Clock Works

Die Suche nach der verlorenen Zeit . . .

Erich Wälde

Forth Gesellschaft e.V. 2017

Am Anfang war . . .

. . . eine simpel aussehende Aufgabe von Pepe:

- ich hätte gerne eine Uhr
- mit großen (Leucht–)Ziffern
- die sich selbst stellt (DCF77?)
- und die UT anzeigt

Und zwar bitte ohne Schluckauf an der Sommerzeit–Umstellung!

Ha döös iss iisii!

Man nehme:

- einen ATmega644P weil der grad rumlag
- 4 LEDs zum blinkern
- einen Uhrenquarz (32768 Hz)
- AmForth drauf
- DCF–77–Empfänger dran
- erst mal ’ne kleine LCD Anzeige
- große Ziffern kommen später

Hardware fertig!

Ha döös iss iisii!

Man nehme:

- einen ATmega644P weil der grad rumlag
- 4 LEDs zum blinkern
- einen Uhrenquarz (32768 Hz)
- AmForth drauf
- DCF–77–Empfänger dran
- erst mal ’ne kleine LCD Anzeige
- große Ziffern kommen später

Hardware fertig! oder so ich dachte . . .

Programm, vielleicht?

Ok, ohne software rührt sich bekanntlich nichts
In die verstaubten Fächer von 2006 gucken.
Been there, done that!

- multitasker
- task 1: terminal
- task 2: timeup (die Zeit zählen, jobs.tick . . . year)
- Umrechnung MEZ/MESZ $\leftrightarrow$ UT
- Die Zeit aus DCF–77 übernehmen
- Datum/Uhrzeit auf die Anzeige beamen

Fertig!

Programm, vielleicht?

Ok, ohne software rührt sich bekanntlich nichts
In die verstaubten Fächer von 2006 gucken.
Been there, done that!

- multitasker
- task 1: terminal
- task 2: timeup (die Zeit zählen, jobs.tick . . . year)
- Umrechnung MEZ/MESZ $\leftrightarrow$ UT
- Die Zeit aus DCF–77 übernehmen
- Datum/Uhrzeit auf die Anzeige beamen

Fertig! oder so ich dachte . . .

Except . . .

. . . ’s duud nedd!

- lokal: Systemzeit
- plus DCF–77–Zeit
- plus RTC–Zeit (Gedächtnis mit extra Batterie)

Und bei der Buchhaltung hab ich mich dann ordentlich auf’s Kreuz gelegt und den ganzen Plunder in die Ecke geschmissen.

Das war so im Herbst 2014.

Nochmal!

Es ist nur zur Übung, nicht zur Strafe!

Sommer 2016

The simplest thing that could possibly work!

- ATmega644P, Uhrenquarz, miniLCD, 4 LEDs
- RTC via i2c, mit PowerCap
- Uhr läuft frei, ohne Sommerzeit–Umstellung

“Clock Works 1 — die kleine Uhr” VD 2016-04

Fast gut.

Except . . .

. . . aus mir unerfindlichen Gründen ging diese Uhr nach!

- ca. 22 Sekunden / Tag!
- Notnagel: Phasen–Akkumulator
- zwar ordentlich dokumentiert
- aber man muss die Konstante im Selbstversuch bestimmen

Für eine Uhr **völlig inakzepabel!**

Ablenkung: Abakus Display

Because we can!

- Anzeige, very geek!
- Ziffern nach Abakus–Art dargestellt
- flimmerfrei dank Schieberegister
- Prototyp fertig

“Clock Works 2 — Anzeige a la Abakus” VD 2017-02

HLD

Entlang des Wegs gefunden:

```
> 60 3 * 21 + .
201 ok
> 60 3 * 21 + .
213 ok
```

- LCD–Anzeige wird per `lcd-emit` beschrieben
- aus Task 2
- wenn Task 1 auch was anzeigt
- dann kommen sich die über HLD in’s Gehege!

Nochmal . . .

. . . die verlorene Zeit suchen!

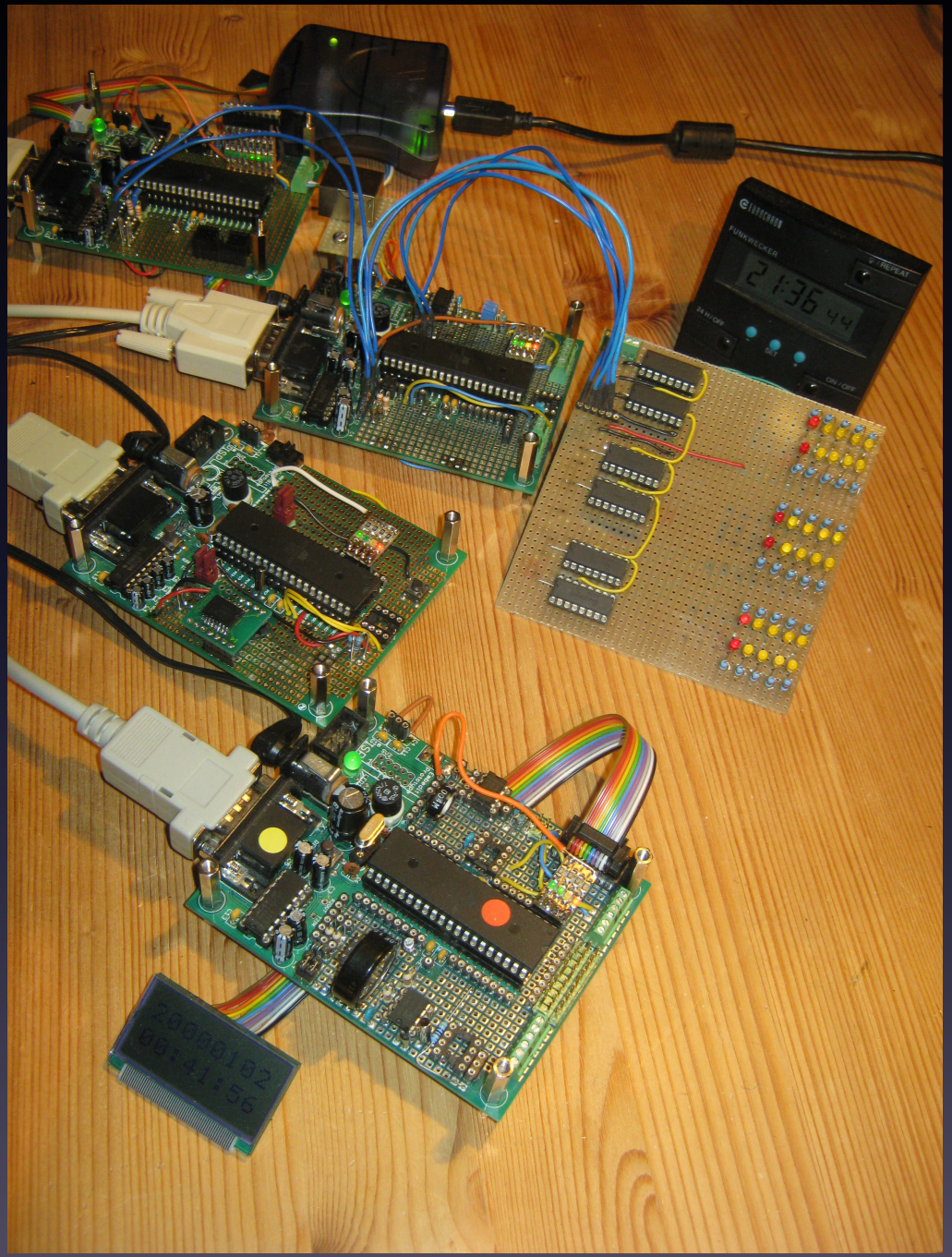
- sind die Uhren-Quarze lumpig?
- oder der ganze Kontroller?
- oder mein tolles Forth Programm?
- oder AmForth?
- ist der Algorithmus (timeup, run-jobs) fehlerhaft?
- oder die Buchhaltung?
- werden Interrupts ignoriert?

Das verlangt systematische Messreihen! Mindestens!

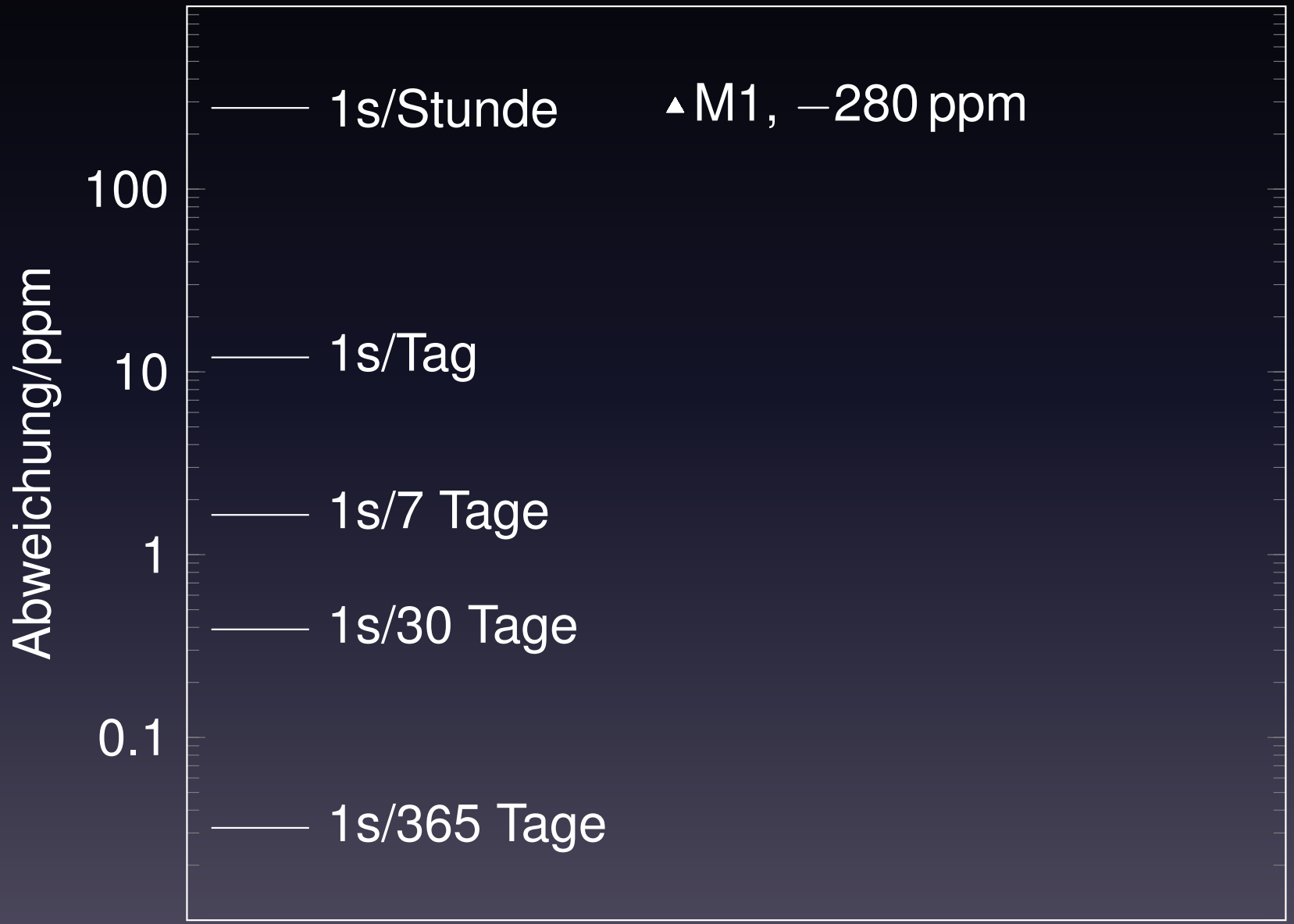
Das Zeit–Labor

- 4 Platinen
- alle unbenutzte Peripherie abgelötet
- überall ein Uhrenquarz gleicher Bauart dran
- und 4 LEDs
- Stromversorgung aus Labor-Netzteil
- Keine Displays!
- und überall das gleiche Programm drauf!

Ein Funkwecker als Vergleichsuhr
Eine Messung sollte mindestens 1 Tag dauern


FUNKWECKER
REPEAT
21:36 44
ON / OFF
28000102
00:41:56

Dimension


1s/Stunde
M1, −280 ppm
100
10
1
0.1
Abweichung/ppm
1s/Tag
1s/7 Tage
1s/30 Tage
1s/365 Tage

1. Messung

- Uhrenquarz 32768/s
- timer/counter2: Vorteiler 8, Max 128
- 32 Ticks/s

ΔT	$61	$62	$63	
151500	151456	151458	151461	s
	-44	-42	-39	diff
	290	277	257	ppm

Alle drei Uhren gehen **nach**, und erstaunlicherweise alle ungefähr gleich viel! (ca. 275 ppm)

2. Messung

- $62: 128 ticks/s
- $63: andere Bauart Uhrenquarz

ΔT	$61	$62	$63	
449100	448979	448954	448964	s
	-121	-146	-136	diff
	269	325	301	ppm
vorher:	290	277	257	ppm

128 Ticks/s vergrößern die Abweichung (325)

der smd–Uhrenquarz ist nicht besser (301)

3. Messung

Ist die ISR zu lang?

```
: tick_isr ( -- ) 1 ct.ticks +! ;
```

- die restliche Arbeit in die Hauptschleife
- viel übersichtlicher!
- $62 bekommt die kurze ISR, 32 Ticks/s

ΔT	$61	$62	$63	
87600	87575	87579	87574	s
	-25	-21	-26	diff
	285	240	296	ppm

Leider keine Verbesserung!

4. Messung

Vielleicht ist es ein *asynchron*–Thema?

- Ticks aus dem 11.059200 MHz Hauptquarz
- timer/counter1 Vorteiler 256, Maximum 1350
- 11059200/s / (256 * 1350) = 32 Ticks/s
- compare match A interrupt
- kurze ISR
- $63 bekommt das neue Programm

ΔT	$61	$62	$63	
59580	59562	59565	59563	s
	-18	-15	-17	diff
	302	252	285	ppm

No dice!

5. Messung: ATmega32

Vielleicht sind meine ATmega644P schimmlig?

- zusätzliche Platine: $64
- Anpassung von Register–/Bit–Namen

ΔT	$61	$62	$63	$64	
99300	99272	99279	99273	99256	s
	-28	-21	-27	-44	diff
	282	211	272	443	ppm

Eher schlechter!

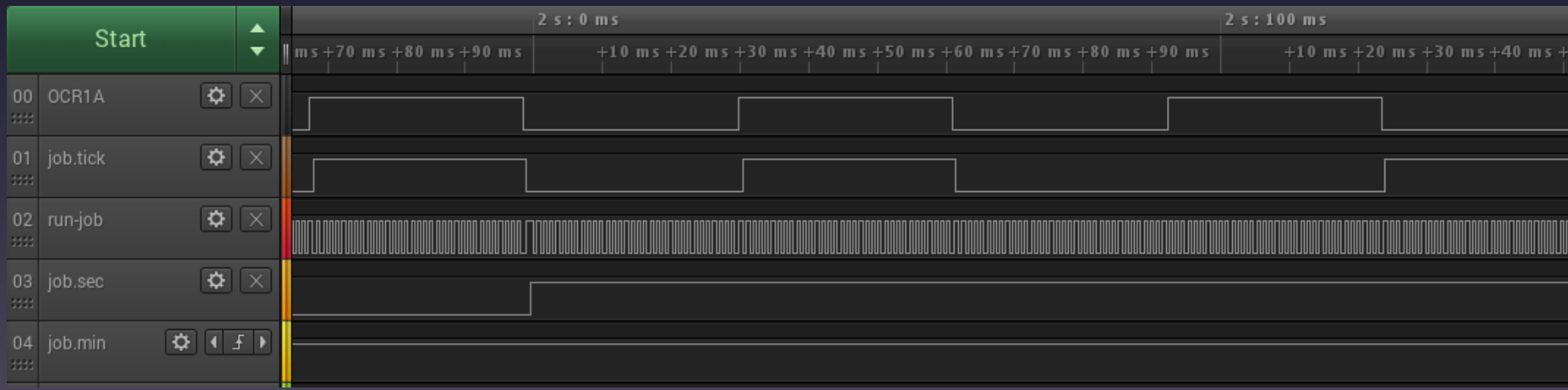
Zeitscheibchen

Vielleicht ist die einzeilige ISR immer noch irgendwie zu lang?

- 32 Ticks/s: 11059200/32 = 345600 Zyklen/Tick
- 50 Zyklen / inner interpreter loop = 6912 Runden!

Das klingt jetzt nicht so arg nach Zeit-Not . . .

Einziger Hinweis (4 mal in 300 s = ca. 400 ppm):



6. Messung: ISR in Assembler

- ISR in Assembler
- ISR erhöht eine 16-Bit Variable um 1
- die Variable ist als `ct.ticks` in Forth sichtbar
- die ursprüngliche Forth–ISR wird nicht mehr registriert!
- Station $63 bekommt das neue Programm

6. Messung: ISR in Assembler

- ISR in Assembler
- ISR erhöht eine 16-Bit Variable um 1
- die Variable ist als `ct.ticks` in Forth sichtbar
- die ursprüngliche Forth–ISR wird nicht mehr registriert!
- Station $63 bekommt das neue Programm

ΔT	$61	$62	$63	$64	
84720	84693	84700	84730	84681	
	-27	-20	+10	-39	
	319	236	118	460	ppm

Wie jetzt? Zu **schnell**? Ja wie denn das?

But . . .

. . . why?

- zu schnell ist nicht schlimm, da kann man noch Kondensatoren zum bremsen spendieren!
- Aber warum jetzt genau an der Stelle?
- Wir haben eigentlich Zeit genug, aber ISR in Assembler macht den Unterschied?
- Begründung, bitte?

Der Vollständigkeit halber

- mit einem 8–Bit–Zähler (statt 16–Bit) ergibt sich kein sichtbarer Unterschied in zwei Tagen
- die einzelnen Kontroller liegen mit 115 bis 140 ppm weiterhin ähnlich in der Streuung
- verschiedene Ticks/s Einstellungen (128,64,32,16) ergeben auch keine Unterschiede mehr

11. Messung — DS3231 RTC

- Uhrenbaustein von Maxim, TCXO, absolute Genauigkeit < 5 ppm
- nicht ganz billig
- produziert ein 32768 Hz Signal an einem Pin
- Damit kann man Timer0 und 1 treiben, aber nicht Timer2
- Pin T0 sieht das externe Clock–Signal
- Timer/Counter0 benutzt den externen Takt (fallende Flanke) als Zählimpuls
- zählt ohne Vorteiler auf 256 $\rightarrow$ 128 Ticks/s
- löst einen overflow Interrupt aus
- ISR: `lds / inc temp0 / sts`

11. Messung — DS3231 RTC

- Station $62 bekommt den RTC Baustein und das neue Programm

ΔT	$61	$62	$63	$64	
158400	158422	158400	158419	158417	s
	+22	0	+19	+17	diff
	139		120	107	ppm

Die Uhr geht jetzt *mit der Zeit*! Nach mehreren Tagen keine sichtbare Abweichung von der Funkuhr.

12. Messung: asm / Forth ISR?

zusätzlich:

- Pin T1 sieht das externe Clock–Signal
- Timer/Counter1 benutzt den externen Takt (steigende Flanke) als Zählimpuls
- zählt ohne Vorteiler auf 256 $\rightarrow$ 128 Ticks/s
- löst einen clear on compare match Interrupt aus
- ISR: `1 ct.ticks0 +!`

ΔT	$61	$62.t0	$62.t1	$63	$64	
68700	68710	68700	68673	68709	68708	s
	+10	0	-27	+9	+8	diff
	146		393	131	116	ppm

Ergebnis

Die Behandlung des Interrupts mit AmForth–Bordmitteln verliert ticks oder interrupts.

Auch wenn völlig unklar ist, wie das genau gehen soll . . .

Ideen?

Interrupt Behandlung

- interrupt request
- Instruction Pointer verzweigt
- generic ISR, eine für alle Interrupts (top half)
- die merkt sich die Nummer des aufgetretenen IRQs
- und setzt im Statusregister das T–Bit
- an dieser Stelle kein *return from interrupt*!
- das unterbrochene Wort läuft dann weiter
- bei der nächsten Runde durch den *inner interpreter* wird das T–Bit überprüft. Ist es gesetzt
- dann wird isr-exec aufgerufen
- welches das T–Bit löscht und
- die registrierte Forth-ISR aufruft (bottom half)
- isr-exit: *return from interrupt* erlaubt die Interrupts wieder

Interrupt Behandlung

- Dieses Verfahren erlaubt Forth–ISRs (statt nur asm)
- der Preis sind lange Latenzzeiten!

Können da Interrupts verloren gehen?

Eigentlich nicht, weil in der Zwischenzeit anstehende IRQs sichtbar werden, sobald die Interrupts wieder erlaubt werden.

Oder doch?

Interrupt Counts

Könnt's vielleicht sein, dass wir von *spurious* Interrupts geplagt werden?

Matthias hat die ISR aufgebohrt, und zählt mit (8 Bit)

```
~62> irqdump
 275  1 0
 276  2 0
 ...
 288 14 249    <-- timer1.compare.match.A
 293 19 255    <-- timer0.overflow
 295 21 24     <-- usart0.rx.complete
```

Auch nach Stunden keine extra Interrupts zu sehen.

Interrupt Behandlung 2

Man kann Interrupts verlieren, wenn zwischen

- generic isr (top half) und
- isr-exec (bottom half)

Das T–Bit gelöscht wird. Dann isser futsch!

Gibt's das? In der Doku steht mal nichts verdächtiges drin …

Versuch macht kluch …

Ohne T–Bit

- eine RAM Variable
- generic ISR setzt diese auf einen Wert > 0
- inner interpreter begutachtet diese Variable und verzweigt entsprechend
- isr-exec setzt den Wert der Variablen auf 0

Leider sind das jetzt ein paar mehr Instruktionen auf dem am häufigsten ausgeführten Codepfad!

Des Rätsels Lösung:

nur vier Wochen später

Das T–Bit ist lumpig!

AmForth 6.5

Matthias hat statt der Variablen ein Register spendiert.

AmForth Version 6.5 (rev. 2223)

Ich kann die Uhr jetzt mit einer simplen Forth–ISR realisieren ohne Assembler–Zusätze!

Das Problem ist nicht verstanden!

Dimension


Abweichung/ppm
100
10
1
0.1
1s/Stunde
1s/Tag
1s/7 Tage
1s/30 Tage
1s/365 Tage
M1, −280 ppm
M7, +120 ppm
M22, +7 ppm, 20 pF

NeXT . . .

Hatte ich erwähnt, dass dieses Projekt fröhlich in allerlei Richtungen explodiert ist?

Anzeigen:

- 4 x große 7–Segment Anzeige (HH:MM)
- 20 x kleine 7-Segment Anzeige (YYYY-mm-dd HH:MM:SS) oder Epochensekunden
- Morse Piepser
- 60 RGB–LEDs im Kreis
- Strichlaser auf Servo
- Schrittmotoren und Zeiger
- Platinen?

NeXT . . .

Hatte ich erwähnt, dass dieses Projekt fröhlich in allerlei Richtungen explodiert ist?

Andere Zeiten:

- UT und andere Zeitzonen, auch mit Sommerzeit
- Epochensekunden
- Sternzeit
- internet beats

NeXT . . .

Hatte ich erwähnt, dass dieses Projekt fröhlich in allerlei Richtungen explodiert ist?

Andere Funktionen:

- Wecker
- elektrischer Kuckuck
- Erinnerungs–Butler (nagbar)
- Sensorik?
- RS485–Bus Anschluss?
- die Konstante `ct.fcontrol` automatisch aus der DCF-77 Zeit ermitteln?

Es wird mir nicht langweilig werden! Happy Forthing!

Dank

geht mindestens an

- Matthias Trute (of AmForth fame!)
- Bernd Paysan (of net2o and more fames!)
- Martin Bitter (of Wunderland fame!)
- Frank Eisele

und alle, die ich mit dieser Geschichte belästigt habe!

slide set generated with LaTeX/Beamer(Manhattan.sty)