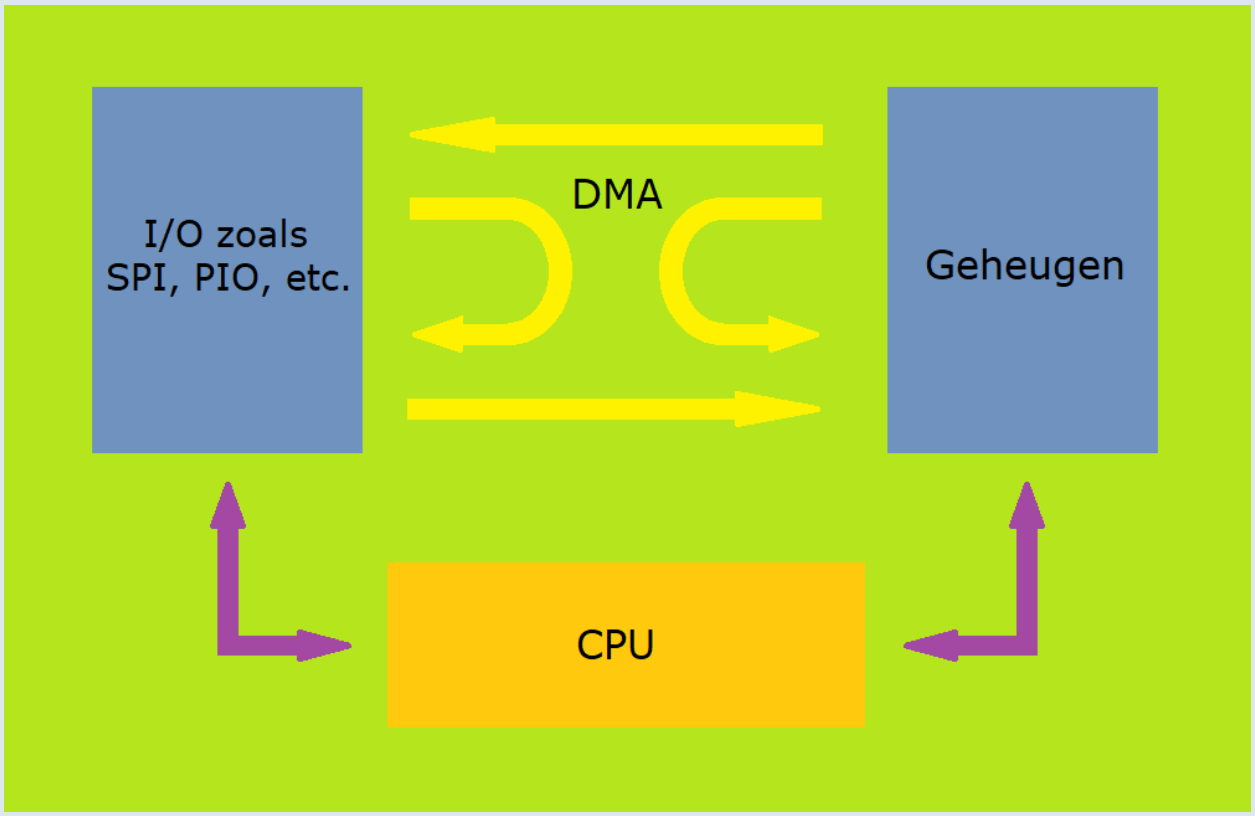
PIO en DMA

1) Wat is de PIO

De PIO is een programmeerbare IO unit, en bestaat in totaal uit 8 state machines met een eigen opcode set. Per vier state machines is er een programma geheugen van 32 instructies beschikbaar. Dat lijkt weinig maar over het algemeen valt dat erg mee.
Meer informatie over de PIO zie mijn lezing op youtube, de PDF ervan vindt je hier

2) Wat is DMA

DMA staat voor Direcy Memory Access, directe geheugen toegang. Het wordt gebruikt om blokken data te verplaatsen van de ene buffer naar een ander, of van een buffer naar een device. Dat kan elke device op de RP2040 zijn, zoals ADC, PWM, PIO, etc.



DMA architectuur

Er zijn een paar zaken om op te letten:

- Het DMA lees en schrijf adres moet worden gezet
- De hoeveelheid en formaat van de te verplaatsen data moet je opgeven; byte, half-word, word
- De configuratie (Dit is de lastigste)
- Eventueel de 'alignment' van de data, die moet op de ring-buffer lengte gezet worden

De DMA registers:

Offset	**+0x0**	**+0x4**	**+0x8**	**+0xC** (Trigger)
0x00 (Alias 0)	READ_ADDR	WRITE_ADDR	TRANS_COUNT	CTRL_TRIG
0x10 (Alias 1)	CTRL	READ_ADDR	WRITE_ADDR	TRANS_COUNT_TRIG
0x20 (Alias 2)	CTRL	TRANS_COUNT	READ_ADDR	WRITE_ADDR_TRIG
0x30 (Alias 3)	CTRL	WRITE_ADDR	TRANS_COUNT	READ_ADD_TRIG

Elk register staat er vier keer in, het laatste (gele) register) start (triggert) een DMA-transfer. We lichten het DMA controle register er even uit (CTRL of CTRL_TRIG). Dat is de lastigste.

Bits	Settings
15:20	TREQ = 0 - 3A = DREQ, 3B = Timer0, 3C = Timer1, etc.
11:14	Koppel deze aan andere DMA = 0 to 11
10	Circulaire buffer op lees pointer = 0, Schrijven = 1
6:9	Buffer grootte 1 = 2, 2 = 4, …, 15 = 32768
5	Verhoog schrijf pointer = 1
4	Verhoog lees pointer = 1
2:3	Data formaat; 0 = Byte, 1 = Half woord, 2 = Woord
1	Hoge prioriteit = 1, 0 = lage prioriteit
0	DMA actief = 1, 0 = uit

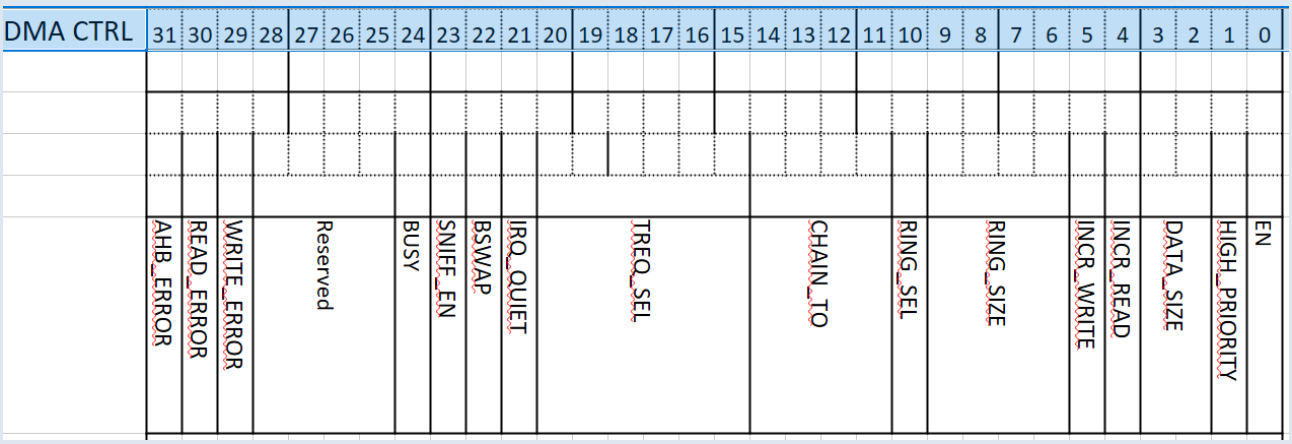
Voorbeeld 1, CMOVE met DMA:

```
: DMA-MOVE8      ( src dst len -- )
  begin
      1000000 5000000C bit** 0=  \ DMA0_CTRL prev. transfer done?
  until
  50000008 !             \ DMA0_COUNT
  50000004 !             \ DMA0_WRITE_ADDR
  50000000 !             \ DMA0_READ_ADDR
  001F8031 5000000C ! ; \ DMA0_CTRL  this triggers transfer
```

Wat staat er in het DMA controle register?

Bits	Settings
	001F8031 = 0001-1111-1000-0000-0011-0001
15:20	3F = Gebruik de maximum snelheid
11:14	0 = Deze DMA is niet gekoppeld aan een andere
10	0 = Circulaire buffer op lees pointer (indien actief)
6:9	0 = De circulaire buffer ongebruikt
5	1 = Verhoog schrijf pointer
4	1 = Verhoog lees pointer
2:3	0 = De DMA data zijn bytes
1	0 = Prioriteit is normaal
0	1 = DMA is actief

Een hulpje voor het DMA controleregister:



Voorbeeld 2, golfvorm generator met timer:

Een simpele golfvorm generator met tabel en timer. Hier wordt pacing timer0 gebruikt, om in een gedefinieerd tempo golfvorm data uit de ‘wave tabel naar de PWM module te sturen.

```
: DMA-MOVE      ( src dst len -- )
    50000008 !            \ DMA0_COUNT
    50000004 !            \ DMA0_WRITE_ADDR
    50000000 !            \ DMA0_READ_ADDR
    000104C4 50000420 !   \ Set pacing timer0 x/y to 200 Hz
    001D8019 5000000C ! ; \ DMA0_CTRL  this triggers it all

: WAVE          ( -- )
    pwm-on
    begin
        begin  1000000 5000000C bit** 0= until \ DMA0 ready
        ‘wave pwm-pulse 100 dma-move           \ Send timed waveform
    key? until ;
```

Wat staat er in het DMA controle register?

Bits	Settings
	001D8019 = 0001-1101-1000-0000-0001-1001
15:20	3B = Gebruik pacing timer0
11:14	0 = Deze DMA is niet gekoppeld aan een andere
10	0 = Circulaire buffer op lees pointer (indien actief)
6:9	0 = Een circulaire buffer wordt niet gebruikt
5	0 = Verhoog schrijf pointer niet
4	1 = Verhoog lees pointer
2:3	2 = De DMA data zijn 32-bits (woorden)
1	0 = Prioriteit is normaal
0	1 = DMA is actief

De pacing timer = 0x000104C4, X=1 en Y=1220

De pacing timer bestaat uit twee 16-bits getallen genaamd X en Y

freq = (sysclock * x / y) / (tabel-lengte * 2)
freq = (125.000.000 * 1 / 1220)/(256 * 2) ~ 200 Hz

Voorbeeld 3, servo aansturing via PIO:

We bouwen een RC-servo aansturing met de PIO. Dankzij de PIO hebben we geen interrupts nodig om nette pulsen daarvoor te genereren. Helaas kunnen we met één state machine slechts met moeite de pulsen voor één servo genereren. En dat kost dan ook nog eens 11 van de 32 opcodes. Kan dat niet anders?

We combineren PIO met DMA :

Hieronder staat een PIO executie machine hier gedefinieerd op PIO-0 en state machine-0:

```
clean-pio  decimal          \ Empty code space mirror
0 0 {pio        \ Servo output on GPIO4 etc. on sm-0 & pio-0

    40000 =freq             \ State machine 0 runs on 40kHz
    4 5 =set-pins           \ GPIO4 to GPIO8 for SET
    0 =out-dir              \ Shift OSR to left!

    31 pindirs set,         \ All 5 pins output
    begin,

       wrap-target         \ Begin of wrap loop
           block pull,     \ 1
           16 exec out,    \ 1 + 1 + [delay]
        wrap               \ End

      ONE                  \ Label one is the time delay
        0 pins set,        \ All outputs off
        9 x set,           \ Wait a while
        begin,
        31 [] x--? until,
    again,                 \ Back to wrap loop
    0 =exec                \ Start this machine
pio}
```

Het PIO programma zet eerst GPIO 4 t/m 8 als uitgang. De lus daarna doet niets anders dan wachten tot er data binnenkomt via TX-fifo, deze data wordt als PIO-opcode uitgevoerd.
Het tweede stukje PIO-code achter ONE, zorgt voor het uitzetten van de laatste puls en voor een nette herhalingstijd. Ongeveer 50Hz, zoals RC-servo’s die verwachten.

Zodra we PIO-opcodes in de fifo zetten worden deze door dit machientje uitgevoerd. Dat kunnen we met de hand uitproberen.

```
hex
    E001 0 >TXF  ( Zet GPIO4 hoog )
    E002 0 >TXF  ( Zet GPIO5 hoog en GPIO4 laag )
    E000 0 >TXF  ( Zet alle bits laag )
```

De volgende stap is dat automatisch te laten doen. Daar schakelen we de DMA voor in. Zodra we daarin slagen, kan de processor druk zijn of stoppen, maar de servo pulsen blijven gegenereerd worden!

Een tabel met PIO-opcodes:

```
hex
\ The 16 half-word opc. table must start on a 32-bytes aligned address!
20 chere 20 mod -  allot

\ Sixteen half-words for totally 5 servo's & pause
chere   \ First 5 servos
    FF01 h,  FF01 h,  EE01 h,   \ Servo 0
    FF02 h,  E102 h,  E102 h,   \ Servo 1
    EC04 h,  E004 h,  E004 h,   \ Servo 2
    FF08 h,  E108 h,  E108 h,   \ Servo 3
    FF10 h,  E110 h,  E110 h,   \ Servo 4
    one> h,                     \ Call to pause
constant OPCODES1
```

We nemen drie PIO-opcodes per servo zodat we een totale resolutie voor de pulslengte van 93 stappen hebben. Met elke PIO opcode krijg je een extra resolutie van 31 stappen.

Even opfrissen, de structuur van een PIO-opcode:

3.4.10. SET

3.4.10.1. Encoding

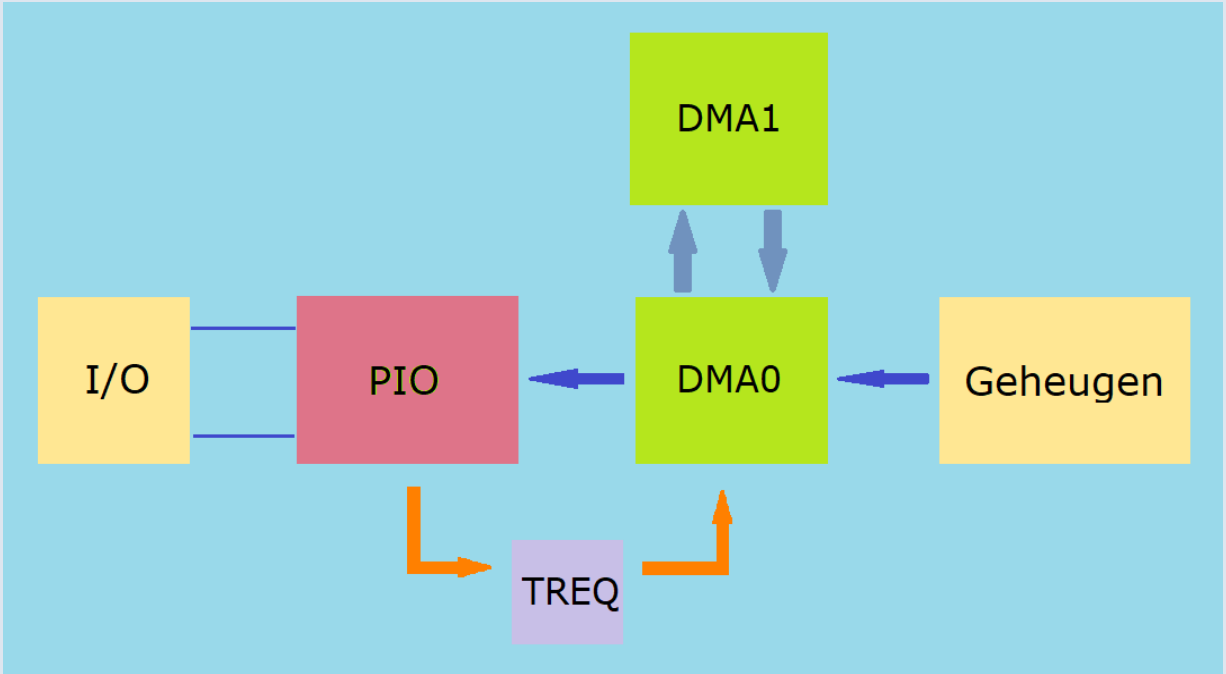
Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SET	1	1	1	Delay/side-set					Destination			Data				

De code om de beide DMA's te starten:

```
create RELOAD  10 ,      \ TRANS_COUNT0 reload
: START-DMA     ( -- )
\ Initialise DMA-0
  opcodes1  50000000 !   \ READ_ADDR0    DMA-0 feeds SM0
  50200010  50000004 !   \ WRITE_ADDR0
  10        50000008 !   \ TRANS_COUNT0
  00000955  5000000C !   \ CTRL_TRIG0 (DREQ=0, Ring=32, CHAIN to DMA-1)
\ Initialise DMA-1
  reload    50000040 !   \ READ_ADDR1    DMA-1 restarts DMA-0
  5000001C  50000044 !   \ WRITE_ADDR1
  1         50000048 !   \ TRANS_COUNT1
  00000009  5000004C ! ; \ CTRL_TRIG1 (CHAIN to DMA-0)
```

Wat doen DMA-0 en DMA-1

- Het lees adres wijst naar de opcode tabel
- Het schrijf adres wijst naar de TX fifo van SM0
- Het aantal te verplaatsen opcodes is 16
- Het controleregister bevat een link (chain) naar DMA-1, de ringbuffer is 32-bytes, de opcode tabel grootte. TREQ = 0, d.w.z. dat hij gekoppeld is aan de TX fifo van SM0, zodat de DMA te vertraagt als de fifo vol is.
- Als alle opcodes verstuurd zijn, geeft hij de controle door aan DMA-1. Deze DMA doet niets anders dan het herstarten van DMA-0 door de tabel lengte opnieuw te zetten in alias-1. Dat is het TRANS_COUNT_TRIG0 register van de DMA.



TREQ en DREQ:

DREQ	DREQ Channel	DREQ	DREQ Channel	DREQ	DREQ Channel	DREQ	DREQ Channel
0	DREQ_PIO0_TX0	10	DREQ_PIO1_TX2	20	DREQ_UART0_TX	30	DREQ_PWM_WRAP6
1	DREQ_PIO0_TX1	11	DREQ_PIO1_TX3	21	DREQ_UART0_RX	31	DREQ_PWM_WRAP7

Het TREQ veld kan DREQ bevatten of een timer selecteren. Informatie over de DREQ codering is terug te vinden op pagina 95 van het RP2040-datasheet. Voor elke I/O-device kun je daar de DREQ waarde vinden. Dit is een klein stukje van die tabel.

De servo positie opgeven:

```
\ Modify the code in the PIO-opcodes array
: SERVO     ( +n servo -- )     \ valid +n = 0 to 80
  4 umin  6 *  opcodes1 +           \ +n a   Adr. of opc. row for servo
  swap  dm 80 umin  dm 13 +         \ a +n1  Scale servo pulse range
  1F /mod 2>r                       \ a      Get delay values for opc.
  r@ for  FF over 1+ c!  2 +  next  \ a      Store maximum delays
  2r>  3 swap - for                 \ a mod  Calc. opcodes to adjust
      E0 or  over 1+ c!  2 +   0    \ a 0    Store mod or zero delay
  next  2drop ;
```

De SERV0 routine past alle vertragingen in één rijtje van drie PIO-opcodes aan, zodat de gewenste pulsbreedte voor die servo gegenereerd wordt. Gebruiken we alle state machines van PIO0, dan kunnen we met deze methode, maximaal 20 servomotoren besturen. In de code dat de werkelijke resolutie 80 stappen is. Dit omdat de minimum pulsbreedte bij de gebruikte servo's ~0,5 milliseconde is. Hoe bereken je dat:

Er zijn 3 pio opcodes voor één servo = 3 x 3 cycles = 9 cycles
De minimum periode = 13 cycles, dat is samen 22 cycles.

Op 40 kHz kost één cycle 25µs
De minimum puls tijd is dan 25µs x 22 = 550µs
De maximum pulstijd = 25µs x ((3x31)+9) = 2550µs

Voorbeeld 4, onafhankelijk DMA/PIO systeem:

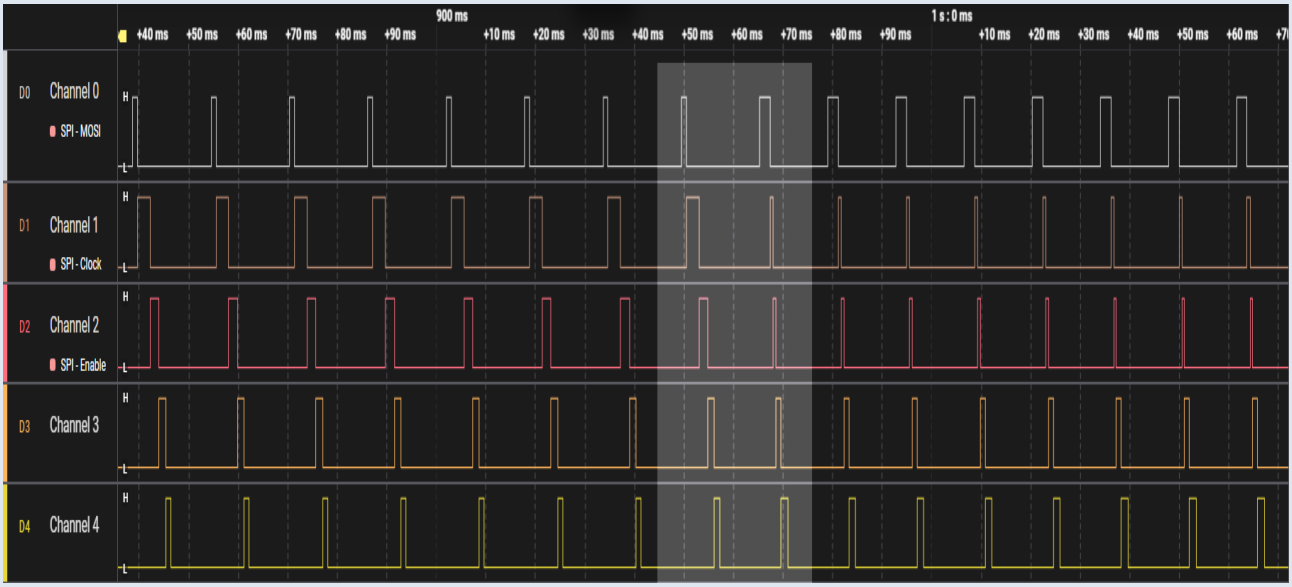
Tot slot 4 DMA’s en één PIO state machine samen een compleet onafhankelijk van de CPU draaiend systeem. Deze code initialiseert 4 DMA controllers om afwisselend twee tabellen met puls data naar de uitgangen te sturen.

```
: RUN-DMA ( ctrl cnt wr rd +n -- )  \ Set DMA registers for DMA +n
    40 *  50000000 +                \ and start DMA +n, the last
    10 bounds do  i !  4 +loop ;    \ register triggers this DMA

: SET-DMA ( ctrl cnt wr rd +n -- )  \ Set DMA registers for DMA +n
    40 * >r  50000000 r@ +          \ does not start DMA +n
    0C bounds do  i !  4 +loop      \ Uses non-trigger registers only
    r> 50000010 + ! ;

\ DMA0 CTRL = DREQ=0, Ring=32, CHAIN=1, CNT=2560
\ DMA1 CTRL = CHAIN=2
\ DMA2 CTRL = DREQ=0, Ring=32, CHAIN=3, CNT=2560
\ DMA3 CTRL = CHAIN=0
: START-DMA     ( -- )
  0955 A00 50200010 opcodes1  0 run-dma   \ DMA 0 runs opcodes1 at SM0
  1009 001 5000009C reload    1 set-dma   \ DMA 1 start DMA 2
  1955 000 50200010 opcodes2  2 set-dma   \ DMA 2 runs opcodes2 at SM0
  0009 001 5000001C reload    3 set-dma ; \ DMA 3 start DMA 0
```

De DMA’s aan het werk:



Nog vragen?