In Robobits83, December 2018, Jaargang 21, nummer 4, schreef Rob Hamerling een artikel over **Micropython en 32-bits microcontrollers**. Bij het artikel hoort de hier gepresenteerde Code en Python script.

Code (zie onderstaande listing)

Class SRF05

Voor de SRF05 is een class gedefinieerd met een aantal methods.

- In de __init__ method wordt opgegeven welke I/O pins van het PyBoard worden gebruikt als trigger en echo, en welke timer wordt gebruikt voor het genereren van een 10 us pulse elke 50 ms (20 Hz PWM frequency). Tevens worden enkele variabelen en constanten gedeclareerd, zoals de list (array) van meetresultaten en de omrekenfactor van us naar mm.
- De method 'trigger' is de ISR (Interrupt Service Routine) voor de timer. Zoals het hoort in een real-time applicatie moet de interrupt handler zo kort en efficient mogelijk worden gehouden. Het enige dat deze ISR hoeft te doen is aansturen van een taak om de lengte van de echo-pulse te meten. Die taak wordt buiten de interrupt-state uitgevoerd!
  NB: Voor de oplettende lezer: het aansturen van de meet-taak gaat via een 'pointer' die in de __init__ method is gespecificeerd. Het voert hier iets te ver om de reden/achtergrond hiervan te behandelen. Zie eventueel referentie 3 onderaan dit artikel.
- In de method 'measure' wordt de lengte van de echo-pulse bepaald en opgeslagen in de volgende entry van de pulse-list. Deze list (=array) wordt gebruikt als een circular buffer, alleen de laatste 5 metingen worden bewaard.
- In de method 'distance' wordt in de afstand (in millimeters) berekend uit het gemiddelde van de laatste 5 metingen. Deze method wordt in de main loop periodiek aangeroepen.
- Met de method 'close' wordt de triggering van de SRF05 gestopt.

Hoofdlijn

De hoofdlijn van de applicatie begint met:

- Specificatie van de pins waaraan de SRF05 aan het PyBoard zijn aangesloten.
  Eenmaal ge-initialiseerd loopt combinatie van trigger en measure methods onafhankelijk van de mainline en wordt de pulse-list up-to-date gehouden.
- Specificatie van welk SPI interface wordt gebruikt voor het 7-segment display en de pin van het PyBoard voor de selectie van het display.

Dan volgt een while-loop welke ongeveer 2 keer per seconde wordt doorlopen:

- opvragen van de actuele afstand
- weergeven van de afstand op het display.

Bij be-eindiging van de loop, bijv. door een keyboard interrupt (Control-C) of een fatale fout wordt de timer interrupt routine gedeactiveerd, het display gewist en het script be-eindigd.

Inhoud file robo_dist.py:

File Edit Format Run Options Window Help

```
"""
    PyBoard variant
    Test of SRF05 ultrasonic sensor with separate trigger and echo leads.
    A timer is used to generated a trigger pulse periodically, the pulse-width
    is determined by a timer channel setting.
    The machine.time_pulse_us() function is used for pulse width measurement.
    Speed of sound is presumably 343 m/s (0.343 mm/us).
    The result is presented on a 7-segment display.
"""

from micropython import const
from pyb import Pin, SPI, Timer, delay, udelay
from machine import time_pulse_us
import max7219_7seg

# for debugging of interrupt handlers:
import micropython
micropython.alloc_emergency_exception_buf(100)

class SRF05(object):
    """ distance sensor
        trigger_pin    - Pin to send trigger pulse to SRF05
        echo_pin       - Pin with echo pulse from SRF05
        timer_number   - Hardware Timer for trigger pulse (natively running at 84 or 168 MHz)
        channel_number - Number of the timer channel (selection is Pin en Timer dependent!)
    """

    def __init__(self, trigger_pin, echo_pin, timer_number=8, channel_number=2):
        # Prepare pins for triggering and measurement of SRF05
        self.trigger_pin = Pin(trigger_pin, Pin.OUT_PP)         # output
        self.echo_pin = Pin(echo_pin, Pin.IN, Pin.PULL_DOWN)    # input, pull-down
        # prepare timer for 1 us cycle and overflow every 50000 cycles (--> 20 Hz).
        self.timer = Timer(timer_number, prescaler=83, period=49999)
        if (self.timer.freq() > 20):                # check if it is a 168 MHz timer
            self.timer.prescaler(167)               # if yes: adapt prescaler
        # prepare a timer-channel to generate a pulse of 10 timer cycles (10 us)
        # at the trigger pin of the SRF05 with every timer overflow and specify ISR
        self.channel = self.timer.channel(channel_number, mode=Timer.PWM,
                        pulse_width=10, pin=self.trigger_pin, callback=self.trigger)
        # init some variables/constants
        self.pulse = [0] * 5                        # 5 subsequent pulse measurements (us)
        self.i = 0                                  # current index in self.pulse
        # convert average of pulse microseconds to millimeters distance
        self.divider = const(len(self.pulse) * 2 * 1000 // 343)  # #meas,fw+bw,mm,spd_of_snd
        # reference (pointer) to measure method
        self._measure = self.measure

    def trigger(self, t):                           # Interrupt Service Routine of timer channel
        # schedule task to measure width of the echo-pulse
        micropython.schedule(self._measure, t)      # task

    def measure(self, t):
        # measure echo pulse (limited to 25000 us, approx 4.3m)
        pulse_us = time_pulse_us(self.echo_pin, 1, 25000)
        if pulse_us > 0:                            # ignore errors (timeouts)
            self.pulse[self.i] = pulse_us           # store pulse width
            self.i = (self.i  + 1) % len(self.pulse)    # increment and wrap around

    def distance(self):
        # Return average of last measurements, converted to millimeters
        return sum(self.pulse) // self.divider      # (average) distance in mm

    def close(self):
        # Deactivate timer
        self.timer.deinit()
```

```
#  ===== M A I N =====

print("Distance measurement with SRF05")    # welcome msg

srf1 = SRF05("Y11", "Y12")                  # SFR05 instance, trigger-pin, echo-pin
                                            # use class defaults for timer and channel

spi = SPI(1, SPI.MASTER)                    # hardware SPI interface 1 (pins X5-X8)
dsp = max7219_7seg.Seg7array(spi, Pin("X4", Pin.OUT_PP))    # chip select Pin X4

while True:
    try:
        dist = srf1.distance()              # obtain current distance
        dsp.display_number(dist)            # show distance
        delay(500)                          # loop about twice a second
    except KeyboardInterrupt:
        break
    except Exception as err:                # fatal exception
        print("Exception:", err)
        break

srf1.close()                                # stop timer
dsp.clear()                                 # clear display
```