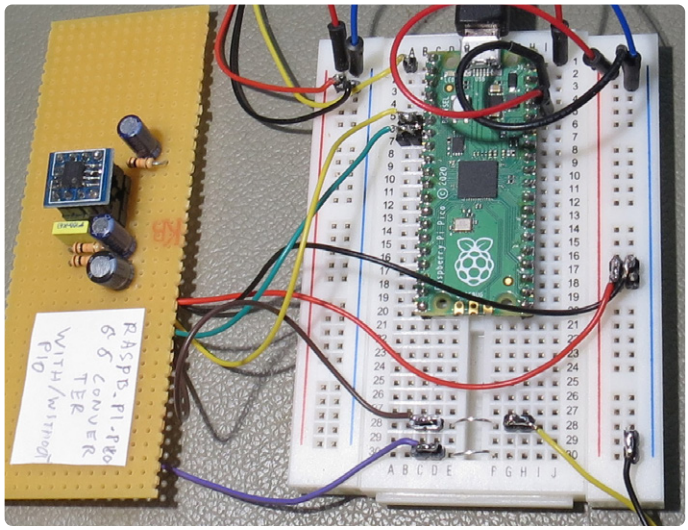
PIO in de praktijk

experimenten met de programmeerbare I/O van de RP2040

Prof. dr. Martin Ossmann (Duitsland)

De Raspberry Pi Pico is een populair ontwikkelboard dat veel rekenkracht biedt bij een gering stroomverbruik. En met een prijs van net € 4 per stuk krijg je veel waar voor je geld. De RP2040-microcontroller op het board heeft een (tot nu toe) unieke eigenschap: twee PIO I/O-blokken met acht toestandsmachines. Aan de hand van enkele praktijkvoorbeelden bekijken we welke voordelen dit voor een toepassing biedt.



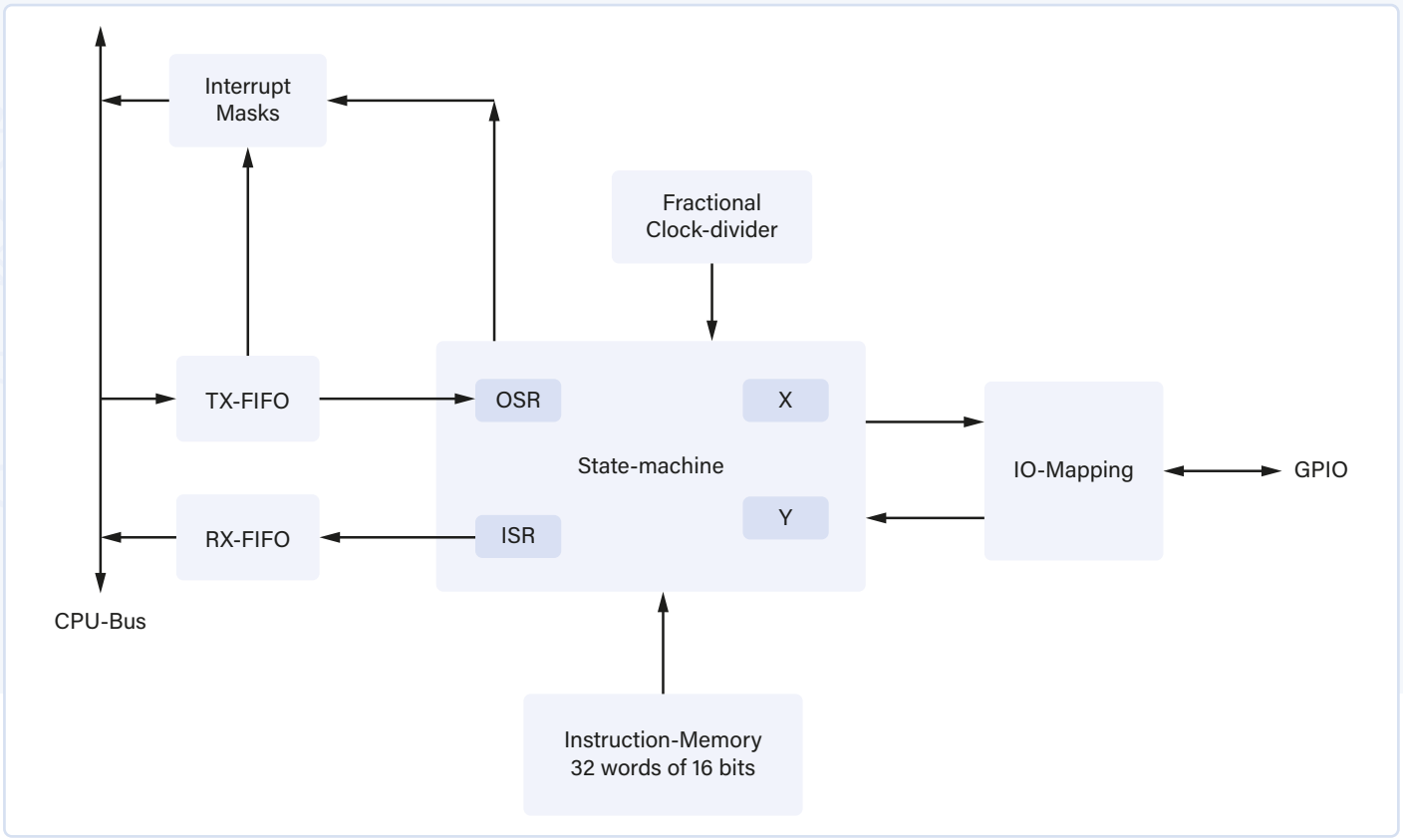
Figuur 1. Het Raspberry Pi Pico-board op een breadboard, en daarnaast de delta/sigma-print.

Als je al wat ervaring hebt met het programmeren van microcontrollers, zul je ongetwijfeld een ingebouwde programmeerbare I/O-unit op prijs stellen om de belasting van de hoofd-processorcores te reduceren. De grote vraag is of deze unit niet onnodig ingewikkeld in het gebruik is en daarom niet de moeite waard. Je zult zeker een beter beeld krijgen van de mogelijkheden als je de hier beschreven praktische voorbeelden op de in de RP2040-processor ingebouwde PIO's hebt doorlopen. **Figuur 1** toont mijn experimentele hardware-opstelling waarop ik de besproken voorbeelden heb ontwikkeld.

De PIO-unit

De flexibiliteit van de programmeerbare input/output-unit maakt het mogelijk de I/O-pinnen te gebruiken om andere soorten communicatieprotocollen te implementeren naast de bestaande standaarden zoals SPI en I²C. De unit bestaat uit twee PIO-blokken, elk met vier I/O-processoren. Elke afzonderlijke I/O-processor is geïntegreerd in de chiparchitectuur zoals geschetst in **figuur 2**.

In totaal zitten er twee PIO's in de RP2040; elk bevat vier toestandsmachines (state machines, SM's) en een instructiegeheugen van 32 woorden om de toestandsmachines aan te sturen. Elke SM heeft twee 32bit-registers X en Y en een 32-bit ingangsschuifregister (ISR) en een 32-bit uitgangsschuifregister (OSR). De PIO-kernen zijn elk verbonden met de RP2040-CPU via twee 32 bit brede FIFO's met een diepte van vier woorden. De TX FIFO-unit wordt gebruikt voor het verzenden en de RX FIFO-unit voor het ontvangen van data.

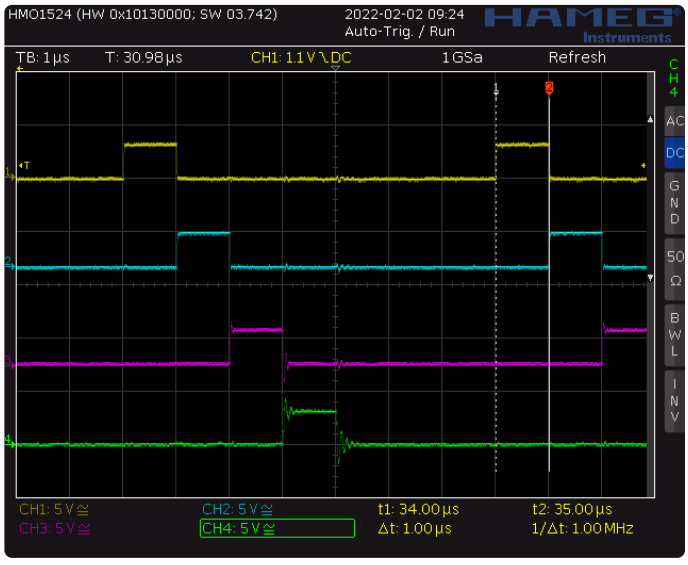


Figuur 2. Blokschema van een van de twee I/O-processoren (bron: [4]).

Als gegevens slechts in één richting gaan, kunnen de twee FIFO's worden geconfigureerd om als een enkele acht woorden diepe FIFO te werken. Elke PIO-processor heeft een 16.8-bit fractionele deler voor het genereren van het kloksignaal (zie verderop). Dit geeft de flexibiliteit om de kloksnelheid van elke processor in te stellen op een waarde tussen 2 kHz en 125 MHz.

De PIO-processoren verstaan slechts negen instructies. Deze zijn 16 bit breed en worden opgeslagen in een 32×16 bit instructiegeheugen dat de vier SM's in elk PIO-blok aanstuurt. De afzonderlijke PIO-programma's zijn kort genoeg om in dit kleine geheugen te passen. De instructies worden hier door de RP2040 CPU opgeslagen. Deze kleine programma's worden geschreven met behulp van een speciaal maar vrij eenvoudig PIO-assemblerprogramma. Elke instructie heeft voor de uitvoering één klokcyclus nodig.

Een enkele PIO-processor kan slechts een paar GPIO-pinnen aansturen, en gebruikt daarvoor korte adressen. De IO-mapping bepaalt welke pinnen dat specifiek zijn. Er zijn verschillende programmeerbare constanten die voor het betreffende gebied (input, output, side set en set) bepalen wat de eerst geadresseerde pin is. Hierdoor kunnen de IO-pinnen van een PIO-processor zeer flexibel worden gedefinieerd. Ook kunnen meerdere processoren toegang krijgen tot dezelfde pin.



Figuur 3. De golfvormen van het eerste voorbeeld.

Wat heb je nodig

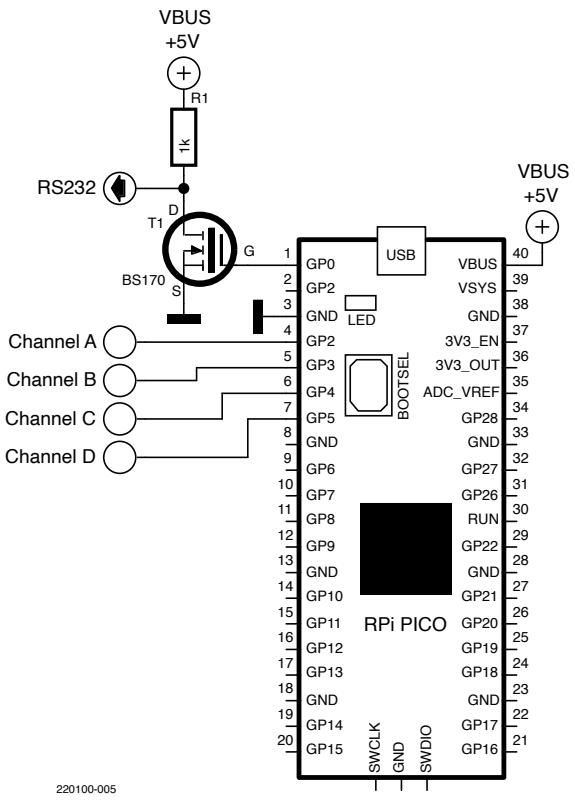
Hier wordt VS Code gebruikt als ontwikkelomgeving. De installatie en het gebruik van deze tool zijn goed beschreven in [1]. Daar vind je ook informatie over het gebruik van de projectgenerator. Hiermee wordt een Raspberry Pi-project gemaakt met alle benodigde bestanden (broncode, configuratiebestanden, make-setup enzovoort).

Je moet ook aangeven of de uitvoer van de `printf()` functie via USB of UART loopt. Hier configureer je ook welke bibliotheken je moet gebruiken. In dit artikel gebruiken we bijvoorbeeld de PIO-tool. Hulp bij de installatie van VS Code kun je vinden in [2]. Er is ook een gedetailleerde beschrijving van welke extra tools (Make, Git enzovoort) geïnstalleerd moeten worden.

Alle softwarevoorbeelden kunnen worden gedownload van [3].

Aan de slag

We willen dat dit eerste programma een puls van 1 µs via vier uitgangen na elkaar uitvoert, zoals te zien in **figuur 3**. We hebben een vierkanaals



Figuur 4. Aansluiting van een vierkanaals 'scoop op de Raspberry Pi Pico om de golfvormen van het eerste voorbeeld te bekijken.

oscilloscoop aangesloten op de vier uitgangen om de resulterende uitgangssignalen te zien. **Figuur 4** toont de aansluiting van de oscilloscoop op de I/O-pinnen van de Raspberry Pi Pico.
Listing 1 bevat het eerste voorbeeld-PIO-programma. Na drie regels assembler directives staat de eerste uitvoerbare instructie op regel 4. Deze en volgende regels genereren vervolgens de vier pulsen.

Listing 1

```
001 .program pioTest
002 .side_set 2 opt
003 .wrap_target
004   nop           side 0b01 // GP2=1, GP3=0
005   nop           side 0b10 // GP2=0, GP3=1
006   set pins,0b01 side 0b00 // GP2=0, GP3=0,
        GP4=1, GP5=0
007   set pins,0b10       //    GP4=0,      GP5=1
008   set pins,0b00  [2]  //    GP4=0,      GP5=0
009 .wrap
```

Regels 4 en 5 bevatten de `nop`-instructies, die zoals je waarschijnlijk al vermoedt niets doen, maar regel 4 wordt uitgebreid met de tekst `side 0b01`. Dit geeft aan dat op deze regel iets anders wordt 'terzijde' wordt gedaan. In dit geval wordt side pin 0 '1' gemaakt en side pin 1 '0' gemaakt. Elke PIO-instructie kan op deze manier worden uitgebreid. Zo kun je gemakkelijk veel meer pinnen tegelijk aansturen. Het is belangrijk om aan te geven welke de side-set pinnen zijn, met behulp van configuratie-opdrachten geschreven in C. De definitie is in C gedaan in **listing 2** met behulp van de functie `sm_config_set_sideset_pins()` in regel 5. Deze geeft aan dat de side-set pinnnen beginnen bij GP2.

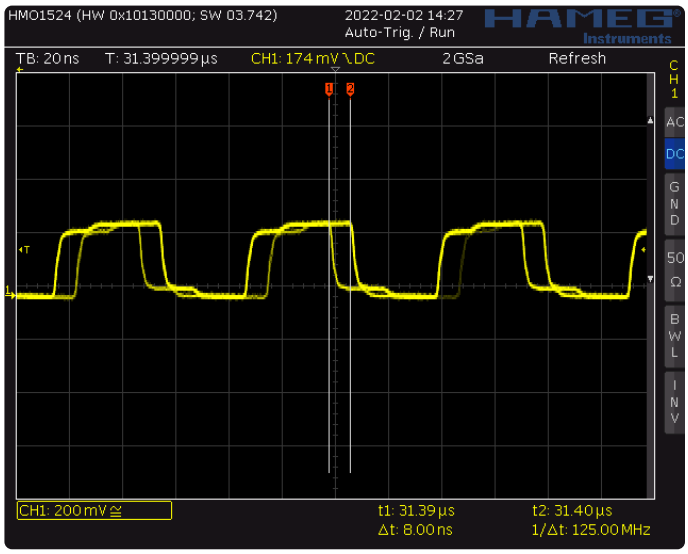
Listing 2

```
001 #define set_base 4   // pins start at GP4
002 #define set_count 2  // number of pins is 2
003 sm_config_set_set_pins (&c,set_base,set_count);
004
005 #define side_set_base 2 // side-pins start at GP2
006 sm_config_set_sideset_pins(&c, side_set_base);
007
008 float div = 125.0;
009 sm_config_set_clkdiv(&c, div);
```

De `nop`-instructie in regel 4 van listing 1 genereert een opgaande flank via side-set op side pin 1 (= GP3). Regel 6 bevat een set-instructie die de pinnen 0 en 1 van de set-pins zet. De puls bereikt dan pin GP4. Side-pinnen 0 (GP2) en 1 (GP3) worden via side-set gereset. In de configuratieroutine in de tweede listing in regel 3 wordt bepaald dat de set-pinnen starten bij pin nummer GP4. De `set`-instructie kan worden gebruikt om registers of IO-pinnen te laden met constante waarden. De modifier `[2]` na de `set`-instructie op regel 8 van de eerste listing zorgt ervoor dat de instructie wordt gevolgd door een vertraging van twee klokcycli. Met deze specificatie tussen rechte haken kan elke instructie met 1 tot 31 klokcycli worden vertraagd.
Nu moet je alleen nog een geschikte klok gebruiken om ervoor te zorgen dat elke puls precies 1 µs duurt. Daartoe moet je de systeemklok van 125 MHz delen door 125. Met de resulterende klok van 1 MHz duurt elke cyclus precies 1 µs. Dit wordt gedefinieerd in de configuratieroutine van de tweede listing hierboven in regels 8 en 9.
De assembler-directives `.wrap_target` en `wrap` in listing 1 zorgen ervoor dat de instructies ertussen in runtime in een eindeloze lus worden herhaald. De sprong terug naar het begin van de lus (`.wrap_target`) kost geen tijd. Je kunt 'zero-cycle-overhead loops' programmeren. In ons programma duurt het zeven klokcycli (vijf instructies + twee cycli vertraging) om de lus eenmaal te doorlopen.

Genereren van de klok: fractionele deler

Zoals al opgemerkt, wordt de klok van de PIO-processor afgeleid van de 125MHz-systeemklok met behulp van een 'fractionele deler.' De 16bit-deler kan tot 2^{16} = 65.536 tellen en heeft acht binaire cijfers na de dcimale punt. De nauwkeurigheid is dus 1/256. De 'fractionele deler' wordt op dezelfde manier gebruikt als in DDS-signaalgeneratoren. De fractie na de decimale punt wordt continu opgeteld en bij elke carry wordt een klokflank gegenereerd. **Listing 3** demonstreert het instellen van een fractionele kloksnelheid.



Figuur 5. Het gegenereerde kloksignaal vertoont jitter.

Listing 3

```
001 float clockFrequency=28e6;
002 float div=125e6/clockFrequency;
003 sm_config_set_clkdiv(&c,div);
```

Door de manier waarop de klok wordt gegenereerd, fluctueert de klokperiode rond een systeemklok-periode en vertoont de klokgolfvorm enige jitter. In het voorbeeld willen we een kloksnelheid van 28 MHz, maar aangezien 28 geen heeltallige deler van 125 is, is ook de deelfactor geen geheel getal, maar heeft die een rest. De klok-deelfactor is 125/28 = 4,4642857...
De derde listing laat zien hoe deze deler wordt gebruikt voor de configuratie. Als demo wordt een eenvoudige blokgolf gegenereerd met het PIO-programma in **listing 4** hieronder.

Listing 4

```
001 .program theProgram
002 .side_set 1 opt
003     nop    side 0
004     nop    side 1
```

De oscilloscoop toont in **figuur 5** de klokjitter heel duidelijk. Bij hogere waarden van de klokdeler wordt de jitter proportioneel kleiner en legt minder gewicht in de schaal, zodat hij vaak kan worden verwaarloosd. Met de fractionele deler kunnen dan baudrates zoals de typische 115.200 bit/s met voldoende nauwkeurigheid worden gegenereerd.

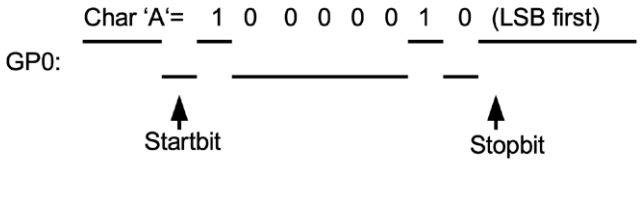
Seriële gegevensoverdracht: een TX-UART

Het volgende voorbeeld is wat praktischer: het gaat om het serieel versturen van gegevens via een RS232-interface. De interface is aangesloten op pin GP4 zodat GP0 van de ingebouwde UART vrij blijft voor debug-doeleinden. Aangezien de interface inverterend is, moet je een geïnverteerd signaal genereren (idle state = hoog). Wanneer het ASCII-karakter wordt verzonden, ziet de pulsvorm op pin GP4 er dan zo uit als in **figuur 6**.
Naast het TXD-signaal moet de UART een 1 uitvoeren op een busy-lijn tijdens het verzenden van de acht databits. GP5 (side-pin 0) moet als busy-lijn wordne gebruikt. Het bijbehorende PIO-programma staat in **listing 5** hieronder.

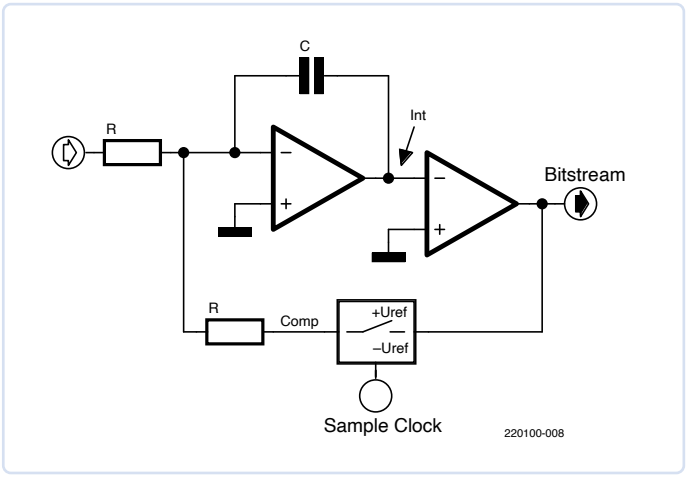
Listing 5

```
001 .program theProgram
002 .side_set 1 opt
003     pull              [1]
004     set pins,0 side 1 [2] // startBit=0
005     set x,7               // loop for 7+1 times
006 bit:
007     out pins,1        [2] // LSB first
008     jmp x-- bit
009     set pins,1 side 0 [2] // stopBit=1
```

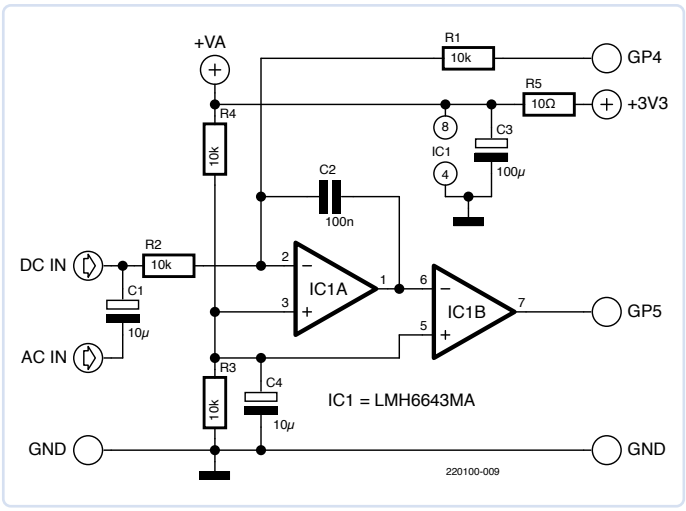
De `pull`-instructie in regel 3 haalt een 32bit-woord uit de TX FIFO op en slaat dat op in de OSR. Als er momenteel geen woord in de TX FIFO staat, wacht de instructie gewoon op een woord en voegt het dan rechtstreeks in de OSR in. De uitgangspen wordt door de set-instructie in regel 4 op 0 gezet om als startbit te fungeren.
De acht minst significante bits van het woord in de OSR worden nu in een lus naar buiten gevoerd. Hiervoor is het nodig de lus-teller te initialiseren. Het X-register dient hier als lus-teller. De `set`-instructie in regel 5 initialiseert deze met de waarde 7.
Het label `bit` in regel 6 markeert het begin van de lus. De eerste en enige instructie in de lus is `out` in regel 7. Deze voert het LSB van de OSR uit op out-pin 0 en verschuift de bits in de OSR één positie naar rechts. Naast de set-pinnen en de side set-pinnen vormen de out-pinnen de derde methode om de GPIO pins te gebruiken. Pin GP4 is ook ingesteld als out-pin.



Figuur 6. Pulsvolgorde om het teken 'A' te verzenden.



Figuur 7. Principeschema van een delta/sigma-ADC.



Figuur 8. Praktische uitvoering van de delta/sigma-ADC van figuur 7.

De voorwaardelijke sprong in regel 8 met de optie `x--` verlaagt het X-register met 1 en springt dan naar het label `bit` (het begin van de lus) wanneer `x` groter is dan 0. Op deze manier wordt de lus acht keer doorlopen. De laatste instructie van de TX-zendroutine is `set` in regel 9, die het stopbit maakt door set-pin 0 op 1 te zetten.
Bij de twee side-extensions wordt het Busy-signaal gegenereerd door side-pin 0 (GP5).
De vertragings-extensions tussen rechte haken zorgen ervoor dat elk bit precies vier klokcycli lang is. De baudsnelheid is dus gelijk aan de kloksnelheid gedeeld door vier. Het instellen van `clkDiv` op 125.000.000/ (4 * 115.200) resulteert dus in de gewenste baudrate van 115.200 bit/s.
De functie `pio_sm_put_blocking(...)` wordt gebruikt om karakters met een C-programma naar de PIO-UART te sturen, zoals de `PIOprint()` functie in **listing 6** demonstreert. Hier wordt een karakter doorgegeven aan de TX FIFO-unit. Als die vol is, blokkeert hij en wacht op meer ruimte.

Listing 6

```
001 void PIOprint(const char *s) {
002    while (*s)
003       pio_sm_put_blocking(pio, sm, *s++);
004    }
```

De TX UART-routine laat goed zien dat je met slechts zes instructies zinvolle functionaliteit kunt bereiken, waarbij de toevoeging van side set- en delay-instructies een belangrijke rol spelen.

Een delta/sigma-ADC

In deze paragraaf wordt een delta/sigma A/D-omzetter geïmplementeerd met behulp van een PIO. Het principe van een dergelijke omzetter is te zien in **figuur 7**. De schakeling probeert de ingangsspanning te compenseren met een schakelbare spanningsbron. Een integrator aan de ingang integreert het foutsignaal en een volgende comparator bepaalt het teken van de compensatiespanning gedurende het volgende tijdsinterval. Aan de uitgang ontstaat een bitreeks die het gemiddelde van de ingangsspanning volgt. **Figuur 8** toont een praktische schakeling volgens dit principe.
Deze schakeling wordt aangesloten op GP4 en GP5 van de Raspberry Pi Pico. De bitvolgorde wordt bepaald via PIO; acht bits worden samen als één byte opgeslagen in de RX FIFO-unit. De microcontroller haalt vervolgens de bytes uit deze unit op en verwerkt ze verder. Het bijbehorende PIO-programma is te zien in **listing 7**.

Listing 7

```
001 .program hello
002 Loop1:
003    set   x,7
004 inLoop:
005    in    pins,1
006    mov   osr,~isr
007    out   pins, 1
008    jmp   x-- go1   [20]
009    push  noblock
010    jmp   Loop1     [23]
011 go1:
012    jmp   inLoop    [26]
```

De buitenste lus met het label `Loop1` wordt eindeloos herhaald. De binnenste lus maakt altijd acht iteraties. Elke iteratie begint met de instructie na het label `inLoop`. De `in`-instructie in regel 5 leest één bit van de ingangspin en schrijft die naar de ISR. In regel 6 wordt de ISR geïnverteerd gekopieerd naar de OSR. In regel 7 wordt het minst significante bit uitgevoerd naar de uitgangspin, waardoor de volgende compensatiestap in de integrator wordt gegenereerd.

Het gelezen bit is het volgende uitgangsbit van de deltasigma-omzetter. Acht bits worden verzameld in de ISR. De `jmp`-instructie in regel 8 bestuurt de binnenste lus. Wanneer deze 8 maal is doorlopen, wordt de `push`-instructie van regel 9 uitgevoerd. Deze instructie slaat de ISR op in de RX FIFO-unit die de MCU vervolgens kan legen. De delay-extensions tussen rechte haken zorgen ervoor dat elke delta/sigma-stap 50 klokcycli duurt. De klokdeler is ingesteld op 25. De bemonsteringsfrequentie van de delta/sigma-omzetter is dus 125 MHz / (25*50) = 100 kHz.
Als demo hebben we een sinusgolf van 50 Hz omgezet met behulp van de delta/sigma-converter. **Figuur 9** toont het resulterende spectrum. De afstand tussen het bruikbare signaal en de ruis is ongeveer 60 dB. Helemaal niet slecht voor zo'n simpele schakeling!

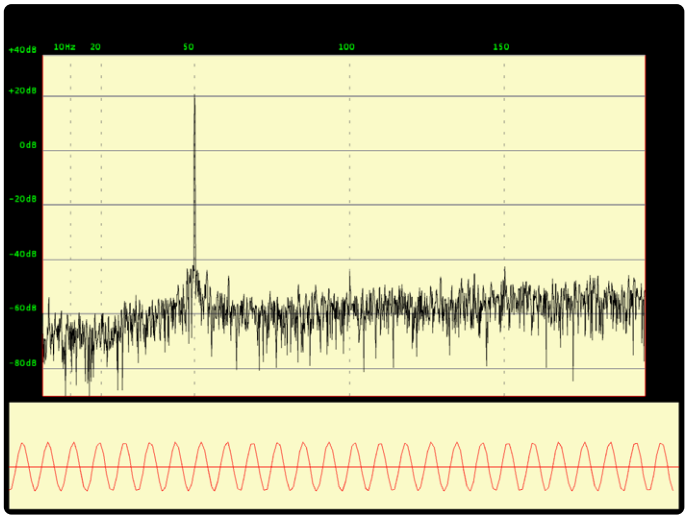
Bouw een signaalgenerator

In dit voorbeeld bouwen we een eenvoudige R2R-DAC met behulp van een laddernetwerk volgens **figuur 10**. Dit wordt aangesloten op de I/O-pinnen van de Raspberry Pi Pico om analoge uitgangssignalen te produceren. De generator moet een continu en periodiek uitgangssignaal genereren. Hiertoe moeten gegevens herhaaldelijk vanuit een tabel in het geheugen naar de D/A-omzetter worden getransporteerd. Dit werkt bijzonder goed met behulp van DMA (*Direct Memory Access*). De DMA-controller van de Raspberry Pi Pico brengt de gegevens over naar de TX FIFO-unit. Het PIO-programma bestaat uit een enkele uitvoerbare instructie, zoals te zien in **listing 8**.

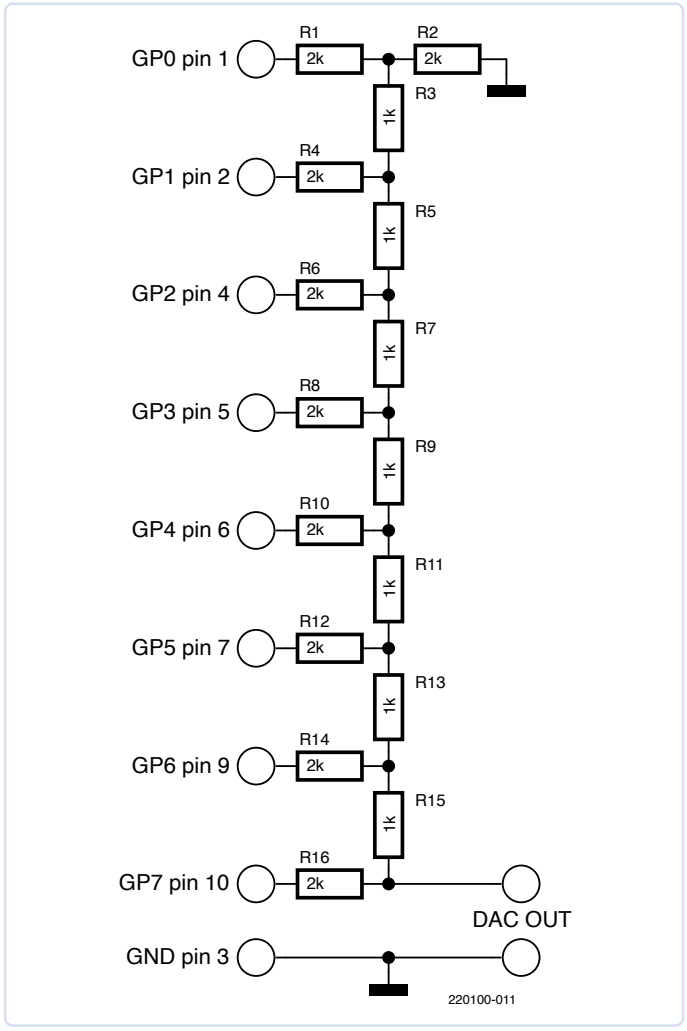
Listing 8

```
001 .program pio_serialiser
002
003 .wrap_target
004     out pins,8 // use autopull
005 .wrap
```

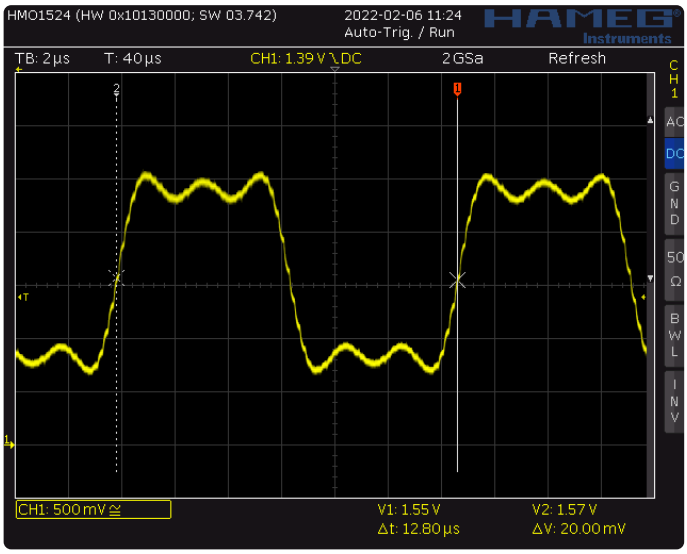
De `out`-instructie brengt de acht minst significante bits van de OSR naar de acht uitgangspinnen GP0...GP7. De OSR wordt dan automatisch opnieuw geladen vanuit de TX FIFO-unit omdat de *auto-pull* PIO-functie was geactiveerd in de configuratieroutine. De volgende acht bits worden dan uitgevoerd met de volgende instructie. De uitvoerlus is dus precies één PIO-klokcyclus lang. De gegevens moeten daarom zo snel mogelijk van het hoofdgeheugen naar de TX FIFO-unit worden getransporteerd. Daartoe zijn twee DMA-controllers van de RP2040 in serie geschakeld in de vorm van een data- en een besturings-DMA. Deze laatste zorgt ervoor dat de data-DMA-controller snel weer wordt opgestart. Met deze techniek kan elke vier systeemkcycli (125 MHz / 4) een nieuwe byte worden uitgevoerd. De eenvoudige D/A-omzetter van **figuur 10** is echter niet geschikt voor een dergelijke hoge uitvoersnelheid.
In **figuur 11** is `clkDiv` ingesteld op 25. De resulterende DAC-klok is 125 MHz / 25 = 5 MHz. Aangezien één periode van het uitgangssignaal 64 samples lang is, wordt de gegenereerde frequentie berekend



Figuur 9. Het spectrum van een 50Hz-sinus na delta/sigma-conversie.



Figuur 10. R2R-DAC-laddernetwerk.



Figuur 11. De grondfrequentie plus twee harmonischen levert een blokgolf op.

als 5 MHz / 64 = 78,125 kHz. De Fourier-benadering van een blokgolf die is opgebouwd uit de grondgolf plus twee harmonischen werd gebruikt om de uitgangs-golfvorm te produceren.

Registerinitialisatie

Het `set`-commando kan worden gebruikt om de registers X en Y met vaste waarden te initialiseren. Aangezien de commando's in 16 bit worden gecodeerd, kan het `set`-commando alleen constanten in een bereik van 0 tot 31 verwerken. Om X of Y met grotere constante waarden te initialiseren, kun je een truc gebruiken.

Door vanuit een C-programma de functie `pio_sm_exec(pio, sm, instruction)` aan te roepen, onderbreek je de instructievolgorde van het PIO-blok `pio` en de toestandsmachine `sm`, en wordt de opdracht `instruction` ingevoegd. De reeks opdrachten van **listing 9** kan worden gebruikt om register X te initialiseren met de waarde 500.000.

Listing 9

```
001 pio_sm_put_blocking(pio, sm, 500000);
002 pio_sm_exec(pio, sm,
      pio_encode_pull(false, false));
003 pio_sm_exec(pio, sm,
      pio_encode_mov(pio_x, pio_osr));
```

Eerst stop je de 500.000 in de TX FIFO-unit, en daarna laat je de `pull`-instructie los op de PIO-controller, die de 500.000 uit de FIFO-unit in de OSR overbrengt. Tenslotte zet de instructie `mov X,OSR` de waarde zoals gewenst in het X-register. Nu kan de eigenlijke PIO worden gestart. Het programma van **listing 10**, dat een LED laat knipperen, dient als voorbeeld.

Listing 10

```
001 .program blink
002 .side_set 1 opt
003    mov y,x   side 1
004 yLoop1:
005    jmp y--   yLoop1
006    mov y,x   side 0
007 yLoop2:
008    jmp y--   yLoop2
```

MicroPython

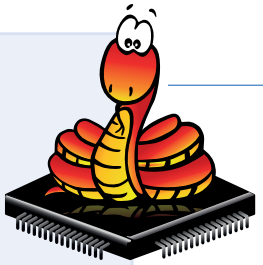
De Raspberry Pi Pico is een platform dat zeer geschikt is voor de ontwikkeling van programma's die in MicroPython zijn geschreven. Er bestaat een overeenkomstige Python-interface voor PIO-programmering. Deze wordt niet geprogrammeerd met de PIO-assembler, maar met de juiste Python-syntaxis, die sterk doet denken aan een assembler-programmeertaal. Een voordeel van het gebruik van MicroPython is dat je complete projecten in één bestand kunt opslaan. Een voorbeeldprogramma is te vinden in de **MicroPython-listing**.

Het eigenlijke PIO-programma is gecodeerd als de Python-functie `pio_prog()`. Het bestaat uit de code in de lus tussen `wrap_target()` en `wrap()`. Het programma schakelt side-set pin 0 aan en uit – in dit geval de LED-pin GP25. De aan/uit-tijd wordt geregeld door een Y-wachtlus. De beginwaarde van het Y-register staat in het X-register. De waarde in het X-register wordt extern geregeld via de TX FIFO-unit. Dit wordt gedaan door de `pull(noblock)`-functie. Als er zich een woord in de TX FIFO-unit bevindt, wordt het naar het OSR-register verplaatst. De parameter `noblock` zorgt ervoor dat de inhoud van het X-register in het OSR wordt geplaatst wanneer de FIFO-unit leeg is en de opdrachtreeks wordt voortgezet.

Tot slot

Hier sluiten we deze inleiding over de PIO-programmering van de RP2040-MCU af. We hebben laten zien dat krachtige interfaces kunnen worden geprogrammeerd met slechts een paar eenvoudige instructies. De PIO-programmeermogelijkheid van deze MCU's maakt ze veelzijdiger, zodat je misschien niet eens het gebruik van FPGA's hoeft te overwegen om de mogelijkheden van de processor uit te breiden. Dit artikel kan slechts een bescheiden overzicht geven van het programmeren van de PIO's. Er valt nog veel meer te ontdekken. Als je een goede praktijkgids wilt, bekijk dan onder andere [1][2][4][5][6][7], waar ook tal van andere voorbeelden in staan.

220100-03



MicroPython-listing

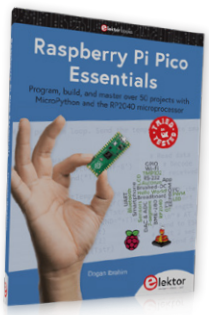
```
001 from machine import Pin
002 from rp2 import PIO, StateMachine, asm_pio
003 from time import sleep
004
005 @asm_pio( sideset_init=(PIO.OUT_LOW))
006 def pio_prog():
007      wrap_target()
008      pull(noblock)
009      out(x, 32)
010
011      mov(y, x) .side(1)
012      label("Loop1")
013      jmp(y_dec, "Loop1")
014
015      mov(y, x) .side(0)
016      label("Loop2")
017      jmp(y_dec, "Loop2")
018      wrap()
019
020 class PIOAPP:
021      def __init__(self, sm_id, pinA, periodLength, count_freq):
023         self._sm = StateMachine(sm_id, pio_prog, freq=count_freq, sideset_base=Pin(pinA)
025         self._sm.active(1) # start PIO execution
026
027      def set(self, value): self._sm.put(value) # put val into TX-FIFO
029
030 pio1 = PIOAPP(0, pinA=25, periodLength=1, count_freq=1_000_000)
032 print("ready1")
033 while True:
034      pio1.set(200000) # 200 ms On/Off time
035      sleep(1) # for one second
036      pio1.set(100000) # 100 ms On/Off time
037      sleep(1) # for one second
038      print("idle")
038      print("idle")
```

Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur via ossmann@fh-aachen.de of naar de redactie van Elektor via redactie@elektor.com.

Gerelateerde producten

- **Raspberry Pi Pico RP2040 (SKU 19562)**
  www.elektor.nl/19562
- **Dogan Ibrahim, Raspberry Pi Pico Essentials (SKU 19673)**
  www.elektor.nl/19673



WEBLINKS

[1] Aan de slag met Raspberry Pi Pico: https://datasheets.raspberrypi.com/pico/getting-started-with-pico.pdf
[2] Handleiding voor de setup van de toolchain door Shawn Hymel: https://tinyurl.com/yde5dd8a
[3] Software-download: http://www.elektormagazine.nl/220100-03
[4] Datasheet van de RP2040: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf
[5] Raspberry Pi Pico SDK: https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf
[6] SDK-documentatie: https://raspberrypi.github.io/pico-sdk-doxygen/index.html
[7] RP2040 MicroPython: https://docs.micropython.org/en/latest/library/rp2.html