

noForth T en I²C



16-bits EEPROM lees actie

De I²C primitieven

Eerst een overzicht van de I²C woordenset in “Egel” en “Project Forth Works” stijl.

I2C-ON

Activeer I²C driver

{I2C-WRITE

Open I²C om +n bytes te schrijven

{I2C-READ

Open I²C om +n bytes te lezen

I2C}

Sluit I²C lees of schrijf operatie

BUS!

Zend byte via de I²C bus

BUS@

Lees byte via I²C bus

DEVICE!

Zet I²C device adres

{DEVICE-OK?}

True als een device in gebruik/actief is

GPIO functies

Elke pin krijgt een functie, deze tabel toont de mogelijkheden.
Default staat elke pin op functie-5, dat is standaard I/O.

	Function								
GPIO	F1	F2	F3	F4	F5	F6	F7	F8	F9
0	SPI0 RX	UART0 TX	I2C0 SDA	PWM0 A	SIO	PI00	PI01		USB OVCUR DET
1	SPI0 CSn	UART0 RX	I2C0 SCL	PWM0 B	SIO	PI00	PI01		USB VBUS DET
2	SPI0 SCK	UART0 CTS	I2C1 SDA	PWM1 A	SIO	PI00	PI01		USB VBUS EN
3	SPI0 TX	UART0 RTS	I2C1 SCL	PWM1 B	SIO	PI00	PI01		USB OVCUR DET
4	SPI0 RX	UART1 TX	I2C0 SDA	PWM2 A	SIO	PI00	PI01		USB VBUS DET

GPIO functie tabel

Hardware of bit-bang I²C master

Activeer de I²C bus, eerst de bit-bang versie:

```
: I2C-ON      ( -- )
    5 0C gpio! 5 0D gpio! \ Use normal I/O on GPIO12 & GPIO13
    4A 0C pads! 4A 0D pads! \ Set GPIO12=SDA & GPIO13=SCL with pull up
    bus D000,0020 **bic      \ I2C bus GPIO are inputs (high)
    bus D000,0010 **bic ;    \ Initialise bus outputs to low

: WAIT        ( -- )      0D for next ; \ About 200 KHz with 125 MHz clock
```

Hetzelfde, maar voor de ingebouwde I²C hardware:

```
40044000 value 'I2C      \ I2C0_BASE      I2C0 base register
: I2C-ON      ( -- )
    3 0C gpio! 3 0D gpio! \ I2C0 on GPIO12 & GPIO13
    4A 0C pads! 4A 0D pads! \ Set GPIO12=SDA & GPIO13=SCL with pull up
    1 'i2c 6C + **bic      \ Disable I2C
    dm 240 'i2c 1C + !      \ Set high & low clock period (~200kHz)
    dm 294 'i2c 20 + !
    dm 12 'i2c A0 + !      \ Spike suppressing to 96 ns (7 for high speed)
    0065 'i2c !            \ 7-bit master, fast speed, restart & slave off
    1 'i2c 6C + **bis ;    \ Enable I2C
```

Hardware I²C master

De documentatie van de I²C hardware was wat betreft de register beschrijving was vanwege de overlap niet direct duidelijk voor me. Hier een vertaald overzicht van de beide I²C status registers.

I²C Interrupt Status Register

12	0	Herstart interrupt actief hoog
11	0	General call interrupt actief hoog
10	0	Start interrupt is actief hoog
9	0	Stop interrupt is actief hoog
8	0	Lees activiteit interrupt is actief hoog
7	0	Lezen klaar interrupt is actief hoog
6	0	Zenden mislukt interrupt is actief hoog
5	0	Lees aanvraag interrupt is actief hoog
4	0	Zenden klaar interrupt is actief hoog
3	0	Zend buffer overflow interrupt is actief hoog
2	0	Lezen afgerond interrupt is actief hoog
1	0	Lees buffer overflow interrupt is actief hoog
0	0	Lees buffer leeg interrupt is actief hoog

I²C Status Register

6	0	Slaaf is actief
5	0	Meester is actief
4	0	Ontvangst buffer is vol
3	0	Ontvangst buffer niet leeg
2	1	Zend buffer is leeg
1	1	Zend buffer niet vol
0	0	De I2C bus is nog bezig

Voor de volgende test wordt een I²C implementatie gebruikt, die de beide status registers een aantal malen afdrukt.

Empirisch de status register functies bepalen

ICIS = I²C Interrupt Statusregister

ICS = I²C Statusregister

```
@)blink      ( Twee schrijf acties naar een PCF8574 chip )
```

```
ICIS-ICS     ( 1 )
```

```
10 27      - 27 = I2C bus actief
```

```
10 27
```

```
10 6       - 6 = I2C bus in rust ( klaar )
```

```
10 6
```

```
ICIS-ICS     ( 2 )
```

```
10 27      = 27 = I2C bus actief
```

```
10 27
```

```
10 6       = 6 = I2C bus in rust ( klaar )
```

```
10 6
```

```
@)input .    ( Lees actie op PCF8574 chip )
```

```
ICIS-ICS     ( Start lees actie )
```

```
10 27      - 27 = I2C bus actief
```

```
10 27
```

```
10 E       - E = I2C bus data ontvangen
```

```
14 E
```

```
ICIS-ICS     ( Data verwerkt )
```

```
10 6       - 6 = I2C bus in rust ( klaar )
```

```
10 6
```

```
10 6 0      ( Resultaat op de stack )
```

@)1 10 ec! (Bewaar 1 op EEPROM adres 16)

ICIS-ICS (Start schrijf actie)

0 23 - 23 = I2C bus actief

0 23

10 6 - 6 = I2C bus in rust (klaar)

10 6

ICIS-ICS (Controleer of schrijven klaar is)

10 6

50 6 - 50 = Geen ACK ontvangen EEPROM nog bezig

50 6

ICIS-ICS (Controleer of schrijven klaar is)

10 27 - 27 = I2C bus actief

14 E - 0E = Laatste byte ontvangen

14 E

ICIS-ICS

10 6 - 6 = I2C bus in rust (klaar)

10 6

10 6

@)0 e@ .

ICIS-ICS

10 27 - 27 = I2C lees adres 0

10 27

10 6 - 6 = I2C bus in rust (klaar)

10 6

ICIS-ICS

10 2F - 2F = I2C bus actief & byte ontvangen

14 2F

14 2F

ICIS-ICS

10 27 - 27 = I2C bus actief, lees adres 1

10 27

10 27

ICIS-ICS

10 E - E = Tweede en laatste byte ontvangen

14 E

14 E

ICIS-ICS

10 6 - 6 = I2C bus in rust (klaar)

10 6

10 6 8 (Resultaat lees actie)

Dat levert deze test voorwaarden voor de code op:

I2C interrupt status registers:

ICIS = 00	Nog geen reactie van de I2C bus ontvangen
ICIS = 10	Er staat geen data meer klaar om te verzenden
ICIS = 14	Geadresseerde device reageert met ACK
ICIS = 50	Geadresseerde device reageert met NO-ACK of bestaat niet

I2C Status register:

ICS = 06	De I2C bus is in rust
ICS = 0E	Laatste data byte ontvangen via de I2C bus
ICS = 23	De I2C bus is actief, maar er is niets meer te verzenden
ICS = 27	De I2C bus is nog actief
ICS = 2F	Databyte ontvangen, bus nog actief

De meester I²C code

Door deze test voorwaarden toe te passen, hebben we slechts 3 van de 42 registers nodig en komen we op onderstaande code uit.

10 = IC_DATA_CMD **Data buffer and command register**
2C = IC_INTR_STAT **Interrupt status register**
70 = IC_STATUS **Status Register**

```
40044000 value 'I2C      \ I2C0_BASE      I2C register pointer
0          value SUM      \ Count of bytes to transmit or receive
: DEVICE!      ( dev -- )
    1 'i2c 6C + **bic      \ Disable I2C
    7F and 400 or 'i2c cell+ ! \ Set TARget address & start condition
    1 'i2c 6C + **bis ;      \ Enable I2C
: DATA!      ( +n -- )      \ Send data +n
    -1 +to sum sum 0= 200 and \ Decr. byte count, last add stop
    or 'i2c 10 + ! ;      \ condition to last byte & send
: {I2C-WRITE    ( +n -- )
    to sum begin 'i2c 70 + @ 6 = until ; \ Bus free? (6)
: {I2C-READ      ( +n -- )      {i2c-write ;
: I2C}          ( -- ) ; immediate \ Dummy I2C ending
: BUS!          ( b -- )
    FF and data!      \ Send data byte b
    begin 'i2c 70 + @      \ Read status register (6/27)
        6 sum if 21 + then \ Bus ready- or busy-status
    = until ;      \ Ok
: BUS@          ( -- b )
    100 data!      \ Send dummy byte
    begin 'i2c 2C + @ 50 = ?abort \ Abort on invalid read (50)
        'i2c 70 + @      \ Read status register (E/2F)
        0E sum if 21 + then \ Bus ready or busy
    = until      \ Wait until data is received
    'i2c 10 + @ FF and ;      \ Read returned data b
```

54 = IC_CLR_TX_ABRT **Clear TX abort flag by reading**

```
: {DEVICE-OK?} ( -- f )      \ Leave true when device matches
    1 {i2c-read 100 data!      \ Start dummy read data with stop condition
    begin
        'i2c 2C + @ dup 14 =      \ Device present & ready (14)?
        swap 50 =      \ Device not present or busy (50)?
    or until      \ Wait for response
    true 'i2c dup 2C + @ 40 and \ Device not present (40)?
    if >r 0= r> 44 +      \ Yes, change flag & correct address
    then 10 + @ drop ;      \ Dummy read on data or abort register
```

noForth T workshop II

0) Pico aansluiten en terminal starten op 460k8

1) noForth uitbreiden en bewaren (systeem afhankelijk)

File erbij laden: noForth-T-tools.f

En de assembler: noForth-T-asm.f

FREEZE

WORDS

2) Een tweede systeem bewaren

Laad: noForth-T-das.f

FREEZE2

WORDS

COLD

3) Wisselen tussen beide bewaarde systemen

COLD2

WORDS

COLD

WORDS

4) De configuratie bekijken en veranderen

Laad de file: print-cfg.f

Kloksnelheid omlaag, laad file: config-T-460k8-12MHz.f
.CFG

Kloksnelheid omhoog, laad file: config-T-460k8-132MHz.f
.CFG

Kloksnelheid standaard: COLD

5) De GPIO configureren

Laad de file: `blink.f`

Deze file initialiseert GPIO25 als uitgang en herdefinieert de BOOTSEL toets als gewone ingang
BOOTKEY? . 100 ms many

BLINK

Bekijk de code...

6) De ADC als temperatuur meter

Laad voorbeeld: `temperature.f`

TEMP-DEMO

Voeg aan de file uit een ijkroutine toe, om de value #CAL met een correcte waarde te vullen.

7) Hardware interrupts, sluit een snoertje aan op GPIO2 het programma start onmiddellijk.

Laad voorbeeld: `hw-interrupt-1a.f`

8) Gebruik van de ingebouwde timer

Laad de file: `Timer-examples.f`

200 alarm00 (Gebruikt armed vlag)

200 alarm-0 (Gebruikt interrupt vlag)

one ch A .ch two ch B .ch many (Interval timers)

start-alarm0 (Timer interrupt)

9) Nog vragen

noForth T workshop extras

10) Extra seriële poort maken met de PIO

Vorige code er uit: ASM\

Laad de file: PIO-assembler.f

Laad de file: PIO-disassembler.f

FREEZE

Selecteer de folder: PIO examples

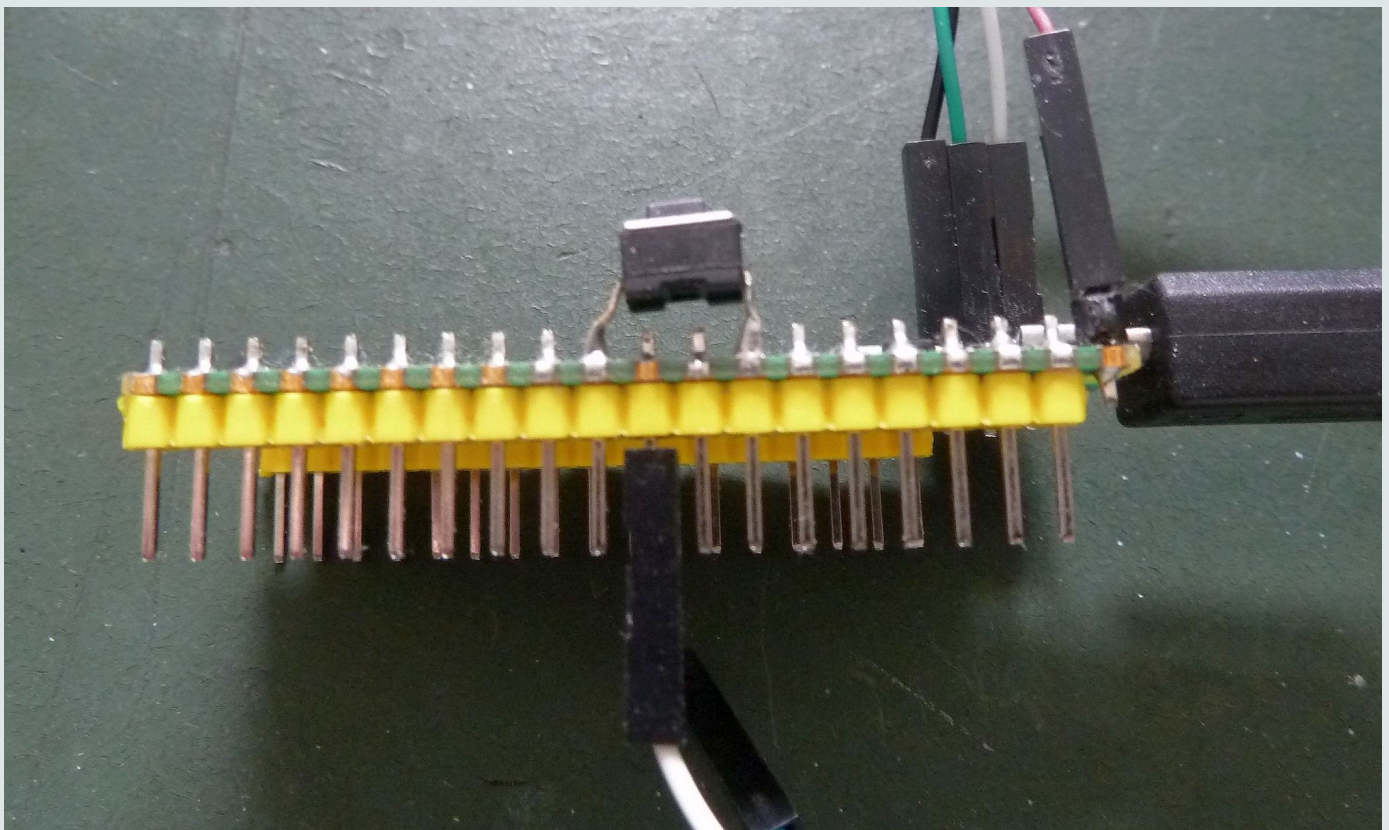
Laad de file: uart-0.f

Sluit GPIO26 en GND een tweede RS232 aan

en open een tweede terminal op 115K2

ABC stuurt de string 'ABC' naar 2^e terminal

PICO stuurt de string 'RP2040' naar 2^e terminal

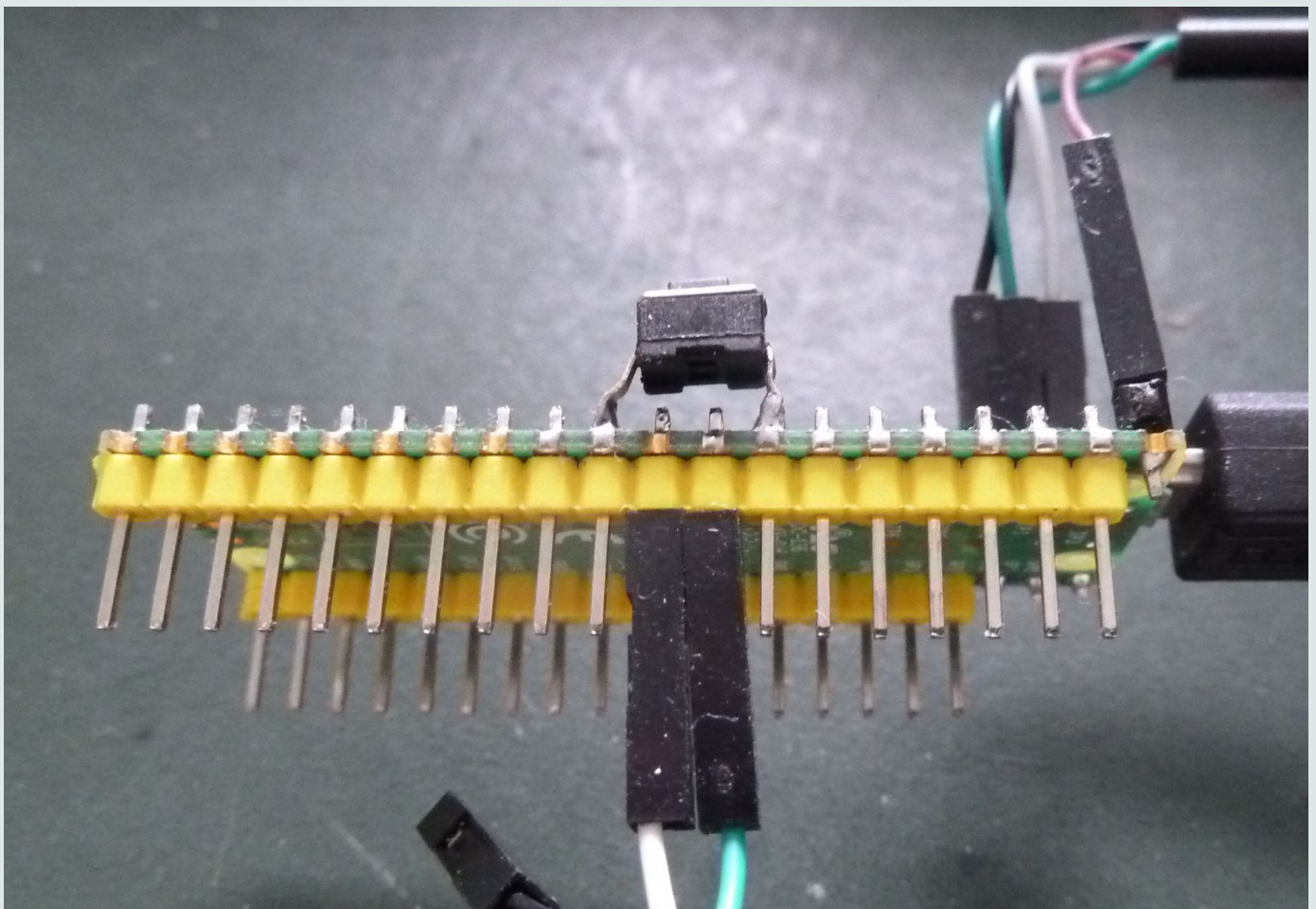


Verbinding voor UART-0 voorbeeld op GPIO26

11) Als alles lukt kun je deze nog proberen!
De noForth seriële poort omzetten naar GPIO26 & 27.
Laad de file: uart-4.f
Zet de tweede terminal op 460k8 en verbind GPIO26
en GPIO27 beiden met de tweede RS232 kabel

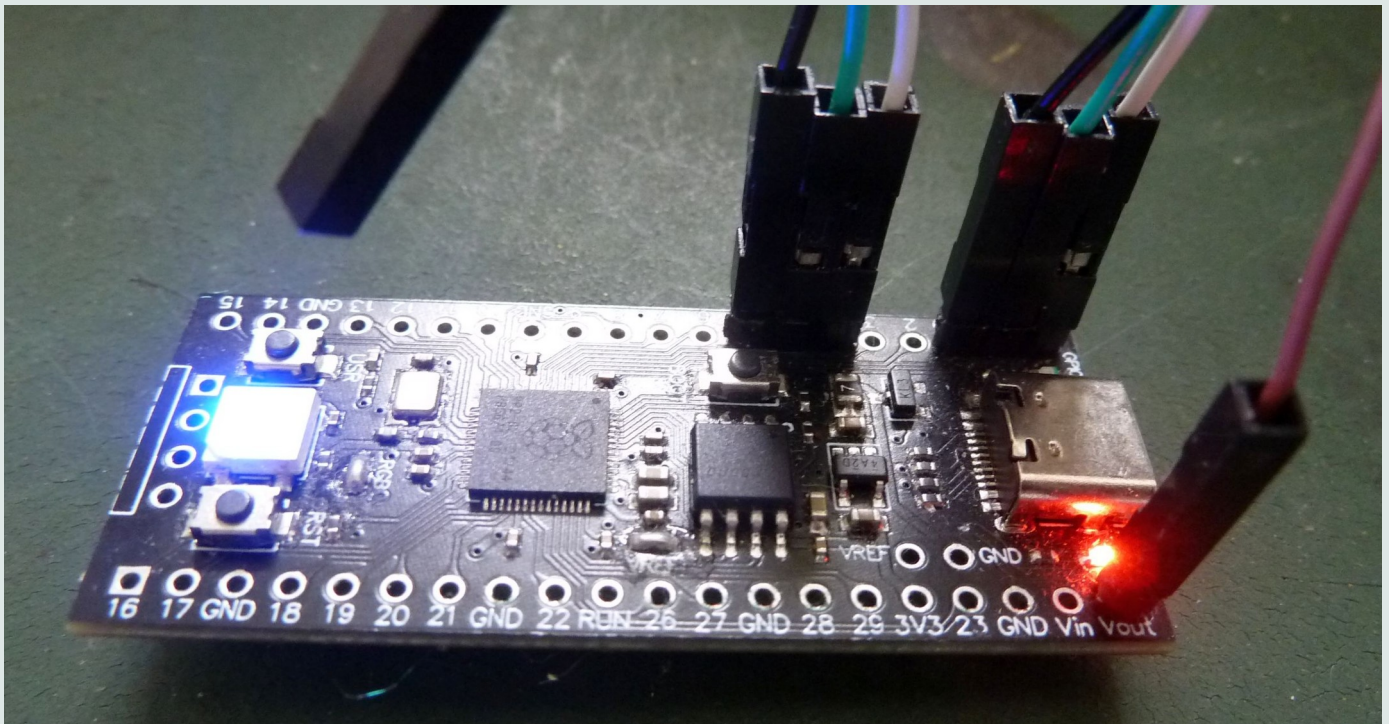
Probeer het eerst uit door PICO uit te voeren
Krijg je RP2040 in de 2^e terminal, dan werkt het.

Type nu ALT in en noForth werkt in de 2^e terminal!



Verbinding alternatieve UART op GPIO26 & 27

- 12) Gebruik van eerder gegenereerde PIO drivers.
PIO-assembler er uit: ASM\
Laad de file: mini-PIO.f
Laad een file uit de folder: PIO-ready
- 1) Flash-1.f (Simpele LED flitser)
 - 2) Flash-2.f (Bestuurbare LED flitser)
 - 3) Multiflash.f (WS2812, alleen op Chinese kloon)



WS2812 PIO driver op Chinese Pico kloon