# Project IoT : Datacommunicatie

**Project leden :** Yoram van Hamme, Rune De Mesmaeker, Mike Aelbrecht, Wouter Bonnarens

# Inhoudstabel

1.0 Algemeen voorstel

2.0 SPI
- 2.1 Algemeen
- 2.2 Hardware
- 2.3 Bedrading
- 2.4 Software

3.0 I2C
- 3.1 Algemeen
- 3.2 Hardware
- 3.3 Bedrading
- 3.4 Software

4.0 MQTT
- 4.1 Algemeen
- 4.2 Software

5.0 Blokschema

# 1.0 : Algemeen

Dit project bestaat uit twee bedrade communicaties en een draadloze communicatie.

Het doel van dit project is om met vier knoppen een RGB led te besturen via verschillende communicaties.

De data die de knoppen sturen verwerkt worden door SPI aan de hand van een IC. De data die naar de RGB led wordt verstuurd zal worden verwerkt door I2C communicatie, dit zal gebeuren met een ATTiny. De data van de knoppen worden draadloos verstuur met MQTT en dit zal gebeuren aan de hand van twee Raspberry Pico's.

Hierdoor hebben we vier knoppen waarmee we rood, groen en blauw kunnen aansturen aan de RGB led en de vierde knop zal gebruikt worden om de RGB led uit te zetten.

Voor deze Raspberry Pico's te programmeren gebruiken we MicroPython, een vorm van Python speciaal ontworpen voor de Raspberry Pico.

Voor de ATTiny gebruiken we C / C++.

# 2.0 : SPI

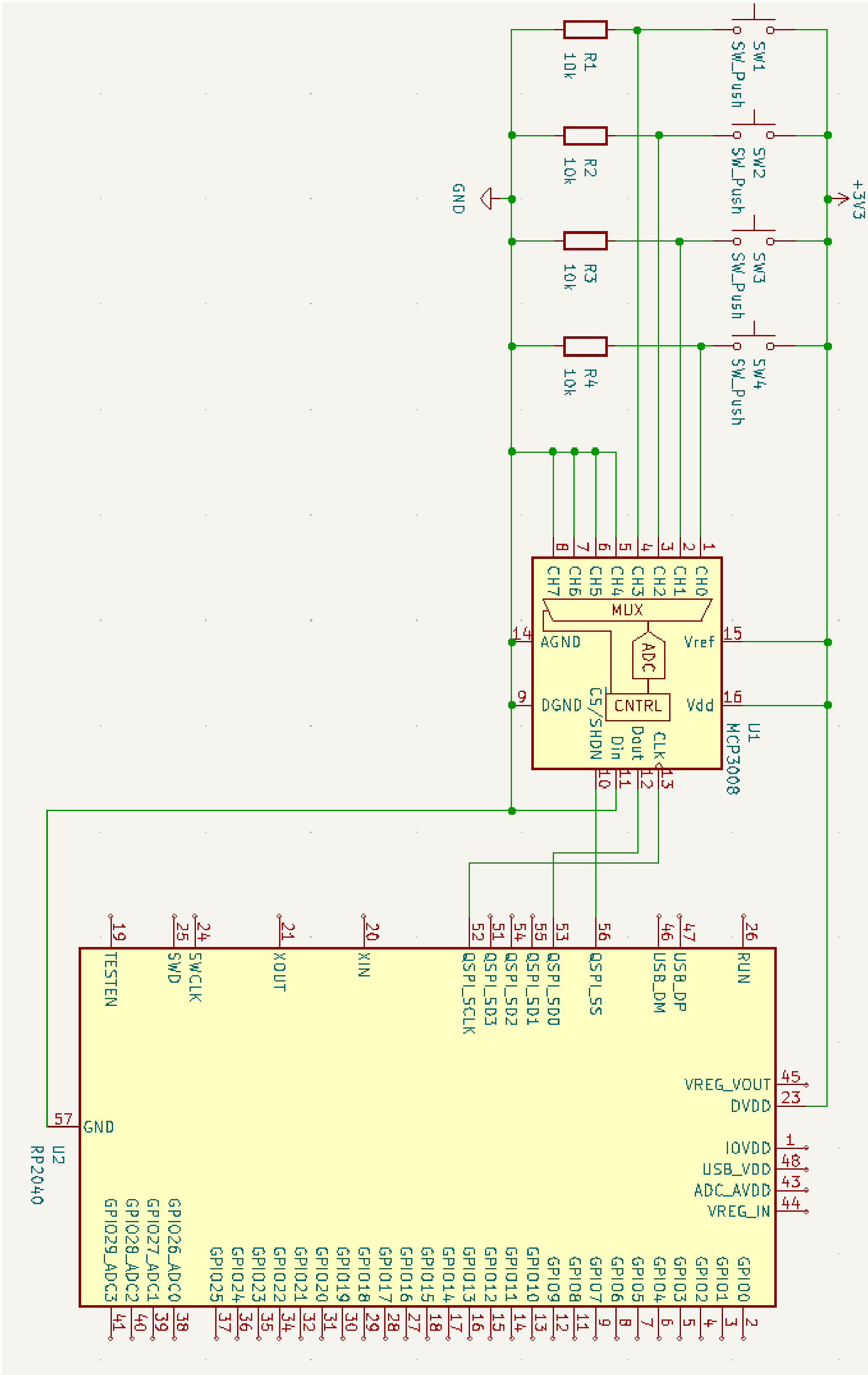
## 2.1 : Algemeen

Voor de vier knoppen de besturen zullen we het SPI protocol gebruiken. Er zijn veel opties voor SPI te gebruiken maar wij gebruikte de MCP3008 IC. Deze IC kan 8 analoge kanalen inlezen en deze analoge data digitaal doorsturen. We zullen deze IC dus gebruiken om de data van de vier knoppen door te sturen naar de Raspberry Pico.

## 2.2 : Hardware

- 4x knoppen
- 4x 10kΩ weerstanden
- 1x MCP3008
- 1x Raspberry Pico

# 2.3 : Bedrading



## 2.4 : Software

Voor de MCP3008 gebruikte we de volgende library :

```
import machine


class MCP3008:

    def __init__(self, spi, cs, ref_voltage=3.3):
        """
        Create MCP3008 instance

        Args:
            spi: configured SPI bus
            cs:  pin to use for chip select
            ref_voltage: r
        """
        self.cs = cs
        self.cs.value(1) # ncs on
        self._spi = spi
        self._out_buf = bytearray(3)
        self._out_buf[0] = 0x01
        self._in_buf = bytearray(3)
        self._ref_voltage = ref_voltage

    def reference_voltage(self) -> float:
        """Returns the MCP3xxx's reference voltage as a float."""
        return self._ref_voltage

    def read(self, pin, is_differential=False):
        """
        read a voltage or voltage difference using the MCP3008.

        Args:
            pin: the pin to use
            is_differential: if true, return the potential difference between two pins,


        Returns:
            voltage in range [0, 1023] where 1023 = VREF (3V3)

        """

        self.cs.value(0) # select
        self._out_buf[1] = ((not is_differential) << 7) | (pin << 4)
        self._spi.write_readinto(self._out_buf, self._in_buf)
        self.cs.value(1) # turn off
        return ((self._in_buf[1] & 0x03) << 8) | self._in_buf[2]
```

```python
from machine import Pin, SPI
from time import sleep, sleep_ms
from mcp3008 import MCP3008
import network
from simple import MQTTClient

ssid = "MiniRouter_M"
password = "MikeAelbrecht1"

spi = SPI(0, sck=Pin(2),mosi=Pin(3),miso=Pin(4), baudrate=100000)
cs = Pin(22, Pin.OUT)
cs.value(1) # disable chip at start

chip = MCP3008(spi, cs)

c = None

topic = "/data2023/data"

def connect_wifi():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(ssid, password)
    print("Connected to the WiFi")

def connect_mqtt():
    global c
    c = MQTTClient("pico_master", "broker.hivemq.com")

```

```python
def read_data():
    value1 = chip.read(0, True)
    value2 = chip.read(1, True)
    value3 = chip.read(4, True)
    value4 = chip.read(7, True)

    if value1 == 0:
        print(f"KNOP1   {value1}")
        send_data("ra")
    if value2 == 0:
        print(f"KNOP2   {value2}")
        send_data("ga")
    if value3 == 0:
        print(f"KNOP3   {value3}")
        send_data("ba")
    if value4 == 0:
        print(f"KNOP4   {value4}")
        send_data("au")

    sleep(0.1)

def send_data(_data):
    data = b"" + _data
    c.connect()
    c.publish(topic, data)
    c.disconnect()

if __name__ == "__main__":
    connect_wifi()
    connect_mqtt()

    while True:
        read_data()
```

# 3.0 : I2C

## 3.1 : Algemeen

We ontvangen data van de Raspberry Pico die moet verstuurd worden naar de RGB led. Deze data verwerken we aan de hand van het I2C protocol. Het I2C protocol zal worden gedaan door de ATTiny, de ATTiny zal de data verwerken en correct versturen naar de RGB led.

## 3.2 : Hardware

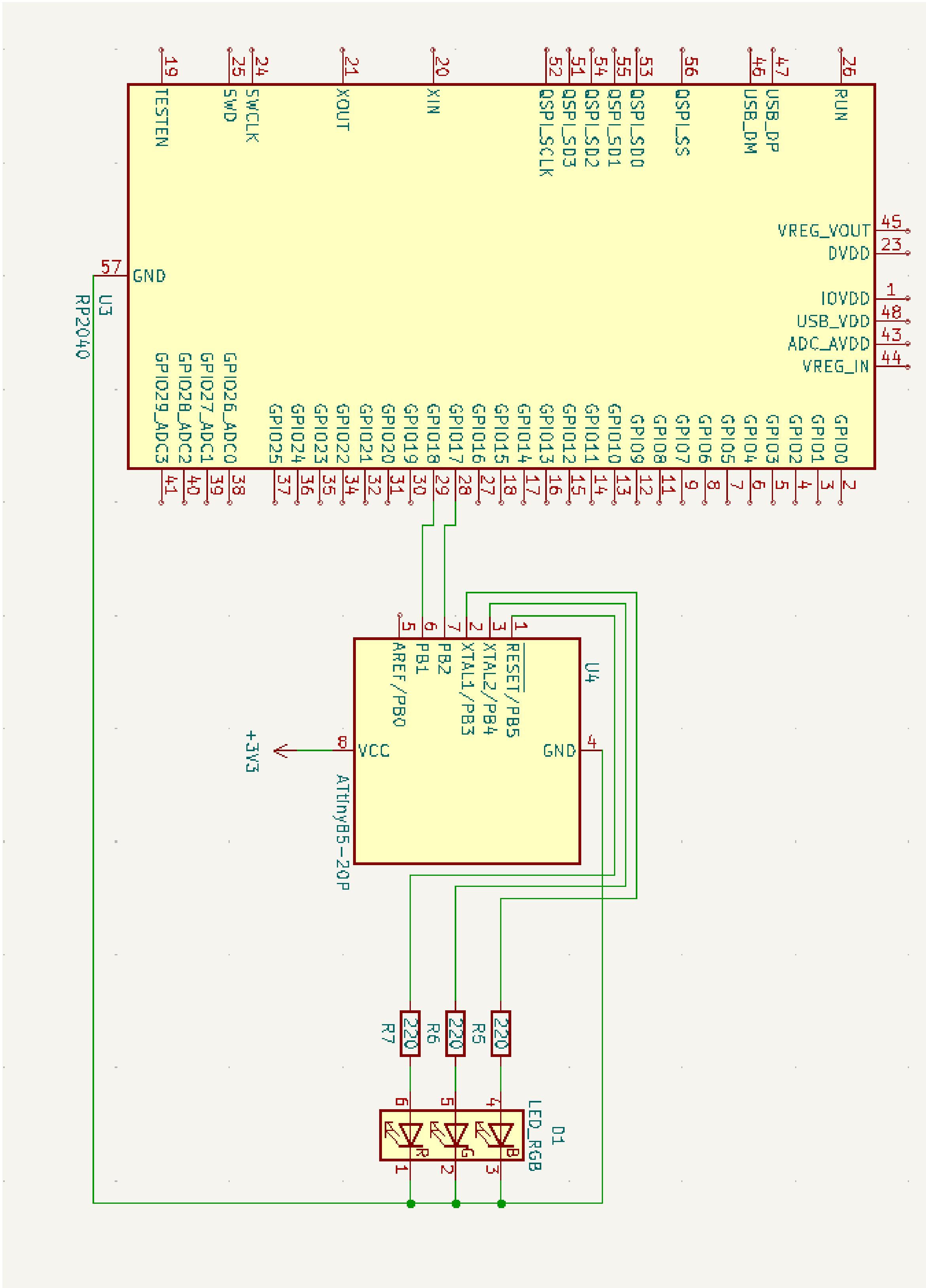
3x 220Ω Weerstanden

1x RGB led

1x ATTiny85

1x Raspberry Pico

## 3.3 : Bedrading



# 3.4 : Software

```
#define I2C_SLAVE_ADDRESS 0x13 // the 7-bit address (remember to change this when adapting this example)
// Get this from https://github.com/rambo/TinyWire
#include <TinyWireS.h>
// The default buffer size, Can't recall the scope of defines right now
#ifndef TWI_RX_BUFFER_SIZE
#define TWI_RX_BUFFER_SIZE ( 16 )
#endif

#define R_pin 1
#define G_pin 3
#define B_pin 4

bool handleData = false;
uint8_t dataArray[2];

volatile uint8_t i2c_regs[] =
{
  0xDE,
  0xAD,
  0xBE,
  0xEF,
};

// Tracks the current register pointer position
volatile byte reg_position;
const byte reg_size = sizeof(i2c_regs);

```

```
/**
 * This is called for each read request we receive, never put more than one byte of data (with TinyWireS.send) to the
 * send-buffer when using this callback
 */
void requestEvent()
{
  TinyWireS.send(i2c_regs[reg_position]);
  // Increment the reg position on each read, and loop back to zero
  reg_position++;
  if (reg_position >= reg_size)
  {
    reg_position = 0;
  }
}

/**
 * The I2C data received -handler
 *
 * This needs to complete before the next incoming transaction (start, data, restart/stop) on the bus does
 * so be quick, set flags for long running tasks to be called from the mainloop instead of running them directly,
 */
void receiveEvent(uint8_t byte_count)
{
  if (!TinyWireS.available() || byte_count != 2)
  {
    for (int i = 0; i < 2; i++)
    {
      digitalWrite(R_pin, HIGH);
      tws_delay(100);
      digitalWrite(R_pin, LOW);
      tws_delay(100);
    }

    return;
  }
```

```
  handleData = true;

  dataArray[0] = TinyWireS.receive();
  dataArray[1] = TinyWireS.receive();

  if (dataArray[0] == 'r' && dataArray[1] == 'a')
  {
    digitalWrite(R_pin, HIGH);
    digitalWrite(G_pin, LOW);
    digitalWrite(B_pin, LOW);
  }

  if (dataArray[0] == 'g' && dataArray[1] == 'a')
  {
    digitalWrite(G_pin, HIGH);
    digitalWrite(R_pin, LOW);
    digitalWrite(B_pin, LOW);
  }

  if (dataArray[0] == 'b' && dataArray[1] == 'a')
  {
    digitalWrite(B_pin, HIGH);
    digitalWrite(R_pin, LOW);
    digitalWrite(G_pin, LOW);
  }

  if (dataArray[0] == 'a' && dataArray[1] == 'u')
  {
    digitalWrite(R_pin, LOW);
    digitalWrite(G_pin, LOW);
    digitalWrite(B_pin, LOW);
  }

  uint8_t data =  dataArray[1];
  TinyWireS.send(data);
}
```

```
void setup()
{
  TinyWireS.begin(I2C_SLAVE_ADDRESS);
  TinyWireS.onReceive(receiveEvent);
  TinyWireS.onRequest(requestEvent);

  // Whatever other setup routines ?
  pinMode(R_pin, OUTPUT);
  pinMode(G_pin, OUTPUT);
  pinMode(B_pin, OUTPUT);

  digitalWrite(B_pin, HIGH);
  delay(500);
  digitalWrite(B_pin, LOW);
}

void loop()
{
  TinyWireS_stop_check();
}
```

# 4.0 : MQTT

## 4.1 : Algemeen

Voor de input data van de eerste Raspberry Pico te versturen naar de tweede Raspberry Pico gebruiken we het MQTT protocol. Via dit protocol kunnen we data draadloos versturen tussen de twee Raspberry Pico’s. Zo kunnen we de input data van de knoppen draadloos leveren aan de output RGB led.

## 4.2 : Software

```python
import time
from machine import Pin, I2C

from wifi import Wifi
from simple import MQTTClient

ssid = "MiniRouter_M"
password = "MikeAelbrecht1"

mqtt_client = "pico_slave"
mqtt_server = "broker.hivemq.com"

i2c_addres = 19

Wifi(ssid, password)
i2c = I2C(0, sda=Pin(0), scl=Pin(1), freq=400000)

def sub_cb(topic, msg):
    msg = msg.decode()
    print(msg)
    if msg == "ra":
        i2c.writeto(i2c_addres, b"ra")
    elif msg == "ga":
        i2c.writeto(i2c_addres, b"ga")
    elif msg == "ba":
        i2c.writeto(i2c_addres, b"ba")
    elif msg == "au":
        i2c.writeto(i2c_addres, b"au")
    else:
        print(f"An invalid message")

if __name__ == "__main__":
    mqtt = MQTTClient(mqtt_client, mqtt_server)
    mqtt.set_callback(sub_cb)
    mqtt.connect()
    mqtt.subscribe(b"/data2023/data")

    while True:
        mqtt.check_msg()
        time.sleep(1)

    mqtt.disconnect()


```

# 5.0 : Blokschema

Op volgende afbeelding zie je de totale schakeling.