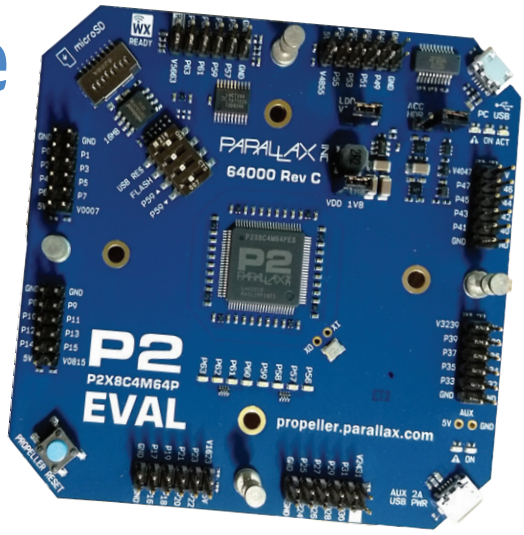
Kennismaking met de Parallax Propeller 2 (deel 3)

smart pins en seriële data (UART)



Mathias Claußen (Elektor)

Vorige keer lieten we een LED branden. Nu laten we hem knipperen. Daarna vormen we met de smart pins een UART om karakters te kunnen versturen. Onze eerste stappen in communicatie met de Parallax Propeller 2!

In de twee vorige artikelen in deze serie zijn we begonnen met het programmeren van de Propeller 2 MCU en hebben we een LED laten branden met SPIN2. Nu zal ik een manier presenteren om hem te laten knipperen; daarna gaan we verder met de I/O-pinnen.

Een simpele oplossing

U kunt een eenvoudig schema volgen: uitschakelen en weer inschakelen. Daarvoor hebben we de volgende ingrediënten nodig:

- pinwrite()
- repeat()
- waitms()

Dit moeten we dan doen:

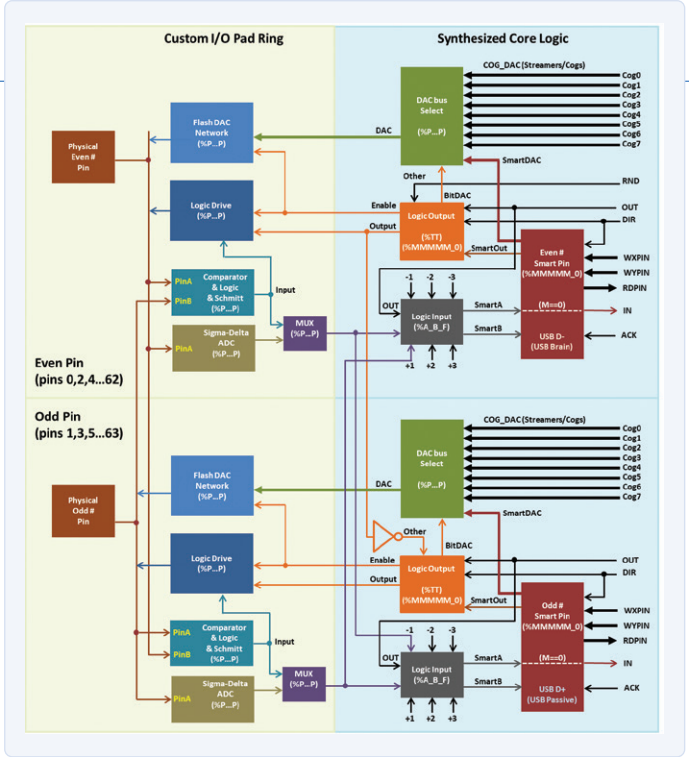
- de LED aanzetten
- 500 ms wachten
- de LED uitzetten
- 500 ms wachten
- en dit herhalen

De code ziet eruit als in **figuur 1** en is te downloaden van de projectpagina bij dit artikel [1].
Nu we pinnen als uitgang hebben gebruikt, zou de volgende stap zijn te laten zien hoe we ze als ingang kunnen gebruiken. Dat doen we nu echter nietn, we komen daar later op terug. Wanneer we een soort seriële data-uitgang hebben om met een PC te communiceren, is het veel leuker de status te tonen die we van een I/O-pin hebben gelezen dan alleen maar een LED te laten knipperen. Bovendien kunnen we deze uitgang gebruiken om statusgegevens te versturen en code te debuggen. Daarom gaan we nu met de smart pins aan de slag.

Slimme pinnen

Tegenwoordig noemen marketing-jongens en -meisjes producten graag 'smart' (zoals 'smart city', 'smart data' en 'smart contracts'). Met de smart pins is het een ander verhaal, omdat ze veel meer kunnen zijn dan gewone I/O-pinnen. Op andere MCU's hebben we soms de mogelijkheid om bij één pin uit meerdere functies te kiezen. Sommige MCU's, zoals de ESP32, hebben een I/O-matrix die elk intern perifeer I/O-signaal naar elke pin kan leiden. In deze gevallen is een I/O-pin nog steeds gewoon een I/O-pin; functies zoals UART, SPI of ADC

Figuur 1. Code om een LED te laten knipperen.



Figuur 2. Het inwendige van de smart pin (bron: Rayman/Parallax).

bevinden zich in een speciaal blok hardware dat gewoon met de pin zelf verbonden is. Met de smart pins van de Propeller 2 verandert dat, omdat de speciale perifere blokken niet langer speciale functieblokken in de MCU zijn die door een I/O-matrix worden geleid, maar geïntegreerd zijn – althans gedeeltelijk – in elke I/O-pin. Vandaar de benaming *smart pin*.

Gebruiker 'rayman' heeft op het Parallax Forum [2] gelukkig goed werk verricht door de community een overzicht te geven van het inwendige van zo'n smart pin. U kunt zijn post over de Propeller 2 en de afbeelding vinden op [3] – of u kunt een blik werpen op **figuur 2**.

In het kader 'Smart Pin-overzicht' is een deel de Propeller 2 -datasheet overgenomen. U kunt zien dat één pin veel functies kan hebben. Op dit moment zijn we alleen geïnteresseerd in *11110* = async serial transmit* om data te verzenden voor latere debug-doeleinden. De eerste stap is om uit te zoeken hoe we de juiste modus voor de pin kunnen instellen en of dat kan met alleen SPIN2 kan of dat we er een beetje assembler doorheen moeten mengen.

Smart Pin-overzicht

00000	smart pin off (default)
00001	long repository (P[12:10] != %101)
00010	long repository (P[12:10] != %101)
00011	long repository (P[12:10] != %101)
00001	DAC noise (P[12:10]= %101)
00010	DAC 16-bit dither, noise (P[12:10] = %101)
00011	DAC 16-bit dither, PWM (P[12:10] = %101)
00100*	pulse/cycle output
00101*	transition output
00110*	NCO frequency
00111*	NCO duty
01000*	PWM triangle
01001*	PWM sawtooth
01010*	PWM switch-mode power supply, V and I feedback
01011	periodic/continuous: A-B quadrature encoder
01100	periodic/continuous: inc on A-rise & B-high
01101	periodic/continuous: inc on A-rise & B-high / dec on A-rise & B-low
01110	periodic/continuous: inc on A-rise (/ dec on B-rise)
01111	periodic/continuous: inc on A-high (/ dec on B-high)
10000	time A-states
10001	time A-highs
10010	time X A-highs/rises/edges -or- timeout on X A-high/rise/edge
10011	for X periods, count time
10100	for X periods, count states
10101	for periods in X+ clocks, count time
10110	for periods in X+ clocks, count states
10111	for periods in X+ clocks, count periods
11000	ADC sample/filter/capture, internally clocked
11001	ADC sample/filter/capture, externally clocked
11010	ADC scope with trigger
11011*	USB host/device (even/odd pin pair = DM/DP)
11100*	sync serial transmit (A-data, B-clock)
11101	sync serial receive (A-data, B-clock)
11110*	async serial transmit (baudrate)
11111	async serial receive (baudrate)

* OUT signal overridden

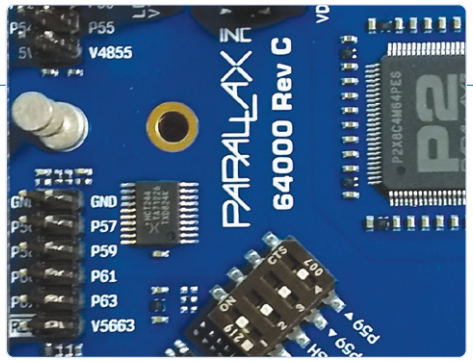
UART-configuratie

Dit is het punt waar we ons achter de oren beginnen te krabben als we voor de eerste keer met SPIN2 en de Propeller 2 werken. De huidige datasheet is min of meer volledig. Maar alles goed lezen en begrijpen kan langer duren dan verwacht. Ons doel is een eenvoudige functie, een `tx()`, die een te verzenden karakter naar een pin stuurt die als een UART werkt. We gebruiken 115200 baud, 8 databits, geen pariteit en slechts één stopbit. Het eerste wat we moeten doen is de pin instellen. Dat gaat als volgt:

- stel de pin in op *async serial transmit*
- stel baudrate en databits in
- schakel de smart pin in (enable)

Deze drie eenvoudige stappen zorgen ervoor dat een pin een zendende UART-pin wordt. Omdat we de code later zullen aanpassen en hergebruiken, stoppen we die in een functie. Een functie bevat eenvoudigweg code of codefragmenten die vaak binnen een programma worden gebruikt. Dit voorkomt knippen en plakken en maakt het ook mogelijk om onderhoud op één plaats te doen. We gebruiken dus een functie die we `serial_start` noemen. Deze heeft geen argumenten en vereenvoudigt de drie hierboven genoemde stappen. De pin die we gebruiken is momenteel hardgecodeerd als pin 57 (een van de LED-pinnen die ook toegankelijk zijn op een van de randconnectoren, zoals te zien in **figuur 3**).

Hier begint de functie met een `PUB` gevolgd door zijn naam en aan het eind lege haken, omdat er geen argumenten zijn.



Figuur 3. Locatie van de pinheader die toegang geeft tot de LED-pinnen.

```
PUB serial_start()
  WRPIN( 57, %01_11110_0  )              'set async tx mode for txpin
  WXPIN( 57, ((217<<16) + (8-1 )) )      'set baud rate to sysclock/115200 and word size to 8
  org
   dirh    #57
  end
```

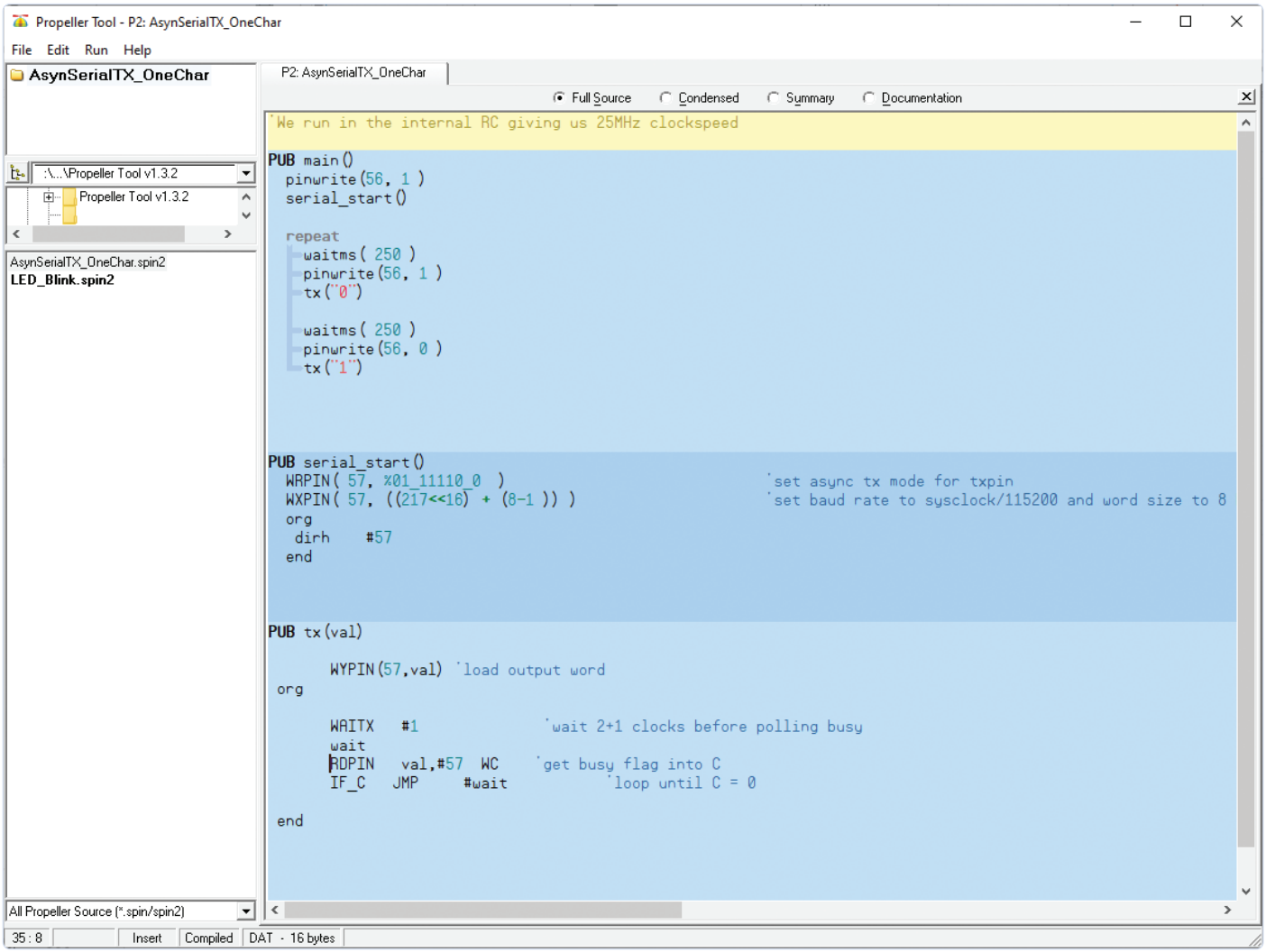
Figuur 4. De code van serial_start().

```
PUB tx(val)

      WYPIN(57,val) 'load output word
org

      WAITX   #1              'wait 2+1 clocks before polling busy
      wait
      RDPIN   val,#57  WC     'get busy flag into C
      IF_C    JMP     #wait           'loop until C = 0

end
```

Figuur 5. De code voor de functie tx.

```
PUB serial_start()
WRPIN( 57, %01_11110_0  )          'set async tx mode
for txpin
WXPIN( 57, ((217<<16) + (8-1 )) ) 'set baud rate to sys
clock/115200 and word size to 8 bits
org                                'start of assembly part
dirh #57
end ' end of assembly part
```

Op regel 1 zien we – zoals hiervoor al aangegeven – de functie, of preciezer gezegd het *function head*. De volgende regel stelt pin 57



Figuur 6. Compleet geassembleerde code.

in op *async serial transmit*, herkenbaar aan de 11110. Het eerste bit is altijd nul, de twee meest significante bits, hier '01', geven aan dat de pin wordt bestuurd door de GPIO of smart pin-functie. We gebruiken hier de SPIN2 `WRPIN`-functie om onze stap één te realiseren. De volgende functie `WXPIN` schrijft, voor een smart pin in asynchrone seriële modus, de gewenste klokdeler en de te gebruiken databits. Om het eenvoudig te houden slaan we het deel met de baudrate-deler nu even over. De waarde voor de baudrate kan worden berekend met systemclock/baudrate – hier 25 MHz/115200 baud – wat resulteert in ongeveer 217. Dit resultaat moet 16 plaatsen naar rechts worden verschoven. Voor het aantal bits dat moet worden overgezet, gebruiken we de formule (gewenst aantal bits – 1) met als resultaat (8 – 1) bits.

Meer tovenarij is niet nodig om de transmissiesnelheid en de databits in te stellen. De volgende drie regels zien er anders uit dan de vorige code. We zien een kleine assembler-sectie, die enige uitleg behoeft. Met `org` kunt u een sectie met assembler-instructies starten. Het assembler-commando `dirh` zal de smart pin-functies inschakelen, zoals nodig voor onze stap drie. Wat nu anders is, is hoe we kunnen aangeven aan welk pinnummer we werken, want dit moet beginnen met een '#'.

De laatste regel `end` beëindigt het assembler-gedeelte. In dit speciale geval vormt deze regel tevens het einde van de functie zelf. We vermijden assembler liever, maar momenteel is er geen SPIN2-equivalent voor de `dirh` assembler-instructie. De geformatteerde en gemarkeerde code is te zien in **figuur 4**.

Nu de pin in de juiste modus staat, kunnen we verder gaan en een functie maken die een karakter verzendt en wacht tot dat klaar is. De functie kan grotendeels uit de datasheet [4], pagina 91, worden overgenomen.

We hebben gezien hoe we functies kunnen bouwen die geen argumenten hebben, wat betekent dat er niets aan wordt doorgegeven. Om een karakter te verzenden, zou het gunstig zijn als we aan de functie kunnen doorgeven wat we willen verzenden. Aangezien we momenteel zoveel mogelijk assembler proberen te vermijden, zullen we, waar mogelijk, de SPIN2-functies gebruiken in plaats van inline assembler.

Verzenden

Voor het verzenden maken we een `tx` functie bijna identiek is aan onze `serial_start()` functie, zoals u ziet in **figuur 5**.

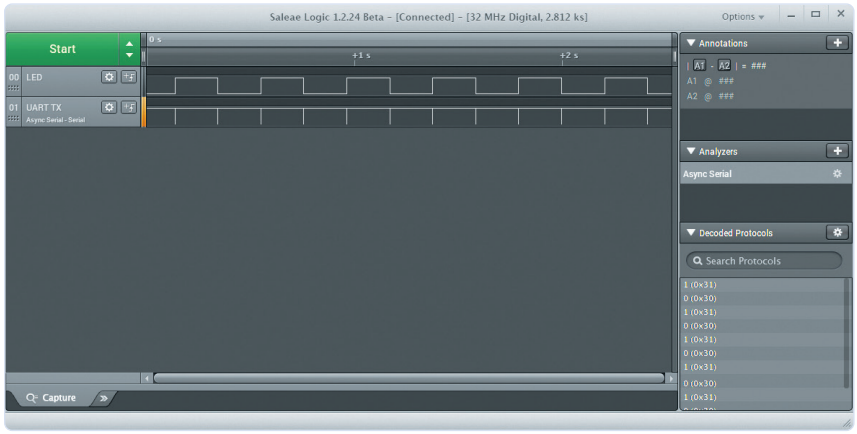
Het zichtbare verschil, naast de naam, is het argument `val` binnen de haakjes. `val` zal het karakter bevatten dat moet worden afgedrukt. Binnen de functie zullen we eerst de waarde in het transmissieregister van pin 57 schrijven met behulp van het `WYPIN` commando. Het volgende gedeelte bestaat weer uit een paar regels assemblercode. We moeten wachten tot de busy-flag voor de zender niet meer geset is en de transmissie is voltooid. Volgens de datasheet moeten we eerst drie CPU-cycli wachten om de vlag op een consistente manier te kunnen lezen. Dit wordt bereikt door de `waitx` instructie met de `#1` parameter, aangezien de uitvoering ervan twee cycli duurt + de hoeveelheid die voor de functie is opgegeven (hier één cyclus). De volgende regel is een label met de naam `wait`, in assembler is dit een plaats waar we later naar toe kunnen springen. `RDPIN` in assembler, zoals hier geschreven, leest de pin-status met carry. Dit wordt aangeduid met de `WC` aan het eind van het statement. Het carry-bit, dat hier dienst doet als bezettoon, is belangrijk omdat het aangeeft of de transmissie voltooid is.

`RDPIN val, #57 WC` leest de status inclusief het carry-bit in onze `val` waarde in. Aangezien de inhoud momenteel wordt verzonden, kunnen we het geheugen van `val` hergebruiken om er de status van de smart pin in te lezen. Het laatste commando `IF_C JMP #wait` is een voorwaardelijke sprong; in BASIC zou dit het beruchte `GOTO`-equivalent zijn gecombineerd met een IF-statement. Vertaald betekent het: *heb je iets gelezen met het carry-bit (hier busy-flag) gezet? Ga terug naar het wait-label en begin van daaruit opnieuw, of ga anders verder*. Onze overdracht is klaar als het carry-bit niet meer gezet is, dus de functie zal doorlopen tot het einde en terugkeren naar waar hij was aangeroepen.

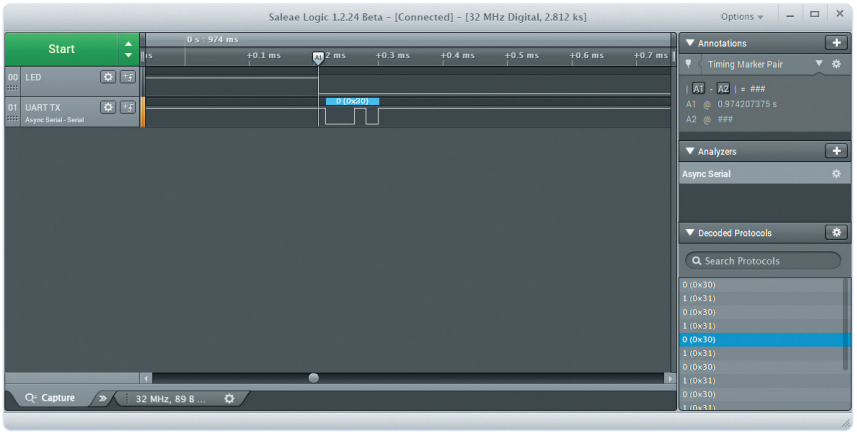
Assembleer alles

We kunnen nu de code assembleren en na elke `pinwrite()` een '0' of '1' zenden door `tx("0")` of `tx("1")` te gebruiken in onze code, zoals te zien in **figuur 6**. Om de uitvoer te kunnen oppikken, moet een USB/serieel-converter worden aangesloten. Voor dit doel hebben we onze vertrouwde Logic 16 genomen en de uitvoer van de LED en de seriële transmissie opgenomen; het resultaat is te bewonderen in **figuur 7** en **figuur 8**.
Maar hoe zit het met strings? Zou het niet leuk zijn om gewoon een `print("Hello World")` te doen en dat serieel te laten verzenden, zoals we dat in de Arduino-wereld doen? Ja, dat kan en dat doen we dus ook. De volgende aflevering van deze serie over de Propeller 2 geeft daarvoor een inleiding.

200479-C-03



Figuur 7. De logic analyzer toont elke 500 ms een uitgezonden karakter.



Figuur 8: Na inzoomen is het uitgezonden karakter te zien.

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

Een bijdrage van
Ontwerp en tekst: **Mathias Claußen**
Redactie: **Jens Nickel, C.J. Abate**
Vertaling: **Jelle Aarnoudse**
Layout: **Giel Dols**

WEBLINKS

[1] **Projectpagina bij dit artikel**: www.elektormagazine.nl/200479-C-03
[2] **Parallax Forum**: https://forums.parallax.com
[3] **Smart Pin Overview by rayman**: https://forums.parallax.com/discussion/171420/smartpin-diagram-now-with-p-p-bit-mode-table/p8
[4] **Propeller 2 Datasheet (voorlopig)**: https://docs.google.com/document/d/1gn6oaT5Ib7CytvlZHacmrSbVBJsD9t_-kmvjd7nUR6o/edit#heading=h.1h0sz9w9bl25