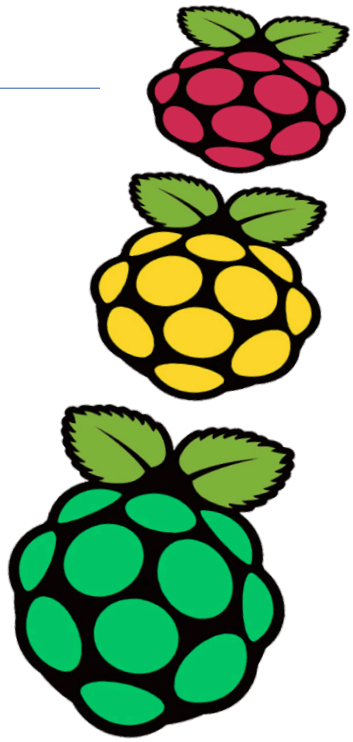
Multitasking met de Raspberry Pi

voorbeeld: controller voor verkeerslichten



Dogan Ibrahim (Groot-Brittannië)

Dit is een voorbeeldproject afkomstig uit het nieuwe boek *Multitasking with RPi*. Het boek is bedoeld voor studenten, praktiserende engineers en hobbyisten die geïnteresseerd zijn in het ontwikkelen van multitasking-projecten met behulp van de programmeertaal Python 3 op de Raspberry Pi.

Multitasking is een *hot topic* op het gebied van microcontrollersystemen, en dan vooral in automatiseringstoepassingen. Naarmate de complexiteit van projecten toeneemt, wordt er meer functionaliteit van gevraagd; bij dergelijke projecten worden verschillende onderling gerelateerde taken gebruikt die op hetzelfde systeem worden uitgevoerd en die de CPU (of meerdere CPU's) delen om de vereiste bewerkingen te implementeren. Als gevolg hiervan is het belang van multitasking in microcontroller-gebaseerde toepassingen de afgelopen jaren gestaag toegenomen. Veel complexe automatiseringsprojecten maken tegenwoordig gebruik van een of andere multitasking-kernel. In het boek wordt de programmeertaal Python 3 gebruikt op de Raspberry Pi 4. Andere modellen van Raspberry Pi zijn eveneens bruikbaar zonder dat de code hoeft te worden aangepast.

Het boek is projectgeoriënteerd. Het belangrijkste doel is om de beginselen van multitasking te leren met Python op Raspberry Pi. In het boek worden veel compleet geteste projecten beschreven die gebuik maken van de multitasking-modules van Python. Elk project wordt in detail beschreven en besproken. Van elk project worden ook de volledige programmalistings verstrekt. Lezers kunnen de projecten gebruiken zoals ze worden beschreven, maar ze ook aanpassen aan hun eigen behoeften.

Voorbeeld: een verkeerslichtcontroller

In dit project ontwerpen we voor gebruik op een kruispunt een eenvoudige verkeerslichtcontroller. Het betreft de kruising van *East Street* en *North Street*. Aan vier zijden van de kruising zijn verkeerslichten. North Street heeft twee oversteekplaatsen voor voetgangers, die met drukknoppen worden bediend. Na indrukken worden alle lichten aan het einde van hun cyclus rood. Dan klinkt er een zoemer om aan te geven dat de voetgangers veilig kunnen oversteken. Ook is er een LCD aangesloten op het systeem waarop te zien is welke cyclus (voetgangers of verkeer) draait. **Figuur 9.12** toont de de plaatsing van de verkeerslichten op de kruising.

In dit project heeft elke verkeerslichtkleur een vaste duur; ook de oversteektijd voor voetgangers (zoemer) ligt vast. Voor het gemak nemen we aan dat beide wegen van de kruising dezelfde timing hebben:

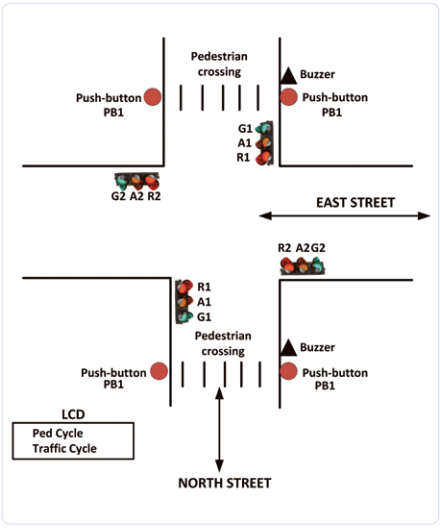
- rood: 19 seconden
- oranje: 2 seconden
- groen: 15 seconden
- oranje + rood: 2 seconden
- oversteektijd voetgangers: 10 seconden

De totale cyclustijd van de verschillende lichten in dit voorbeeldproject bedraagt zodoende 38 seconden.

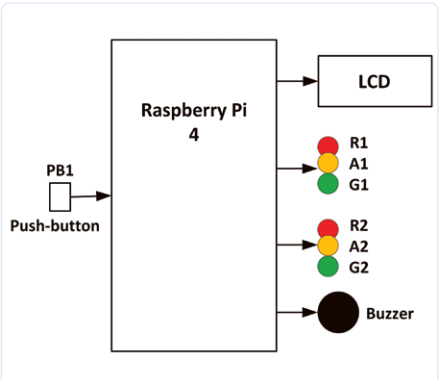
De volgorde van de verschillende lichten is hier als volgt (dat kan overigens van land tot land verschillen):

rood → oranje+rood → groen → oranje ...
groen → oranje → rood → oranje+rood ...

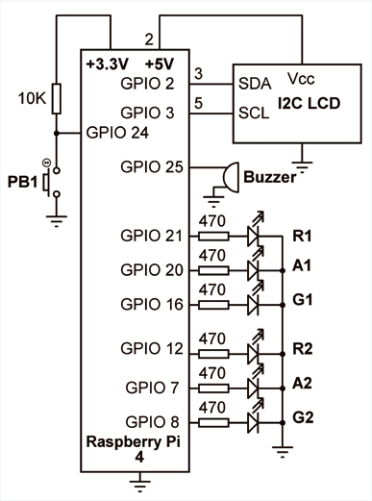
Figuur 9.13 toont het blokschema van het project.



Figuur 9.12. Opstelling van de verkeerslichten op de kruising.



Figuur 9.13. Blokschema van het project.



Figuur 9.14. Het schema van het project.

Het schema van de verkeerslichtcontroller is te zien in **figuur 9.14**. In dit project gebruiken we rode (R), oranje (A, staat voor *amber*) en groene (G) LED's die de echte verkeerslichten simuleren.
De volgende verbindingen moeten tussen de Raspberry Pi en verkeerslichten worden gemaakt:

Raspberry Pi	Verkeerslicht
GPIO 21	LED R1
GPIO 20	LED A1
GPIO 16	LED G1
GPIO 12	LED R2
GPIO 7	LED A2
GPIO 8	LED G2
GPIO 25	Zoemer
GPIO 2	LCD SDA
GPIO 3	LCD SCL
GPIO 24	PB1 (druktoets)

Figuur 9.15 geeft de volledige programmalisting (programmanaam: traffic.py; download: [1]). Aan het begin van het programma worden de modules *RPi*, *time*, *I2C LCD driver* en *multiprocessing* geïmporteerd in het programma en worden de twee wachtrijen `pedq` en `lcdq` aangemaakt.
In het programma zijn twee functies gedefinieerd met de namen `ONOF` en `CONF_OUT`. De functie `ONOF` heeft twee argumenten: `port` en `state`. Deze functie stuurt de status (0 of 1) naar de opgegeven GPIO-poort. De functie `CONF_OUT` heeft één parameter: `port`.

Figuur 9.15. Listing van het programma traffic.py.

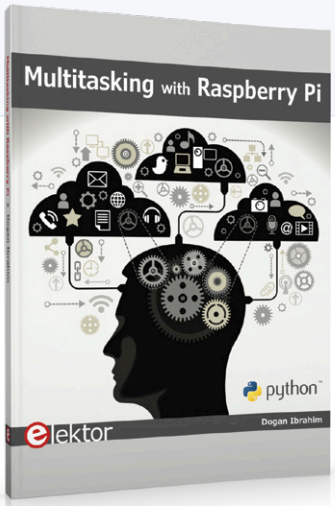
```
#--------------------------------------------------------------------
#                    TRAFFIC LIGHTS CONTROLLER
#                    =========================
#
# This is a traffic lights controller project controlling lights
# at a junction. 6 LEDS are used to represent the traffic lights.
# Additionally a button is used for pedestrian crossing, and an
# LCD shows the state of the traffic lights at any time
#
# Author: Dogan Ibrahim
# File  : traffic.py
# Date  : May 2020
#--------------------------------------------------------------------
import RPi.GPIO as GPIO                         # Import RPi
import multiprocessing                          # Import multiprocessing
import time                                     # Import time
import RPi_I2C_driver                           # I2C library
LCD = RPi_I2C_driver.lcd()                      # Import LCD
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)                          # GPIO mode BCM
pedq = multiprocessing.Queue()                  # Create queue
lcdq = multiprocessing.Queue()                  # Create queue

#
# This function sends data 'state (0 or 1)' to specified port
#
def ONOF(port, state):
   GPIO.output(port, state)

#
# This function configures the specified port as output
#
def CONF_OUT(port):
   GPIO.setup(port, GPIO.OUT)

#
# Process to control the lights
#
def Lights():                                   # Process Lights
   R1=21; A1=20; G1=16                          # LED connections
   R2=12; A2=7;  G2=8                           # LED conenctions
   Buzzer=25                                    # Buzzer connection
   CONF_OUT(R1); CONF_OUT(A1); CONF_OUT(G1)     # Configure
   CONF_OUT(R2); CONF_OUT(A2); CONF_OUT(G2)     # Configure
   CONF_OUT(Buzzer)                             # Configure
   ONOF(R1,0); ONOF(A1,0); ONOF(G1,0); ONOF(R2,0); ONOF(A2,0); ONOF(G2,0)
   ONOF(Buzzer, 0)

   RedDuration = 15
   GreenDuration = 15
   AmberDuration = 2

#
# Control the traffic light sequence
#
   while True:                                  # Do forever
      ONOF(R1,0); ONOF(A1,0); ONOF(G1,1); ONOF(R2,1); ONOF(A2,0);
   ONOF(G2,0)
      time.sleep(RedDuration)
      ONOF(G1,0); ONOF(A1,1)
      time.sleep(AmberDuration)
      ONOF(A1,0); ONOF(R1,1); ONOF(A2,1)
      time.sleep(AmberDuration)
      ONOF(A2,0); ONOF(R2,0); ONOF(G2,1)
      time.sleep(GreenDuration)
```

```
      ONOF(G2,0); ONOF(A2,1)
      time.sleep(AmberDuration)
      ONOF(A2,0); ONOF(A1,1); ONOF(R2,1)
      time.sleep(AmberDuration)

      while not pedq.empty():                # If ped request
         lcdq.put(1)
         ONOF(G1,0); ONOF(R1,1); ONOF(A1,0)  # Only RED ON
         ONOF(G2,0); ONOF(R2,1); ONOF(A2,0)  # Only RED ON
         d = pedq.get()                      # Clear ledq
         ONOF(Buzzer, 1)                     # Buzzer ON
         time.sleep(10)                      # Wait 10 secs
         ONOF(Buzzer, 0)                     # Buzzer OFF
         d = lcdq.get()                      # Clear lcdq


def Pedestrian():                            # Process Pedestrian
   PB1 = 24
   GPIO.setup(PB1, GPIO.IN)                  # PB1 is input

   while True:                               # Do forever
      while GPIO.input(PB1) == 1:            # PB1 not pressed
         pass
      pedq.put(1)                            # Send to Ped queue
      while GPIO.input(PB1) == 0:            # PB1 not released
         pass

#
# Create the processes
#
p = multiprocessing.Process(target = Lights, args = ())
q = multiprocessing.Process(target = Pedestrian, args = ())
p.start()
q.start()

#
# LCD Display control. Display 'Ped Cycle' or 'Traffic Cycle'
#
LCD.lcd_clear()                              # Clear LCD
LCD.lcd_display_string("TRAFFIC CONTROL", 1) # Heading

while True:                                  # DO forever
   if not lcdq.empty():
      LCD.lcd_display_string("Ped Cycle    ", 2)
   else:
      LCD.lcd_display_string("Traffic Cycle", 2)
   time.sleep(1)
```

Afbeelding 9.16.
Weergave op het LC-display.

SHOPPING LIST

- **(Engelstalig) boek "Multitasking with Raspberry Pi"**
  www.elektor.nl/19357
- **(Engelstalig) e-boek "Multitasking with Raspberry Pi"**
  www.elektor.nl/19360



Deze functie configureert de betreffende GPIO-poort als uitgang.
Het programma is twee processen rijk: `Lights` en `Pedestrian`. Aan het begin van het proces `Lights` worden de verbindingen tussen de Raspberry Pi en de LED's (verkeerslichten) en de zoemer gedefinieerd en de betreffende poorten worden als uitgang geconfigureerd. Bovendien worden al deze poortuitgangen op 0 gezet, zodat alle LED' s en de zoemer UIT zijn. De LED's worden dan in de juiste volgorde ent met de juiste timing geactiveerd. Aan het einde van de functie wordt gecontroleerd of de voetgangersknop is ingedrukt. De voetgangersknop is ingedrukt als de wachtrij `pedq` niet leeg is. Tijdens de voetgangerscyclus gaan de rode lichten voor beide straten AAN om het verkeer te stoppen zodat de voetgangers veilig kunnen oversteken.
Ook wordt de zoemer tijdens de voetgangerscyclus gedurende 10 seconden geactiveerd om aan te geven dat veilig kan worden overgestoken.
Het voetgangersproces houdt continu drukknop PB1 in de gaten. Als de knop wordt ingedrukt, wordt er een 1 naar de wachtrij `pedq` gestuurd, zodat het `Lights`-proces dit gemakkelijk kan controleren en de voetgangerscyclus kan starten.
Het hoofdprogramma stuurt ook het LCD aan. Bij het starten van het programma wordt het bericht 'TRAFFIC CONTROL' op de eerste regel van het display getoond. Voor de tweede regel wordt continu de wachtrij `lcdq` gecontroleerd en ofwel 'Ped Cycle' (voetgangers) ofwel 'Traffic Cycle' (verkeer) getoond (zie figuur 9.16).

200381-04

WEBLINK

[1] **Download van het Traffic.py-programma**: www.elektormagazine.nl/200381-04