertung/Diskussion

Multitasking

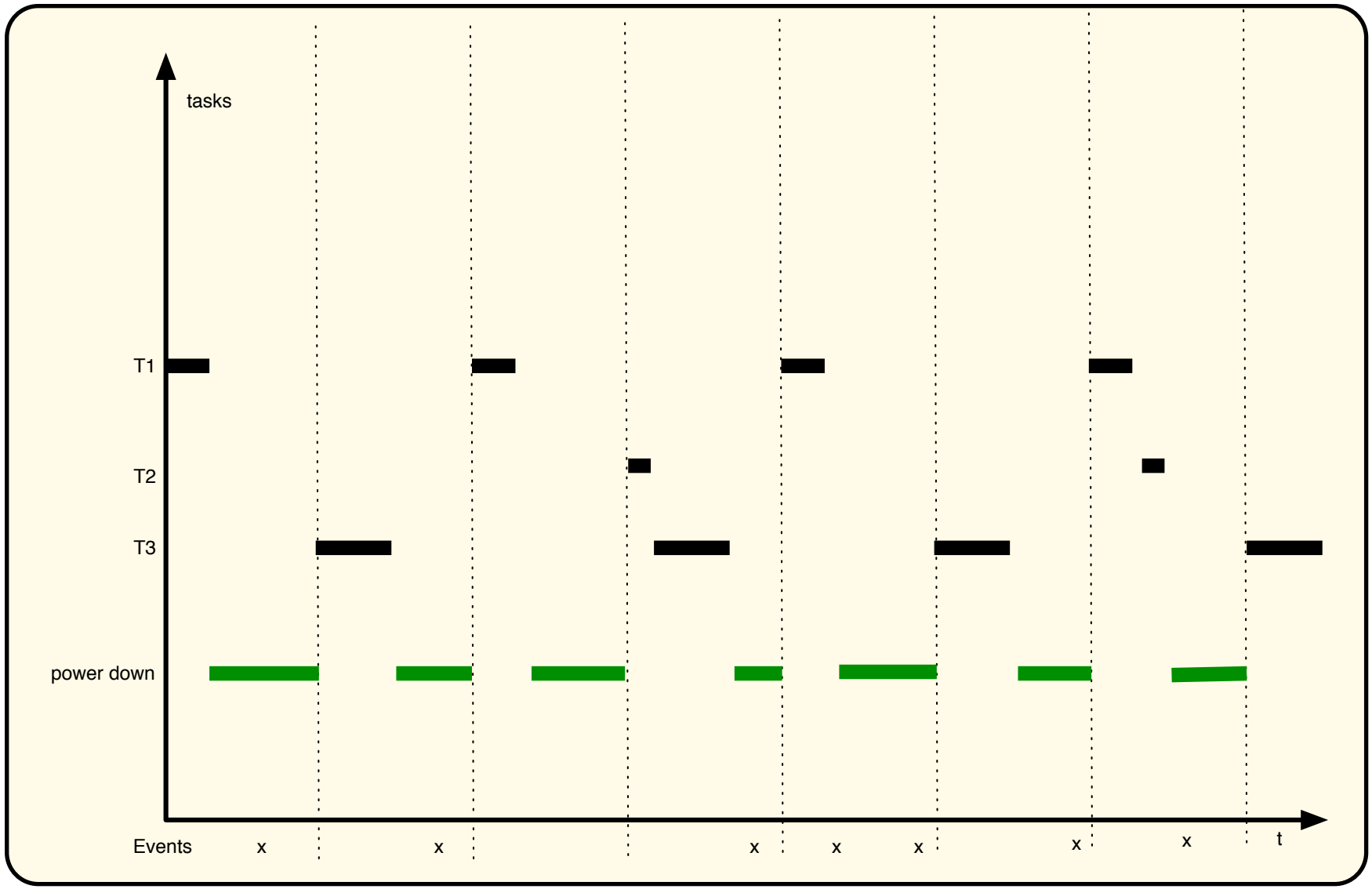
- Reaktion auf asynchrone Ereignisse
  - *Latenz*: Wie schnell erfolgt die Reaktion?
  - *Echtzeit*: Behandlung der Ereignisse erfolgt rechtzeitig garantiert innerhalb eines festen Zeit-Intervalls
- (asynchrone) Interrupts
  - Unterbrechen das laufende Programm (und Interrupts?)
- Preemptive Scheduling
  - Unterbrechen am Ende einer Zeitscheibe
  - Kontext?
- kooperatives Scheduling
  - Task gibt Kontrolle ab, wenn dazu bereit
  - Kontext minimal
- Zeit-gesteuerte Architektur, synchrone Interrupts
  - Das beste beider Welten?

Zeit-gesteuerte Architektur



- *Time Triggered Architecture*
- Michael J. Pont
- Patterns for Timed Triggered Embedded Systems (2001), als PDF vom Autor
- Tasks mit fester Zykluszeit
- deterministische Ausführung
- Scheduler ganz in C (ohne Assembler) realisiert

Zeit-gesteuerte Architektur



Wenn das in C geht,

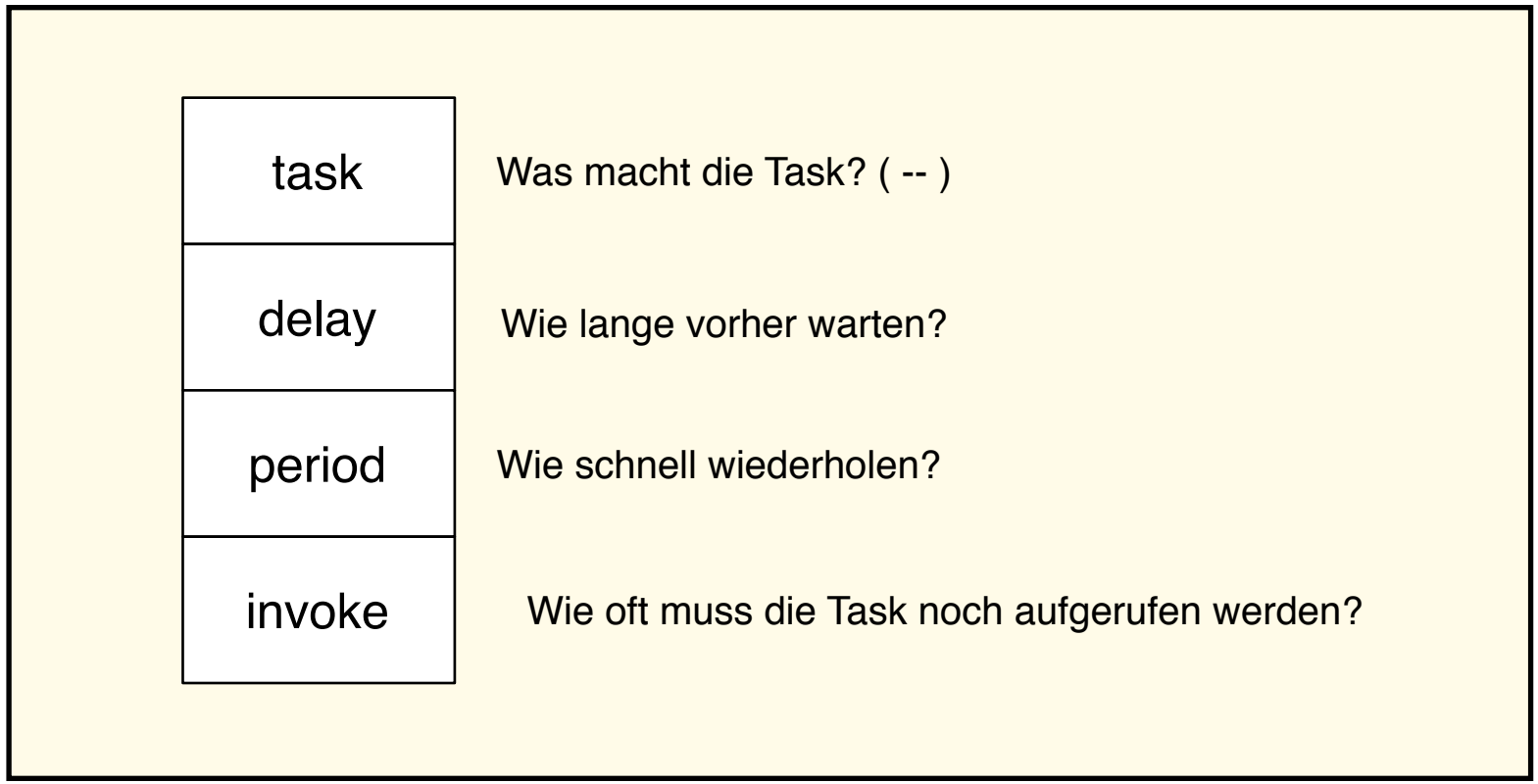
dann wollen wir das auch in Forth!

Interaktivität muss erhalten bleiben

Zeit-gesteuerte Architektur in Forth

- Tasks werden in Forth-Worten realisiert
  - ausgeglichener Stack (--)
  - Kurzläufer
- zeitgesteuerter synchroner Interrupt (*Scheduler*)
  - entscheidet, welche Tasks laufen müssen
- Vordergrund führt Tasks aus (*Dispatcher*)

Task-Control-Blöcke



Festes Maximum an Tasks, in einem Array organisiert

Implementierung

Quelle: ti.com

- Mecrisp-Stellaris auf Tiva Connected Launchpad
  - High-Level Interrupts
  - native Code
- Scheduler ganz in Forth programmiert
- geeignet auch für sehr kleine Systeme (xxx Bytes)

Task-Control-Blöcke

task	Was macht die Task? (--)
delay	Wie lange vorher warten?
period	Wie schnell wiederholen?
invoke	Wie oft muss der Handler noch aufgerufen werden?

```
0
1 cells Field: >xt
1 cells Field: >delay
1 cells Field: >period
1 cells Field: >invoke
Constant /task

4 Constant #tasks
/task #tasks * Buffer: task-table
task-table /task #tasks * + Constant task-table-end
```

Task-Control-Blöcke

```
: find-free-task ( -- 'task ) ... ;

: activate ( delay period xt 'task -- )
   dup >r >xt !
       r@ >period !
       r> >delay ! ;

: add-task ( delay period xt -- )
   find-free-task activate ;

: del-task ( 'task -- )  /task 0 fill ;

: EVENTUALLY ( ticks xt -- ) 0 swap add-task ;

: ALWAYS ( ticks xt -- ) 0 -rot add-task ;
```

Scheduler

```
: schedule ( 'task -- )
    dup >xt @
    IF  dup >delay @ 0=
        IF  1 over >invoke +!
            dup >period @  over >delay !
        ELSE  -1 over >delay +! THEN
    THEN
    drop ;
```

Muss die Task laufen?

in **`invoke`** *zählen!*

Scheduler

```
: tick ( -- )
    task-table
    BEGIN ( 'task )
      dup task-table-end -
    WHILE ( 'task )
      dup schedule
      /task +
    REPEAT ( 'task )
    drop ;
```

Welche Tasks müssen laufen?

Scheduler

```
50 ( us ) Constant #period

: ticker ( -- )
   ['] tick irq-systick !
   16 #period * ( @ 16 Mhz ) systick  eint ;

: milliseconds ( ms -- ticks ) 1000 #period / * ;
: seconds ( s -- ticks ) 1000 * milliseconds ;
```

Scheduler als Interrupt definieren

Dispatcher

```
: dispatch ( -- )
    task-table
    BEGIN ( link )
      dup task-table-end -
    WHILE ( link )
      dup >invoke @
      IF dup >r >xt @ execute
         r> -1  over >invoke +!
         dup >period @ 0= IF  dup del-task THEN
      THEN
      /task +
    REPEAT
    drop ;
```

Lass die Tasks laufen, die noch laufen müssen.

Applikation

```
500 milliseconds  ' toggle-LED ALWAYS

10 seconds  ' led2-toggle  EVENTUALLY
```

wiederkehrende und einmalige Tasks

schön

Und die Interaktivität?

Interaktivität

```
hook-key @ Constant old-key

: ttkey ( -- k )
   BEGIN  dispatch key? UNTIL
   old-key execute ;

' ttkey hook-key !
```

Wir hängen den Dispatcher ins **KEY**

Demo

Bewertung

Vorteile :-)

- einfach, gut zu durchschauen und zu messen
- klein - auch für mini-Systeme einsetzbar
- vorhersagbare Programmausführung
- garantierte Reaktionszeiten: Echtzeit
- low power möglich
- Interaktivität

Nachteile :-(

- Tasks müssen Kurzläufer sein:
  ggf. Zerhacken in Teile nötig, schwierig
- Latenz ist nicht minimal

Fazit :-!

- Die richtige Technik für die richtige Aufgabe einsetzen

Diskussion