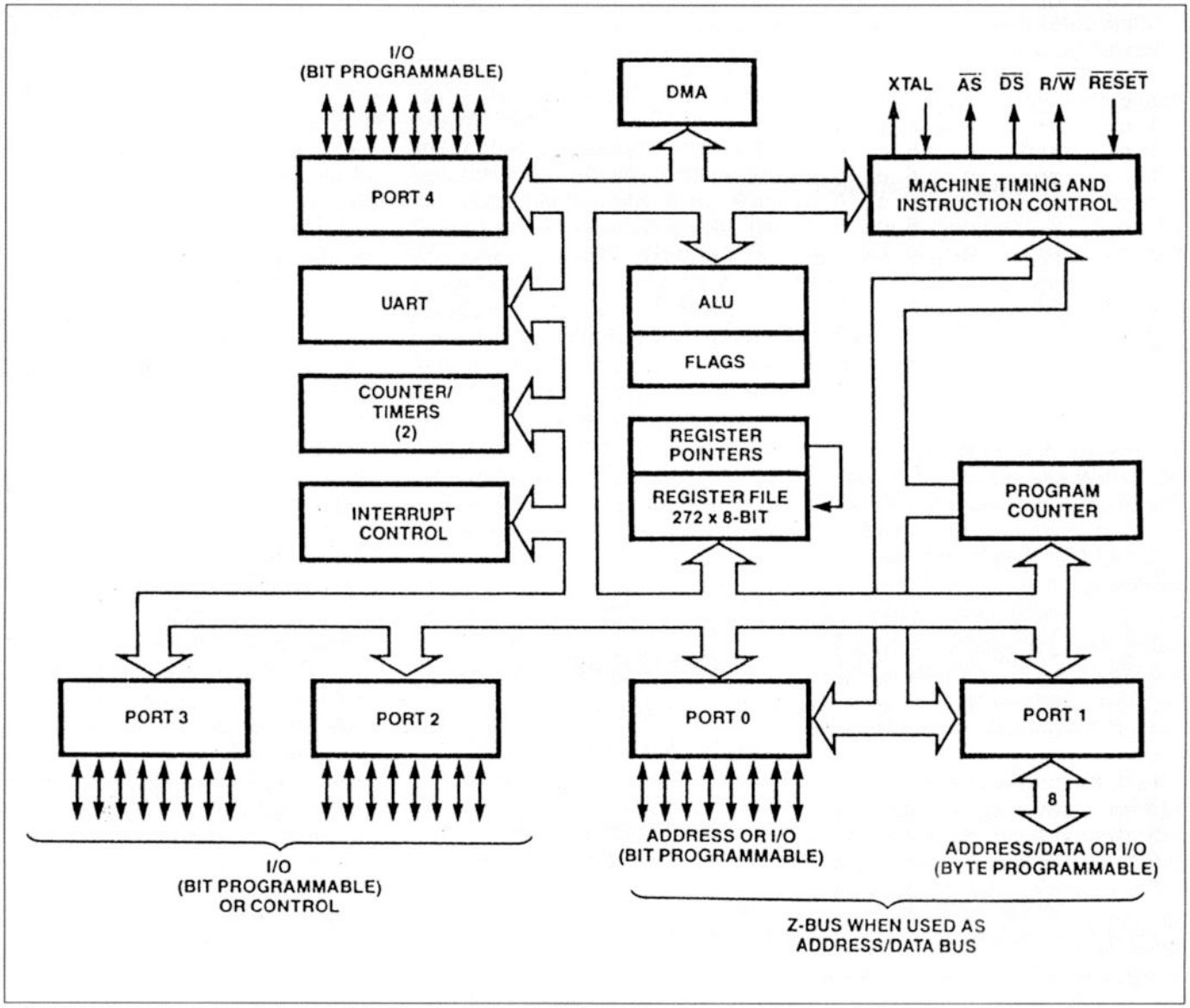
Platine für den Zilog Super8-FORTH-Chip

Heinz Schnitter und Hans-Günther Willers

Der Super8 Microcomputer von Zilog ist ein Nachfolger des Z8. Er zeichnet sich durch die bewährte Register-Architektur, sowie durch den integrierten DMA-Kanal und die schnelle, leistungsfähige Interruptstruktur mit insgesamt 27 möglichen Interruptquellen aus [1] [2].

Als Erweiterung des Befehlssatzes sind Divisions- und Multiplikationsbefehle, sowie die Instruktionen ENTER, EXIT und NEXT implementiert. Die drei letztgenannten Befehle sind genau die, die in den virtuellen FORTH-Maschinen die meiste Rechenzeit verbrauchen. Durch die Implementierung als Maschinenbefehle wird ein FORTH auf dem Super8, das von diesen Gebrauch macht, deutliche Geschwindigkeitsvorteile verbuchen können.



BLOCKSCHALTBILD DES SUPER8 <COPYRIGHT ZILOG INC>

Aus diesen Gründen wurde von Dezember 1989 bis Juni 1990 von zwei Mitgliedern der FORTH-Gesellschaft e.V. (Klaus Kohl und Heinz Schnitter) ein FORTH für den Super8 geschrieben und von Zilog in das ROM einer Super8-Maske übernommen (Zilog 0887520PSC).

Damit nicht jeder Anwender des Super8-FORTH-Chips bei seinen ersten Versuchen das "Rad neu erfinden" muß, wurde eine sehr kompakte Platine mit den notwendigsten Bauteilen entwickelt (Maße 64 x 54 mm). Auf ihr befindet sich alles, was benötigt wird, um mit einem industriekompatiblen PC (oder einem Terminal) FORTH-Programme entwickeln und testen zu können.

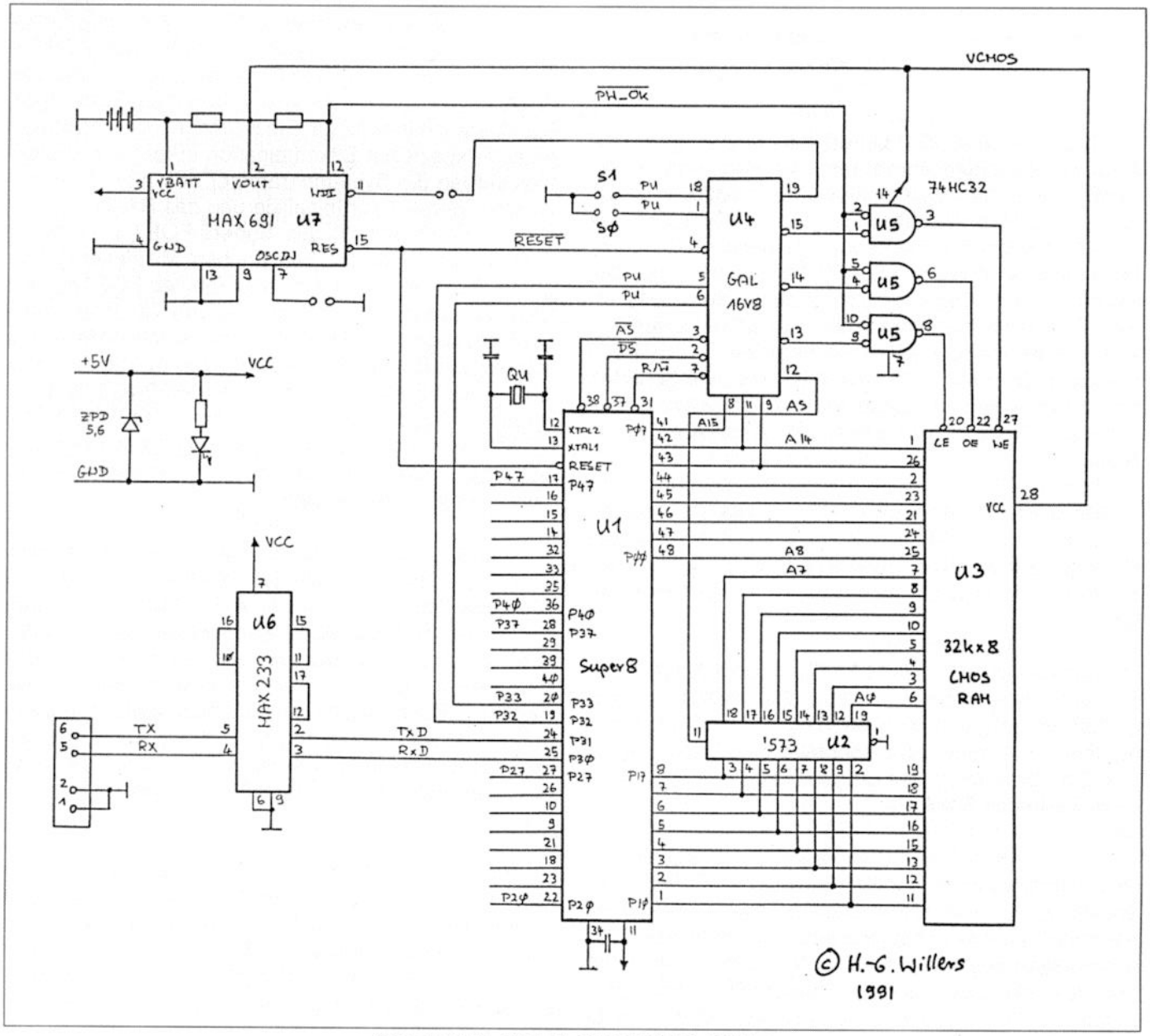
Die Hardware

Herz der Schaltung [4] ist natürlich der Super8 mit dem FORTH im Masken-ROM. Von den fünf 8-Bit Ports des Super8 werden Port0 für die Adressen 8 bis 15, sowie Port1 als gemultiplexter Daten- und Adreßbus verwendet. Während der Adreßphase werden die Adreßbits 0 bis 7 am Bus angelegt und im Latch U2 gehalten.

Alle Port-Bits, sowie die wichtigsten Steuersignale sind auf Pfostenstecker geführt, wo sie zur Adaptierung bereit stehen. Über die Pins der Pfostenstecker wird ebenfalls Masse und die Versorgungsspannung von +5V zugeführt.

Zur Speicherung von Programm und Daten dient das CMOS-RAM U3. Dieses RAM, mit einer Größe von 32 KByte, wird von einem NiCd-Akkumulator auch bei abgeschalteter Versorgungsspannung gepuffert, so daß Programme über längere Zeit (ca. 3 Monate) gespeichert bleiben.

Über U5 werden die Signale 'Chip-Select', 'Output-Enable' und 'Write' des CMOS-RAMs bei abgeschalteter Versorgungsspannung auf inaktivem Pegel gehalten, so daß der Inhalt des RAMs nicht verfälscht wird.



SCHALTBILD DER SUPER8-PLATINE

Im GAL U4 wird die Dekodierung, sowie ein durch Steckbrücken einstellbares Write-Protect gemacht.

Die Gleichungen für die einzelnen Signale sind wie folgt:

```
!ASOUT = ASIN;

  !CS   = A15;

  !OE   = A15 & !DS & WRITE;

  !WR   = A15 & A14 & A13 & !DS & !WRITE
        # A15 & A14 & !S0 & !DS & !WRITE
        # A15 & A14 & !S1 & !DS & !WRITE
        # A15 & A13 & !S1 & !DS & !WRITE
        # A15 & !S0 & !S1 & !DS & !WRITE;

WD.OE = XENABL;

  WD    = P32;

XENABL = RESET & !DS & !WRITE & P32 & !P33
       # RESET & XENABL;
```

Das Super8-FORTH benötigt am oberen Speicherende des 64 KByte Adreßraums ein RAM von mindestens 2 KByte; auf der Platine wird ein CMOS-RAM mit 32 KByte verwendet. Das On-Chip-ROM beginnt bei der Adresse 0 und ist 8 KByte groß. Zwischen On-Chip-ROM und RAM kann ein EPROM oder wie auf der Platine ein schreibgeschützter Bereich des RAMs liegen. Dieser nichtflüchtige Speicher muß auf einer 1 KByte-Grenze beginnen. Innerhalb des CMOS-RAMs kann eine Grenze (RAM-FENCE) in 8 KByte-Schritten mit Steckbrücken eingestellt werden. Unterhalb dieser Adresse RAM-FENCE verhält sich das RAM wie ein ROM.

Mit dem RS-232-Konverter U6 (MAX233) werden die Signale des UARTs des Super8 auf die üblichen Pegel für serielle Schnittstellen gebracht. Die Schnittstellensignale sind auf eine 6-polige Pfostenleiste geführt.

Der Baustein U7 (MAX691 [5]) erzeugt das RESET-Signal für den Super8. Die Umschaltung der Versorgungsspannung von U3 (CMOS-RAM) und U5 auf Batteriespannung geschieht ebenfalls in U7. Das Signal PW_OK (aktiv Low) gibt die Steuersignale des GALs über U5 an das RAM frei.

Die Überwachung der Funktionstüchigkeit von Steuerungsrechnern wird üblicherweise mit Watchdog-Schaltungen sichergestellt. Hierbei muß sich der Rechner periodisch beim Watchdog melden, ansonsten wird das System rückgesetzt. Solch ein Watchdog ist im MAX691 integriert. Er wird durch einen High- oder Low-Pegel am Pin 11 des MAX691 aktiviert. Nach Aktivierung muß am Pin 11 periodisch (je nach Beschaltung von Pin 7 alle 1.6 Sekunden oder 100ms) eine Signalflanke auftreten; bleibt sie aus, wird der Super8 rückgesetzt. Aktiviert wird der Watchdog über eine Kombination von P32 und P33. Die Watchdog-Funktion läßt sich nicht mehr abschalten falls sie einmal aktiviert ist.Dadurch wird dem Programmierer keine Möglichkeit gegeben, den Watchdog zu "überlisten", in dem er ihn kurzfristig ausschaltet. Falls dann vergessen würde, den Watchdog wieder zu aktivieren, wäre die Überwachung auf Funktionstüchtigkeit des Rechners hinfällig.

Die Software

Resetfest programmieren - aber ohne EPROM-Programmiergerät.

Die inkrementelle Programmierumgebung FORTH wird von der Hardware optimal unterstützt. Das Super8-FORTH durchsucht in seiner Startroutine den 64 KByte Adreßraum vom Speicherende in Richtung niedrigere Adressen nach einem RAM und nach einem Bitimage in einem evtl. vorhandenen EPROM. Wird kein Image gefunden, werden die Sytemparameter des FORTH-Kerns aus dem ON-Chip-ROM an den Anfang des RAMs kopiert und die Task- und Variablenbereiche am Speicherende initialisiert. Ein Bitimage im EPROM wird an einer speziellen Bitkombination erkannt. In diesem Fall werden die Systemparameter des hier abgespeicherten Systems zur Initialisierung des RAMs verwendet. Im Handbuch [3] des Super8-FORTH ist dieser Sachverhalt ausführlich beschrieben. Mit dieser Methode hat Klaus Kohl, für ein ROM-fähiges FORTH, eine elegante Lösung zur Verwaltung von variablen Speicherbereichen im RAM gefunden. SAVE-RAM (siehe beigefügten Quelltext) speichert nach Aufruf den aktuellen Zustand ab und gibt die RAM-FENCE und die höchste Adresse des Programms an. Die neue RAM-FENCE wird dann mittels Jumper eingestellt. Eine Warnung erhält man, falls das Programm nicht mehr in den vorhandenen Speicher paßt.

Braucht man z.B. für das zu entwickelnde Programm den S8-Assembler [3], so kann man diesen in das RAM laden, und mit SAVE-RAM und Umstecken der Jumper sichern. Bei einem Programmfehler, während der Entwicklungsphase von neuen Worten, der zum Absturz führt (Normalfall), braucht nach einem Hardware-Reset der Assembler nicht mehr neu geladen werden. Falls die neuen FORTH-Worte wie gewünscht funktionieren, können auch sie mit der oben beschriebenen Prozedur nichtflüchtig gespeichert werden.

Bedingt durch die Hardware (RAM-Größe 32 KByte) kann dieser Vorgang bis zu drei mal wiederholt werden, in dem man in 8 KByte-Schritten die RAM-FENCE nach oben verschiebt. Es stehen dann dem Super8-FORTH bis zu 24 KByte "ROM" im CMOS-RAM und mindestens 8 KByte RAM am Speicherende zur Verfügung. Die eventuell dadurch entstehenden Lücken im Speicher gehen zwar im Moment in der Programmentwicklung

verloren; sind jedoch die so entstandenen Worte ohne Fehler lauffähig, "jumpert" man auf der Platine RAM-FENCE auf die RAM-Startadresse (HEX 08000), lädt alle Teile zusammenhängend in den Super8 und sichert das System wie beschrieben. Beim Neustart des Super8-FORTH wird bei der Initialisierung immer der zuletzt gesicherte Systemzustand gefunden und als aktuelles System in Betrieb genommen.

Es ergibt sich so eine Bottom-Up-Strategie, bei der aufbauend auf Basisfunktionen neue Worte hinzugefügt und nichtflüchtig gespeichert werden, bis das fertige Programm erstellt ist.

Eine Art Hardware-FORGET ist möglich, in dem man mit den Jumpern eine RAM-FENCE einstellt, unterhalb der bereits ein System gesichert worden ist. Nach Reset oder COLD arbeitet man mit dem FORTH-System, das in diesem Bereich abgespeichert wurde.

An einem Beispiel wird die Programmierung des auf der Platine befindlichen Watchdog gezeigt. Dieser Chip erzeugt beim Einschalten der Platine und beim Ablaufen des Wachtdog-Timers einen Reset-Impuls.

Die Aufgabenstellung ist, das FORTH-System in einen Zustand zu bringen, der es ermöglicht, daß bei einer Endlosschleife (das System hat sich "aufgehängt") ein Reset-Impuls generiert wird.

Der Watchdog wird in KEY? und EMIT? nachgetriggert. Um das Auslösen eines Reset-Impulses zu verhindern, dürfen Worte, die KEY? oder EMIT? nicht aufrufen, keine längere Laufzeit als 1,6 Sekunden haben. Dies hat den Effekt, daß die serielle Schnittstelle mindestens alle 1,6 Sekunden bedient werden muß.

Mit WDOG_ON wird der Wachhund scharf gemacht und mit WDOG oder in diesem Beispiel in KEY? und EMIT? immer wieder beruhigt. In einem Multitask-FORTH-System wäre es sicherlich sinnvoll WDOG auch in PAUSE einzubauen. Die Phrase

```
' START_WDOG IS 'COLD
```

in Screen 6 klinkt START_WDOG in COLD ein und SINGLETASK belegt PAUSE mit NOOP. Beim Ausführen von COLD wird dadurch mit START_WDOG das Scharfmachen des Wachhundes veranlaßt, sowie KEY und EMIT auf die neuen I/O-Vektoren umgeleitet.

Mit dem Wort MS kann die Funktion überprüft werden. MS hat eine Laufzeit von ziemlich genau einer Millisekunde. Durch Probieren läßt sich feststellen, daß der Watchdog ab ca. 1,6 Sekunden einen Reset-Impuls generiert.

Diesen Artikel verstehen die Autoren als einen Beitrag zum Rapid Prototyping. Wir wollten zeigen, daß die On-Chip- und On-Board-Hardware schnell und ohne viel Aufwand in FORTH und Assembler programmiert werden kann.

Literaturverzeichnis

[1] Super8 Microcontroller mit 8-Bit-Archi tektur und Hochsprachen-Unterstützung, Werner Meininger, Design&Elektronik, Nr. 5/1986

[2] Super8 Microcomputer Product Specification, 1987
Super8 Microcomputer Technical Manual, 1987
Z8 Family Design Handbook, 1989
Zilog Inc. 210 Hacienda Ave., Campbell, California 95008-6609

[3] Handbuch zum Super8-FORTH V1.0, 1991, Klaus Kohl, Heinz Schnitter

[4] Datenblatt der Super8-Platine, 1991, Hans-Günther Willers

[5] Datenblatt MAX691, MAXIM Databook, 1990

```
Screen # 1

\ RAM-FENCE?                                        hfs 11:33 18.08.91
HEX
| : RAM-FENCE?  ( length -- t|f )
    BASE PUSH HEX                                   \ save BASE
    RAMORG @ +                                      \ höchste Adresse
    DUP  0E000 U<                                   \ zu lang?
    IF   0A000                                      \ erste Möglichkeit
         BEGIN OVER OVER U< NOT
         WHILE 2000 +                               \ 8KB Schritte
         REPEAT CR ." RAM-FENCE: " U.     TRUE
    ELSE 7 EMIT CR ." Programm zu lang " FALSE
    THEN SWAP   CR ." Prog. End: " U.
  ;




Screen # 2

\ SAVE-RAM                                          hfs 11:33 18.08.91
HEX
: SAVE-RAM  ( -- )
  RAMORG DUP 2- SWAP @  50 CMOVE    \ Zustand speichern
  HERE         RAMORG @  -          \ Programmlänge
  TASKORG   @ VDP    @  - DUP >R + \ + Variablen- + Heap-Länge
  0D0          TASK#  @  - 8 *    + \ + 80*Tasks = Gesamtlänge
  RAM-FENCE?                       \ paßt es ins RAM?
  IF   VDP @ HERE R@ CMOVE         \ Variablen und Heap
       HERE R> +                   \ RAM-addr
       LASTTASK  0CF  TASK# @
       DO @ DUP >R OVER 80 CMOVE   \ eine Task um die andere
          80 + ( RAM-addr ) R> 1+
       10 +LOOP DROP DROP
  ELSE RDROP THEN ;




Screen # 3

\ wdog_on wdog                                      hfs 11:33 18.08.91
HEX
  CODE WDOG_ON
       P3 , # 04 OR,              \ P32 High WDOG einschalten
       NEXT,
  END-CODE

  CODE WDOG
       P3 , # 04 XOR,             \ P32 toggeln WDOG streicheln
       NEXT,
  END-CODE
```

```
Screen # 4

\ wd_emit? (wd_emit                              hfs 11:37 18.08.91
HEX
  CODE WD_EMIT?
       P3  , # 04 XOR,              \ P32 toggeln WDOG streicheln
       AX PUSHW          AL , UTC LD,
       AL , # 02 AND,    Z  , 1$ JP,
       AX , # -1 LDW,    NEXT,
  1$:  AX , # 00 LDW,    NEXT,
  END-CODE

  CODE (WD_EMIT
       UIO , AL LD,
       AX POPW           NEXT,
  END-CODE




Screen # 5

\ wd_key? (wd_key                                hfs 11:37 18.08.91
HEX
  CODE WD_KEY?
       P3  , # 04 XOR,              \ P32 toggeln WDOG streicheln
       AX PUSHW          AL , URC LD,
       AL , # 01 AND,    Z  , 1$ JP,
       AX , # -1 LDW,    NEXT,
  1$:  AX , # 00 LDW,    NEXT,
  END-CODE

  CODE (WD_KEY
       AX PUSHW
       AL , UIO LD,
       AH , # 0 LD,
       NEXT,
  END-CODE



Screen # 6

\ wd_key wd_emit                                 hfs 11:33 18.08.91
HEX
  : WD_KEY  BEGIN WD_KEY?  0= WHILE  PAUSE  REPEAT (WD_KEY  ;
  : WD_EMIT BEGIN WD_EMIT? 0= WHILE  PAUSE  REPEAT (WD_EMIT ;

  INPUT:  WD_INPUT  WD_KEY?   WD_KEY  ;
  OUTPUT: WD_OUTPUT WD_EMIT?  WD_EMIT ;

  : START_WDOG
    WDOG_ON WD_OUTPUT WD_INPUT ;

  : MS 0 DO 1F 0 DO LOOP LOOP 7 EMIT ;

  ' START_WDOG IS 'COLD   SINGLETASK    DECIMAL
```