

Programmierung der I/O

Eine vollständige Beschreibung der I/O-Programmierung des GP32 ist nicht angestrebt. Für erste Gehversuche mit nanoFORTH sind einfache Programmschnipsel aber hilfreich. Dabei ist an Port PA ein D/A-Wandler mit R2R-Netzwerk nützlich.

Ports

Die Pins sind nach Reset auf Input geschaltet. Sie müssen erst im Direction-Register DPx auf Output geschaltet werden.

```
<| \ PORT.F08

B% 11111111 PA C! \ all pins high on PA
B% 11111111 DPA C! \ all pins output on PA

DPB 0 B0! \ pin DPB 0 low
PB 0 B@ \ read PB 0

|>
```

A/D-Wandler

Als Initialisierung muß man den Takt einstellen.

Eine Wandlung besteht aus Kanal wählen und damit starten. Flag abfragen um das Ende abzuwarten. Daten auslesen.

Abschalten des Wandlers ist optional, spart Strom.

Das Programm wird ab FCODE ins RAM compiliert. Dabei ist zu beachten, daß solche Testprogramme nicht zu lange werden dürfen, da der Compiler nicht erkennt ob das Ende des RAMs bei 023F überschritten wurde.

Hier wird kontinuierlich das Signal an PB 0 gewandelt und auf Port PA ausgegeben wo ein D/A-Wandler nützlich ist.

```
<| \ ADC.F08

HEX FCODE >C ! \ compile to RAM

:CODE ADC@ \ ( - C1 )read PB0
3E B% 01010000 #. MOV, \ 2,56 MHz /4 = 0,64MHz clock
3C B% 00000000 #. MOV, \ switch ADC on,select channel,
\ start conversion
\ ^ 0 = single conversion
\ 000 PB0
\ ... ...
\ 111 PB7

1 $: 1 $ 3C 7 BBC, \ wait for ready flag set
3D LDA, \ read data
3C 1F #. MOV, \ switch ADC off
DEX, 0,X STA, \ put data on stack
A. CLR, DEX, 0,X STA,
RTS,

CODE;
|>
<| \ ADDA.F08
HEX FCODE >C !

:CODE RUN
DPA FF #. MOV, \ PA to output
3E B% 01010000 #. MOV, \ clock for A/D :
\ 2,56 MHz /4 = 0,64 MHz

2 $:
3C B% 00000000 #. MOV, \ start conversion PB0
1 $: 1 $ 3C 7 BBC, \ wait for end of conversion
PA 3D MOV, \ copy data to PA
2 $ BRA,
CODE;
|>
```

TIMER

Hier löst ein Timer einen zyklischen Interrupt aus in dem der A/D-Wandler gelesen und der Wert auf PA ausgegeben wird.

Die Umlenkung des Interruptvektors in SYS-EE erfolgt auf die feste Adresse FCODE bevor ins RAM kompiliert wird. Durch das Schreiben des EEPROMs wird ja das RAM zerstört.

Die Interruptroutine an FCODE hat keinen Header und Namen. Damit aber Assembler kompiliert werden kann muß dieser erst mit [CODE zugeschaltet werden. Das Rückschalten mit CODE] entfällt hingegen, da dadurch auch auf Compilermodus geschaltet würde. Wenn sich in der Routine also relative Sprungbefehle befinden ist explizit ein SOLVE\$ nötig.

```
<| \ SAMPLE.F08

\ Vector in SYS-EE to FCODE
CC 80D8 EC! \ JMP,-Opcode
FCODE 80D9 E!

FCODE >C ! \ compile to RAM

[CODE \ select assembler
PD 3 MBS, \ Pin PD 3 high
2B 7 MBC, \ clear IRQ-Flag
PA 3D MOV, \ copy data to PA
3C B% 00000000 #. MOV, \ start conversion PB0
PD 3 MBC, \ Pin PD 3 low
RTI,
\ CODE]

:CODE RUN
2E CLR, \ T2MODH samplerate
2F D% 77 #. MOV, \ T2MODL
2B B% 01000000 #. MOV, \ T2SC prescaler
\ 000 /1 2,45 MHz
\ 001 /2
\ 010 /4
\ 011 /8
\ 100 /16
\ 101 /32
\ 110 /64
\ 111 illegal
\ ^ TO2E interrupt enabled

DPA FF #. MOV, \ PA to output
3E B% 00110000 #. MOV, \ clock for A/D :
\ 2,45 MHz /4 = 0,61 MHz
DPD 3 MBS, \ PD 3 to output
CLI, \ interrupt enabled
1 $: 1 $ BRA, \ infinite loop
CODE;

|>
```

Das Programm ohne A/D-Wandler nochmal aber diesmal wird nicht ins RAM sondern ins FLASH kompiliert. Dann macht auch der SaveMarker | Sinn.

Außerdem wird hier ein anderer Timer verwendet.

```
<| \ TIRQ.F08
HEX
:CODE TIRQ
    20 7 MBC, \ T1SC clear interruptflag
N INC, N LDA, PA STA, \ increment N , write to PA
    RTI,
CODE;

:CODE RUN \ timerinterrupt
23 CLR, \ T1MODH T1MOD = 0004
24 4 #. MOV, \ T1MODL
20 B% 01000101 #. MOV, \ T1SC of Timer prescaler
\ 000 /1 2,45 MHz
\ 001 /2 1,225 MHz
\ 010 /4 612 kHz
\ 011 /8 306
\ 100 /16 153
\ 101 /32 76,5
\ 110 /64 38,2
\ 111 illegal
\ ^ TOIE interrupt enabled

DPA FF #. MOV, \ Port PA to output
    CLI, \ interrupt enabled
1 $: 1 $ BRA, \ infinite loop
CODE;

\ Vector in SYS-EE to TIRQ
CC 80E1 EC! \ JMP,-Opcode
^ TIRQ 80E2 E!

| \ SaveMarker
|>
```

Hier wird der Timer durch Polling abgefragt, er zählt kontinuierlich 0 - FFFF.

Es müssen immer und in dieser Reihenfolge beide Bytes von TxCNT gelesen werden.

```
<| \ TIMER.F08
HEX
FCODE >C ! \ compile to RAM
:CODE RUN \ ( - )
23 FF #. MOV, \ T1MODH T1MOD = FFFF
24 FF #. MOV, \ T1MODL
20 B% 00000101 #. MOV, \ T1SC of Timer
\ 000 /1 2,45 MHz
\ 001 /2
\ 010 /4
\ 011 /8
\ 100 /16
\ 101 /32
\ 110 /64

DPA FF #. MOV, \ PA to output
1 $: 21 TST, \ T1CNTH HB lesen
    22 LDA, \ T1CNTL LB lesen
    PA STA, \ write to PA
1 $ BRA,
CODE;

|>
```

Ein PWM-Signal PD4.
 Da innerhalb von RUN das X-Register nicht verändert wurden entfällt sein Speichern und laden in XSAVE.

```
<| \ PWM.F08
HEX
FCODE >C ! \ compile to RAM

:CODE RUN
20 B% 00110000 #. MOV, \ T1SC
\      ^      TSTOP
\      ^      TRST
23 01 #. MOV, \ T1MODH period
24 FF #. MOV, \ T1MODL
26 00 #. MOV, \ T1CH0H pulswidth
27 00 #. MOV, \ T1CH0L

20 B% 00000000 #. MOV, \ T1SC select prescaler
\      000 /1      2,45 MHz
\      001 /2
\      010 /4
\      011 /8
\      100 /16
\      101 /32
\      110 /64

25 B% 00101010 #. MOV, \ T1SC mode of operation
\      ^ TOV1 toggle on overflow
\      ^^^^ clear output on compare
\      ^ link Channel 0 and 1 for buffered PWM
          RTS,
CODE;

|>
```

Man kann natürlich auch simple Rechtecksignale hier z.B. 125kHz erzeugen.

```
<| \ 125KHZ.F08
\ Bus = 2,45 MHz

FBUF >C ! \ compile to RAM

:CODE RUN
\ ca. 125kHz on PD4
0020 B% 00110000 #. MOV, \ T1SC Stop reset
0023      00 #. MOV, \ T1MODH
0024 D% 09      #. MOV, \ T1MODL

25 B% 10010100 #. MOV, \ T1SC0
\      ^ Input Capture or Output Compare on this channel
\      ^ Interrupt disabled
\      ^ Buffered output compare disabled
\      ^ Unbuffered output compare enabled
\      ^^ Toggle
\      ^ Pin Toggles on overflow
\      ^ Duty Cycle

0020 B% 00000000 #. MOV, \ T1SC Prescaler /1
          RTS,
CODE;

|>
```

Autostart

Vor dem Sprung über den Vektor 80F3 wird nur der Stackpointer der CPU initialisiert:

```
:CODE COLD
    00FF 1+ #. LHX, TXS,      \ Load SP
                                SEI,      \ ( automatic )
\ J?                            JSR,      \ Jumper ?
1 $                              BCS,
    80F3                        LDA,      \ Autostart ?
                                FF #. CMP,
1 $                              BEQ,
    80F3                        JSR,
```

1 \$: SOLVE\$

In einem Anwendungsprogramm das nach Reset direkt starten soll sind also noch einige andere Initialisierungen vorzunehmen:

```
\ CONFIG-Register
01 #. LDA, 1E STA, \ CONFIG2'   UART-Baudrate = Bus
01 #. LDA, 1F STA, \ CONFIG1'   Watchdog disabled

\ Clock to 2,45 MHz
    36 5 MBS,      \ PCTL PLL-OFF
36 B% 00000000 #. MOV,      \ PCTL P = 0
                                \ PCTL E = 1
38          01 #. MOV,      \ PMSH
39          2C #. MOV,      \ PMSL
3A          80 #. MOV,      \ PMRS
3B          01 #. MOV,      \ PMDS
                                36 5 MBS,      \ PCTL PLL-ON
                                37 7 MBS,      \ PBWC auto bandwidth
3 $: 3 $          37 6 BBC,      \ wait for PLL to lock in
                                36 4 MBS,      \ PCTL VCO = sysclock

\ RAM to 00
    0000 #. LHX,
                                TXA,
1 $:          40 ,X STA,
                                INX,
1 $          BNE,
2 $:          140 ,X STA,
                                INX,
2 $          BNE,
3 $:          240 ,X STA,
                                INX,
3 $          BNE,
SOLVE$
```

```
\ Pseudoopcodes im RAM für FORTH
81 #.LDA, \ RTS,
NOPC 3+ STA,
NOPC 6+ STA,

\ FORTH-Stackpointer
00DC #. LHX,

\ Delay für 2,45 MHz für Befehl MSEC
D% 64 #.LDA, UDLY STA,

\ LED on
0C 0 MBS, \ PE 0 = output
08 0 MBS, \ PE 0 = high LED = on

\ init UART
13 6 MBS, \ SCC1 ENSCI enable SCI
14 3 MBS, \ SCC2 TE enable transmit
14 2 MBS, \ SCC2 RE enable receive

\ baud rate UART 9600 Baud für 2,45 MHz
19 20 #. MOV,
```