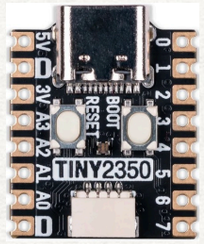
lets boot a rp2350

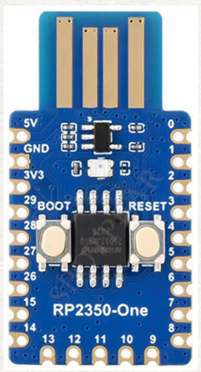
Jeroen Hoekstra

Augustus 2026

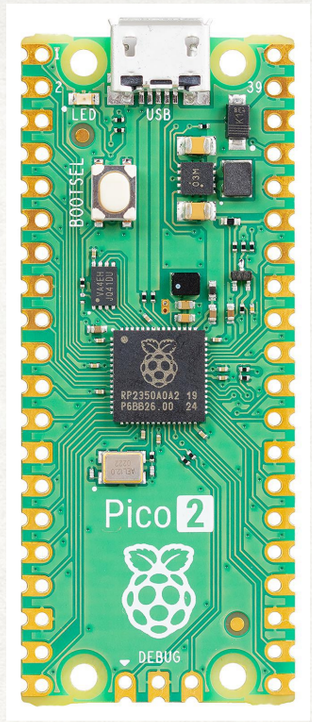
Forth gebruikers-groep HCC

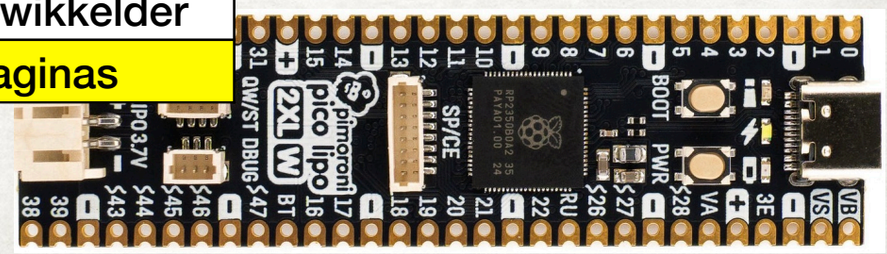
project: NoForth op de Raspberry Pico 2





	Pico 1 - rp2040	Pico 2 - rp2350
ARM		
cpu	M0+	M33
nominale f cpu	133 MHz	150 Mhz
Coremark	2.46/MHz	4.1/MHz
transistors/core	75.000	800.000
flash/ram	2MB/260 KB	4MB/520 KB
FPU	-	+
integer div	-	+
exceptions	1	7
thumb-2	-	+
Trust-zone - OTP	-	+
SHA-256 - TRNG	-	+
resus-copros	-	+
Raspberry		
veel IO - PIO	2	3
HSTX	-	+
Risc-V cores	-	+
low power	-	+
nadelen		
setup CPU	simpel	iets ingewikkelder
Boot process	22 paginas	88 paginas





Floor Plan Area cortex-M0+ en M33

Characteristics

Performance efficiency: 2.46 CoreMark/MHz * and 0.99/1.30/2.58 DMIPS/MHz **

Arm Cortex-M0+ Implementation Data ***			
	180ULL (7-track, typical 1.8v, 25°C)	90LP (7-track, typical 1.2v, 25°C)	40LP (9-track, typical 1.1v, 25°C)
Dynamic Power	47.4µW/MHz	9.37µW/MHz	3.8µW/MHz
Floor Plan Area	$0.098mm^2$	$0.028mm^2$	$0.0066mm^2$

$0.098mm^2$ **$0.028mm^2$**

Characteristics

Performance efficiency: 4.1 CoreMark/MHz * and 1.54/2.05/4.48 DMIPS/MHz **

Arm Cortex-M33 Implementation Data ***			
	40LP (9-track, typical 0.99V, -40°C)	28HPC+ (7-track, typical 0.81V, 0°C)	16FFC (7-track, typical 0.72V, 85°C)
Dynamic Power	12.0µW/MHz	3.8µW/MHz	3.9µW/MHz
Floor Plan Area	$0.028mm^2$	$0.014mm^2$	$0.008mm^2$

$0.028mm^2$ **$0.014mm^2$**

heel veel informatie beschikbaar:

documentatie Pico1 ~1000 pagina's
documentatie Pico 2 ~4000 pagina's

voorbeeld LED-blinkers (Raspberry: UF2=98.3 KB - Python: UF2=400KB)

source Zeptoforth - complete Zeptoforth ~300k UF2

metacompiler NoForth-t voor de Pico 1 met rp2040 cpu

beta-versie 16b-Thumb assembler incl UF2-generatie in WabiForth

tutorial 'RP2350 blink driver'- bare-metal
by **Kevin Thomas aka technotalent** on GitHub

strategie

UF2 Bootfile begrijpen - nieuwe specs - veel meer functionaliteit in de UF2 file dan met de Pico 1

Nieuwe BOOTROM begrijpen

WO: NoForth: consequenties van de veranderde hardware

JH: minimale LED-blinker in assembly om het boot-process onder de knie te krijgen

ervaringen - minimale LED-blinker in assembly

Thumb-assembler ongeschikt voor makro's -> nieuwe assembler geschreven - zo simpel mogelijk -> Dumb-assembler

'smart' assembler	dumb-assembler
r1, r2, r3, ldr	r1, r2, r3, ldr
r1, r2, 20 ldr	r1, r2, 20 i5ldr

WabiForth geen filesysteem -> GForth om UF2 files te krijgen

Source van de tutorial 'RP2350 blink driver'

- inhoudelijk meer dan een minimum boot
- formeel assembly format met compiler-directives, .h etc.
- 32b Thumb-2 opcodes, echter assembler=16b-Thumb

doorbraak!

doorbraak na 4 weken -> eerste blinkende led in assembly

boot process was het meest simpele mogelijk:

- haal gpios en pads uit de reset
- setup gpio 25 en isolatie van pads opheffen
- een LED loop met vertraging

correct geformatteerde UF2-file

omgezet in NoForth was de executable 122 bytes lang
(Raspberry blinkertje is ~49KB bytes)

Als je een LED aan kunt zetten dan kan je alles!

- secondaire boot naar RAM ipv FLASH

- LED patroon ipv aan/uit knipperen
- simpele exception handler

- start clocks: rosc, xosc, 2 plls, 10 clock_generators, 6 tick_generators

- SIO co-processor
- LED-loop via watchdog-reset
- voedingsspanning van de cores

  test van min en max *f* cpu op xosc:
  1.8Hz - 650MHz -> 0.23-450mA

- aanroepen routines in de bootrom

LED	betekenis
knippert 'OK'	alles ok
flasht 1x kort per 1,5sec	exception
'off'	endless loop of systeem crash
'on' maar geen patroon	fout in LED-loop
knippert 1x het patroon stopt dan	fout in watchdog
snelheid van knipperen	f CPU
patroon veranderlijk	te heet

'OK' in morse: 0000_0000_0111_0101_1100_0111_0111_0111

Niet elk voorbeeld is perfect...

```
enable_tick_generators:
        ldr r0, =TICKS_BASE | ALIAS_CLR
        ldr r1, =TICKS_CTRL_ENABLE
        str r1, [r0, #TICKS_PROC0_CTRL]
@       str r1, [r0, #TICKS_PROC1_CTRL]
        str r1, [r0, #TICKS_TIMER0_CTRL]
        str r1, [r0, #TICKS_TIMER1_CTRL]
        str r1, [r0, #TICKS_WATCHDOG_CTRL]

        ldr r0, =TICKS_BASE
        ldr r1, =TICKS_CTRL_RUNNING
1:      ldr r2, [r0, #TICKS_PROC0_CTRL]
        tst r1, r2
        bne 1b
@       ldr r2, [r0, #TICKS_PROC1_CTRL]
@       tst r1, r2
@       bne 1b
        ldr r2, [r0, #TICKS_TIMER0_CTRL]
        tst r1, r2
        bne 1b
        ldr r2, [r0, #TICKS_TIMER1_CTRL]
        tst r1, r2
        bne 1b
        ldr r2, [r0, #TICKS_WATCHDOG_CTRL]
        tst r1, r2
        bne 1b

        ldr r0, =TICKS_BASE
        ldr r1, =INIT_TICKS_CYCLES
        str r1, [r0, #TICKS_PROC0_CYCLES]
@       str r1, [r0, #TICKS_PROC1_CYCLES]
        str r1, [r0, #TICKS_TIMER0_CYCLES]
        str r1, [r0, #TICKS_TIMER1_CYCLES]
        str r1, [r0, #TICKS_WATCHDOG_CYCLES]

        ldr r0, =TICKS_BASE | ALIAS_SET
        ldr r1, =TICKS_CTRL_ENABLE
        str r1, [r0, #TICKS_PROC0_CTRL]
@       str r1, [r0, #TICKS_PROC1_CTRL]
        str r1, [r0, #TICKS_TIMER0_CTRL]
        str r1, [r0, #TICKS_TIMER1_CTRL]
        str r1, [r0, #TICKS_WATCHDOG_CTRL]

        ldr r0, =TICKS_BASE
        ldr r1, =TICKS_CTRL_RUNNING
1:      ldr r2, [r0, #TICKS_PROC0_CTRL]
        tst r1, r2
        beq 1b
@       ldr r2, [r0, #TICKS_PROC1_CTRL]
@       tst r1, r2
@       bne 1b
        ldr r2, [r0, #TICKS_TIMER0_CTRL]
        tst r1, r2
        beq 1b
        ldr r2, [r0, #TICKS_TIMER1_CTRL]
        tst r1, r2
        beq 1b
        ldr r2, [r0, #TICKS_WATCHDOG_CTRL]
        tst r1, r2
        beq 1b
```

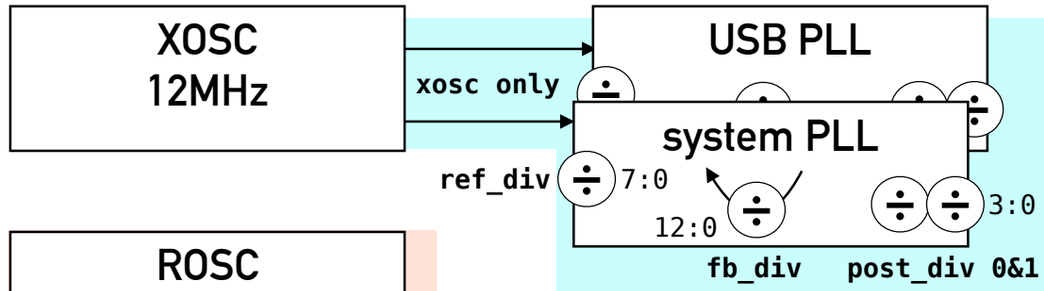
```
\ enable_tick_generators: init and start 5 ARM tick-streams -----
\ r2, counter -> 0, 12, 24, 36, 48 - if 60 -> ready

\ preamble
   r2, 0            ldv8,   \ counter
   r3, C            ldv8,   \ =12 - divisor and offset for tick_generators
   r4, 1            ldv8,   \ a number '1'

tickinitloop:
   r0, 4010B000     ldv32,
   r4, r0, r2,      str,    \ store @ r0+r2 - clear enable

   r0, 40108004     ldv32,
   r3, r0, r2,      str,    \ store '12' in tick-deler=04

   r0, 4010A000     ldv32,
   r4, r0, r2,      str,    \ store - set enable bit

   r2, r2, r3,      adds,   \ add 12 to counter
   r2, 3C           i8cmp,  \ 0x3C=60 - if 0x48 -> RISC-V counter also set up
   bne, tickinitloop:
```

minimum boot

Zeptoforth

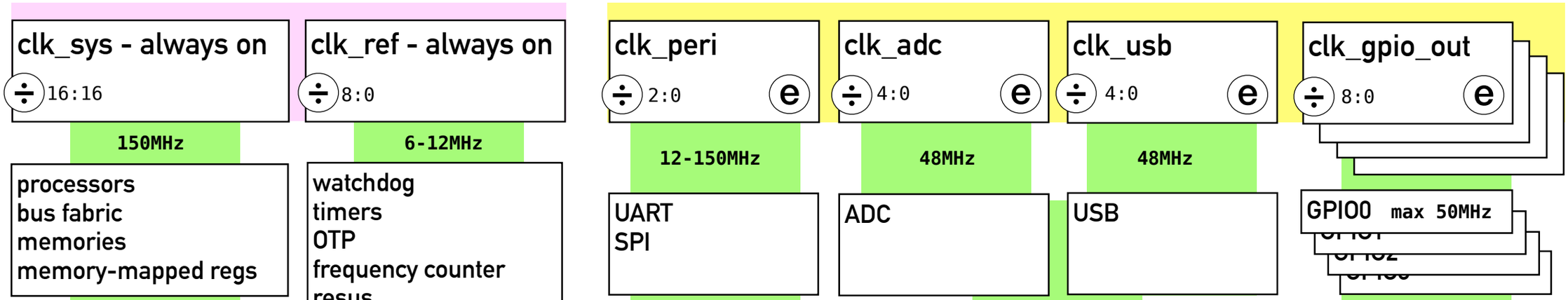
oscillators and clock generators rp2350

simplified



order of start of PLLs after boot
1. set fb_div - ref_div stays 1
2. turn on PLL and VCO power
3. wait for PLL to lock
4. set post_div 0 & 1
5. turn on power of post_divs
6. enable connected clocks

limits & constraints PLLs
ref_f in: 5-800MHz
f_outvco: 400-1600MHz
ref_div: 1-63 (normally 1)
fb_div: 16-320
post_divs: both 1-7



clk_sys auxsrc glitching
0x0 clksrc_pll_sys
0x1 clksrc_pll_usb
0x2 rosc_clksrc
0x3 xosc_clksrc
0x4 clksrc_gpin0
0x5 clksrc_gpin1

clk_sys src non-glitching
0x0 clk_ref
0x1 clksrc_clk_sys_aux

clk_ref auxsrc
0x0 clksrc_pll_usb
0x1 clksrc_gpin0
0x2 clksrc_gpin1
0x3 clksrc_pll_usb
primary_ref_opcg

clk_ref src non-glitching
0x0 rosc_clksrc_ph
0x1 clksrc_clk_ref_aux
0x2 xosc_clksrc
0x3 lposc_clksrc

clk_peri auxsrc
0x0 clk_sys
0x1 clksrc_pll_sys
0x2 clksrc_pll_usb
0x3 rosc_clksrc_ph
0x4 xosc_clksrc
0x5 clksrc_gpin0
0x6 clksrc_gpin1

clk_usb & clk_adc auxsrc
0x0 clksrc_pll_usb
0x1 clksrc_pll_sys
0x2 rosc_clksrc_ph
0x3 xosc_clksrc
0x4 clksrc_gpin0
0x5 clksrc_gpin1

clk_gpio_out auxsrc
0x0 clksrc_pll_sys
0x1 clksrc_gpin0
0x2 clksrc_gpin1
0x3 clksrc_pll_usb
0x4 clksrc_pll_usb
primary_ref_opcg
0x5 rosc_clksrc_ph
0x6 xosc_clksrc
0x7 lposc_clksrc
0x8 clk_sys
0x9 clk_usb
0xA clk_adc
0xB clk_ref
0xC clk_peri
0xD clk_hstx
0xE otp_clk2fc

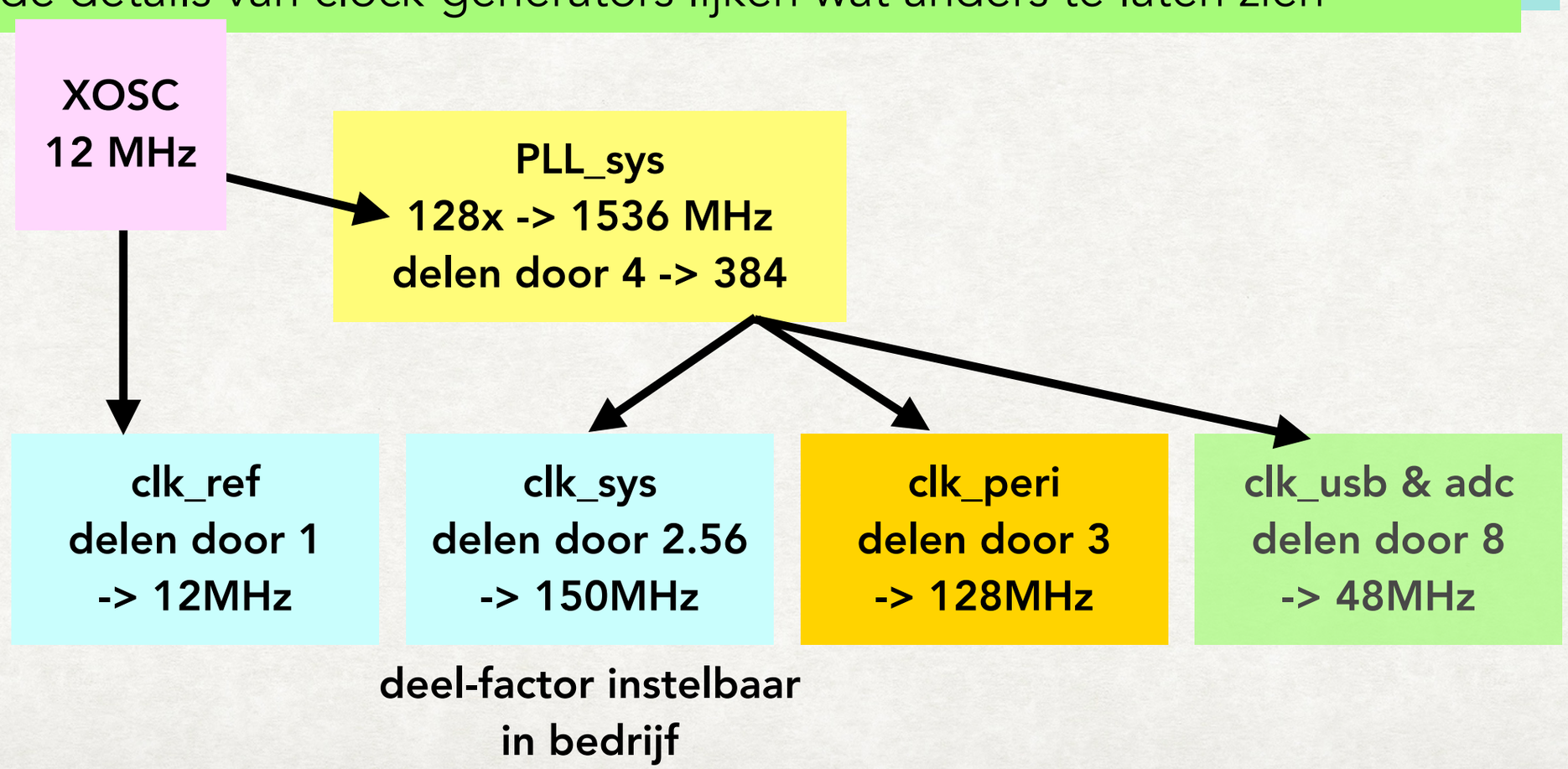
Een beter clock-systeem?

Pico 2 handboek: **1 PLL is mogelijk mits clk_sys = 48MHz**

voordelen 1 PLL

- spaart stroom
- 1 PLL vrij voor de programmeur
- een goed compact clock-systeem dat 'eigen' is voelt beter

de details van clock-generators lijken wat anders te laten zien



aan de slag met clock_system v2

clock_system v1 bleek erg fragile

De toevoeging van een neutrale opcode resulteerde regelmatig in een exception

hoofd-oorzaak - fout in het zetten van een deel-factor:

stap 1: bits van deel-factor naar '0' met clear_bits

stap 2: deler naar gewenste waarde met set_bits

Probleem: een deel-factor van '0' geeft een ongedefinieerde toestand

prototype klaar: ~250 naar ~100 opcodes - nu testen met scoop.

Hoe klein kan een LED-blinker zijn?

Universal blinker: 1000 KB .UF2 -> 500 KB image
Python versie: 400 KB .UF2 -> 200 KB image
De Raspberry blinker: 98.2 KB .UF2 -> 49.1 KB image

De eerste versie in NoForth was 122 bytes

Mijn eerste poging: 104 bytes
Appje van Willem: 88 bytes
Ik zag in die code iets wat evt. weg kon: ~80 bytes
Willem liet meer weg: 68 bytes
Ik zag nog 2 overbodige bytes: 66 bytes
Willen vond nog 2 bytes door LDM te introduceren: 64 bytes

64 byte LED blinker in noForth

```
:::NOFORTH:::
trace

    40038068 ,          \ 00 HOP - GPIO25 control address
    -1 ,                \ 04 DAY - Reset vector
    400280CC ,          \ 08 SUN - IO_BANK0 GPIO25

image-def

\ ip  sp  w   tos hop day sun moon
\ R0  R1  R2  R3  R4  R5  R6  R7
\ Setting up a simple LED flasher for CORE-0
\ Using only CPU registers and one I/O-pin
\ More info on SIO_BASE from address 36/54 ff.
  chere 10000000 - ivecs cell+ vec! \ Install BOOT at second location in IVECS
\ Build data pointer
    w 1 # movs,                     \ Set 1 in W
    tos w 1C # lsls.mv,             \ Set data pointer in TOS = 1000,0000
    tos  { ip hop day } ldm,        \ Load pointers in one go
\ Configureer GPIO25 voor SIO (Functie 5)
    sp 5 # movs,                    \ enable SIO
    sp  sun ) str,                  \ IO_BANK0 GPIO25, IP = 400280CC
\ PADS isolation off
    sp 5A # movs,
    sp  hop ) str,                  \ GPIO25 control address, IP = 40038068
\ Activate output, W = already 1
    w 19 # lsls,                    \ Set bit 25, W = 200,0000
    ip 0D # movs,                   \ Build SIO_BASE, IP = D000,0000
    ip 1C # lsls,
    w  ip 30 #) str,                \ GPIO_OE
\ Blink loop
    begin,
\ Led toggle
        w  ip 28 #) str,            \ GPIO_OUT_XOR
\ inline delay
        sp ip 0B # lsrs.mv,         \ Scale delay to ~ 1,700,000
        begin,  sp 1 # subs, =? until,
    again,

;;;NOFORTH;;;
```

```
hex
\ -- constants and start address -----
    40038068 word,              \  0=0x00 - r0, - misused as gpio25 control address
    10000021 word,              \  4=0x04 - 0x21=33 vector to start address + 0x1 - Thumb bit set
    400280CC word,              \  8=0x08 - r2, io_bank_base + CC for gpio25 C

    ffffded3 word,              \ 12=0x0C - magic word
    10210142 word,              \ 16=0x10 - family
    000001ff word,              \ 20=0x14 - << de 1 duidt op 1 PICO.BIN block
    00000000 word,              \ 24=0x18 - 0x0 = loop back to first block
    ab123579 word,              \ 28=0x1C - magic word
                                \ 32=0x20 - 1 ->0x21

\ -- pseudo literals -----
    r7, 1           i8movs,     \ KEEP
    r0, r7, 1C      i5lsls,     \ r0=base address of pseudo-literals - KEEP
    r0, {, r0, r1, r2, }, ldm,  \ r1,=useless value - C807 16b,

\ -- setup of the alt-function for gpio25 -----
    r1, 5           i8movs,     \ alt-func =x5 = GPIO
    r1, r2, 0       i5str,      \ 400280CC - store new alt-function of gpio25

\ -- pad isolation gpio25 off -----
    r1, 5A          i8movs,
    r1, r0, 0       i5str,

\ -- set GPIO25 as output   -----
    r5, r7, 19      i5lsls,     \ 0x19=0d25 -> 0x2000000 in r5 - KEEP
    r4, D           i8movs,
    r0, r4, 1C      i5lsls,     \ make address 0xD0000000 0x1C=0d28
    r5, r0, 30      i5str,      \ set bit25 in 0xD0000030 to enable output

\ --  blink LED using xor -----
LEDloop:
    r4, r5, 6       i5lsrs,     \ wait-time derived from r5=bit25

LEDloop2:
    r4, r4, 1       i3subs,
    bne,            LEDloop2:
```

64 byte LED blinker in Wabiforth

64 byte LED blinker in disasembly

```
0X10000000  40038068
0X10000004  10000021
0X10000008  400280CC
0X1000000C  FFFFDED3
0X10000010  10210142
0X10000014  000001FF
0X10000018  00000000
0X1000001C  AB123579

0x10000020  01 27   movs  r7, #1            \ you always need a '0x1'
0x10000022  38 07   lsls  r0, r7, #0x1c     \ shift with 0d28 to get XIP base address
0x10000024  07 C8   ldm   r0, {r0, r1, r2}  \ load first three values in r0 to r2, r1=dummy

0x10000026  05 21   movs  r1, #5
0x10000028  11 60   str   r1, [r2]          \ set alt-function of GPIO25 to output

0x1000002a  5A 21   movs  r1, #0x5a
0x1000002c  01 60   str   r1, [r0]          \ switch off pad-isolation for gpio25

0x1000002e  7D 06   lsls  r5, r7, #0x19
0x10000030  0D 24   movs  r4, #0xd
0x10000032  20 07   lsls  r0, r4, #0x1c     \ r0=0xD0000000
0x10000034  05 63   str   r5, [r0, #0x30]   \ activate output for gpio25

0x10000036  AC 09   lsrs  r4, r5, #6        \ r4=start value of wait loop
0x10000038  64 1E   subs  r4, r4, #1        \ sub 1 from r4
0x1000003a  FD D1   bne   #0x20000038       \ on not-zero branch back

0x1000003c  85 62   str   r5, [r0, #0x28]   \ xor the output of the LED on gpio25
0x1000003e  FA E7   b#    0x20000036        \ branch back
```

conclusie

Is het booten van een rp2350 kei-moeilijk?

Wel nee, het is stik-makkelijk!

the end

Een beter clock-systeem?

Pico 2 handboek: **1 PLL is mogelijk mits clk_sys = 48MHz**
Maar: de details van clock-generators lijken wat anders te laten zien

1 PLL spaart stroom
geeft de programmeur flexibiliteit
een goed compact clock-systeem dat 'eigen' voelt beter

Ontwerp clock_system v2:

- pll_sys: feedback deler: 128 -> 12*128=1536 MHz
  output-deler=4 -> pll_sys out 384 MHz

- clk_sys: pll_sys input-deler=2,56 -> 150 MHz (6kHz-384MHz)
- clk_usb en adc: pll_sys out input_deler=8 -> 48 MHz
- clk_peri: pll_sys input_deler=3 -> 128 Mhz

- clk_ref: xosc 12 MHz direct

heel veel documentatie beschikbaar:

handboek pico2 - 1380 pag (pico 1: 642)

handboeken ARM-M33
- ARMv8-M handboek - 2149 pag (M0+: 374)
- ARM-M33 tech ref - 137 pag
- ARM-M33 user-guide - 316 pag

voorbeeld blinkers (Raspberry: UF2=98.3 KB - Python versie UF2=400KB)
source Zeptoforth - complete Zeptoforth ~300k UF2

tutorial 'RP2350 blink driver'- bare-metal
by **Kevin Thomas aka technotalent** on GitHub

metacompiler NoForth-t voor de Pico 1 met rp2040 cpu

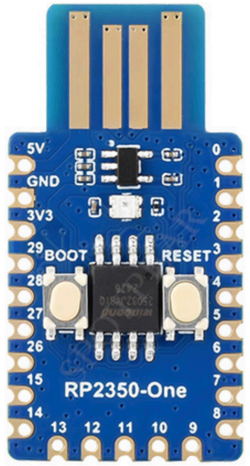
beta-versie 16b-Thumb assembler incl UF2-generatie in WabiForth

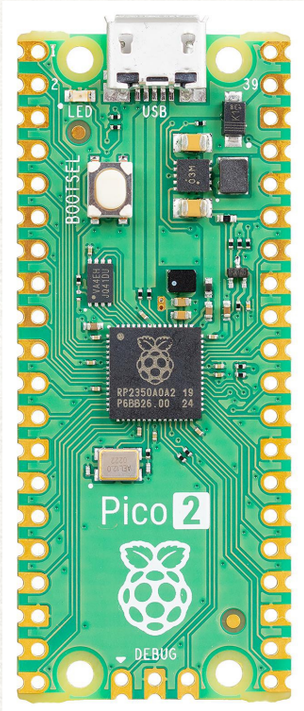
project: NoForth op de Raspberry Pico 2

ARM-M33 processor @ 150 MHz - 4.1 vs 2.46 Coremark/MHz
DSP, FPU, division, Thumb-2, SIO-coprocessor, Trust-zone, exception-handling, SHA-256, TRNG, resuscitate-coprocessor, Risc-V cores, 520K ram

M33: ~800.000 trans./core - M0+: ~75.000 trans./core
(ARM2: ~30.000 - 1986) 2354

3x PIO
low-power mogelijkheden zijn beter
HSTX voor grafisch output
verder ongeveer als de Pico 1





Nadelen
De setup van de CPU is wat complexer
het bootproces is VEEL complexer dan de Pico 1 (88 v 22 pagina's)