Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий

# Курсовая работа

Колесная платформа
по дисциплине «Микропроцессорные системы»

Выполнили: Ферапонтов М.В.
Савчук А.А.
Дорошин Д.А.
Луцай П.П.
Артеев Д.Д.
Яровой В.Д.
Группа: гр. 3530904/00104

Проверил: Круглов С.К.

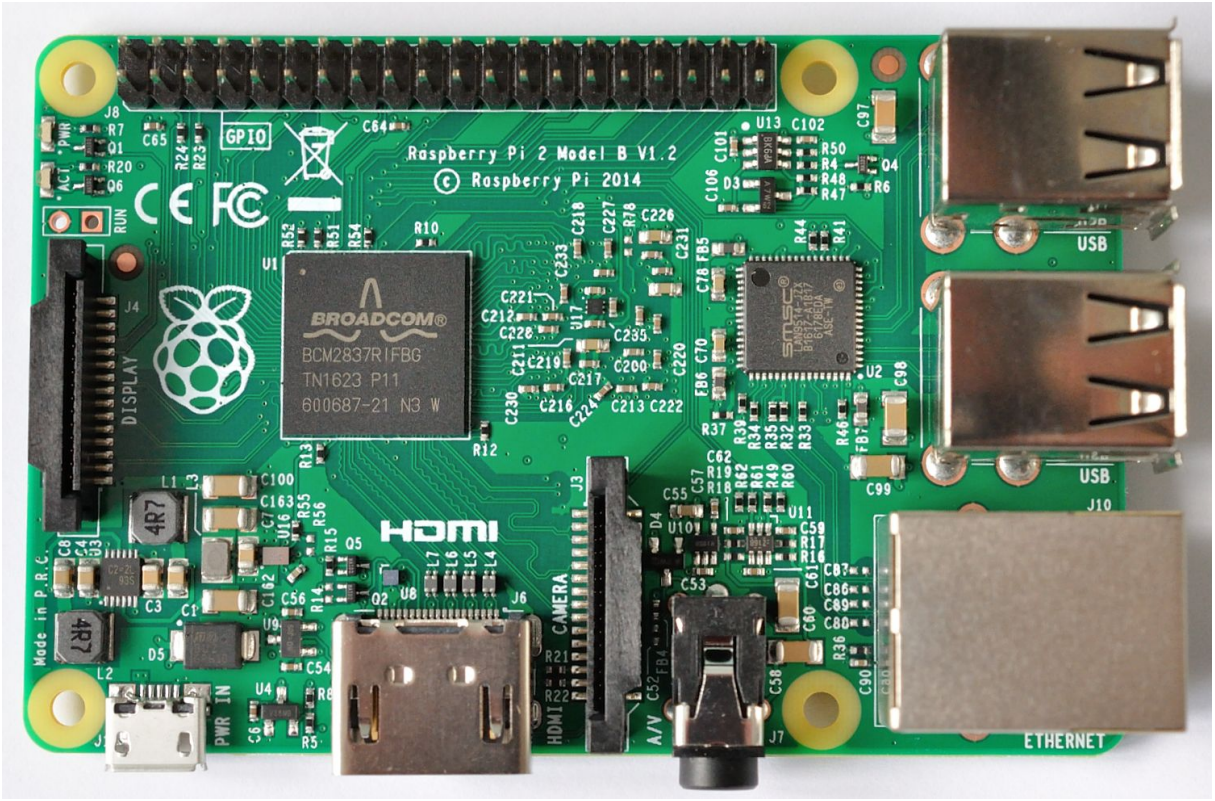
Санкт-Петербург
2023

# Содержание

# 1 Аппаратная реализация1

## 1.1 Raspberry pi 2b+

При разработке был использован Raspberry Pi 2 model B+.




Характеристики:

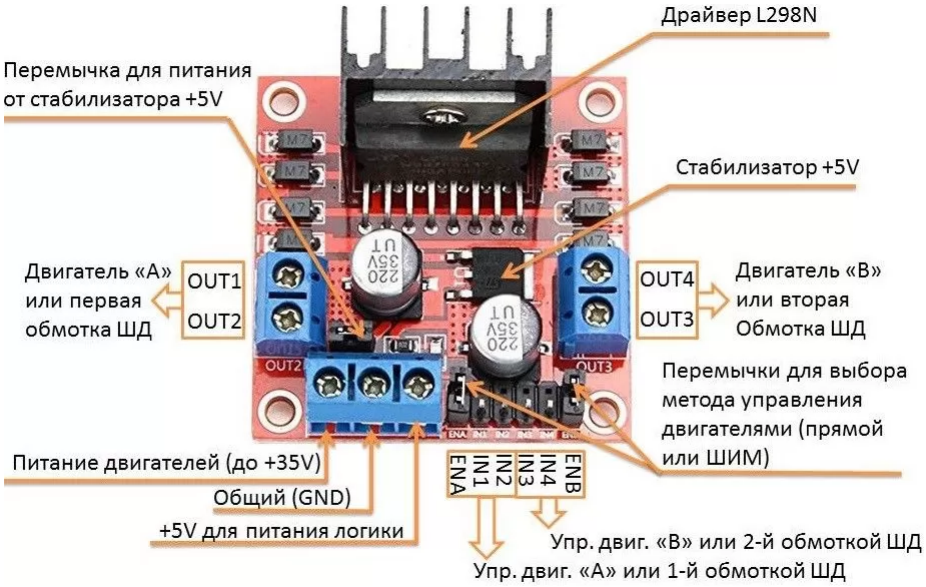
- Процессор - ARM Cortex-A7 CPU 900MHz
- Количество ядер - 4
- Оперативная память - 1Gb

## 1.2 Используемые детали и модули

Были использованы детали из набора Arduino

1. Драйвер управления движения моторов L289N
2. Колеса 4 шт.
3. Кнопка для управления движения 4шт.
4. Резистр 22Ω 4 шт
5. Корпус
6. Плата расширения

Драйвер L298N используется для многофункционального управления двигателями постоянного тока. Схема модуля, состоящая из двух H-мостов, позволяет подключать к нему один биполярный шаговый двигатель или одновременно два щёточных двигателя постоянного тока. При этом есть возможность изменять скорость и направление вращения моторов. Управление осуществляется путём подачи соответствующих сигналов на командные входы, выполненные в виде штыревых контактов. На рисунке показан внешний вид модуля с кратким описанием всех его составляющих.




## 1.3 Широтно-импульсная модуляция

В драйвере L289N *ENA*, *ENB* – контакты для активации/деактивации первого и второго двигателей или соответствующих обмоток ШД. Подача логической единицы на эти контакты разрешает вращение двигателей, а логический ноль – запрещает. Для изменения скорости вращения щёточных моторов на эти контакты подаётся ШИМ-сигнал.

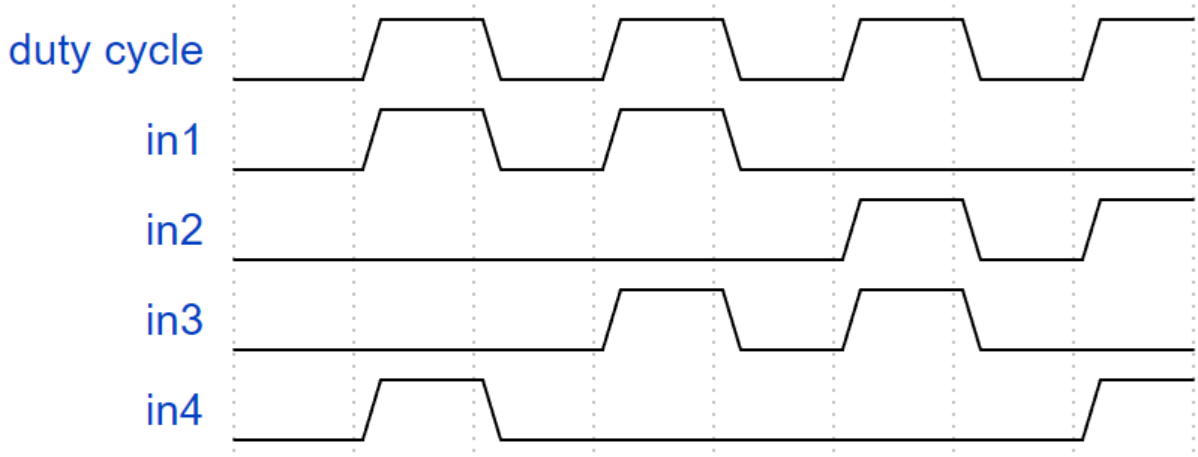
**ШИМ** - метод уменьшения средней мощности, передаваемой электрическим сигналом, путем эффективного разделения его на отдельные части. Среднее значение напряжения (и тока), подаваемого на нагрузку, регулируется быстрым включением и выключением переключателя между питанием и нагрузкой. Чем дольше переключатель включен по сравнению с периодами выключения, тем выше общая мощность, подаваемая на нагрузку.

```
1   pub fn set_frequency(&self, frequency: f64, duty_cycle: f64) -> Result<()>
```

- frequency - частота
- duty cycle - скважность, задается числом с плавающей точкой в промежутке между 0.0 и 1.0.

В нашей программе изначально значение скважности установлено на 0, что означает что на двигатели будет подаваться в среднем 0V. Никакого движения не происходит.

При нажатии на конпку мы подаем сигналы на левые и правые двигатели и устанавливаем скважность на 1, что означает что на двигатели будет подаваться в среднем 5V.

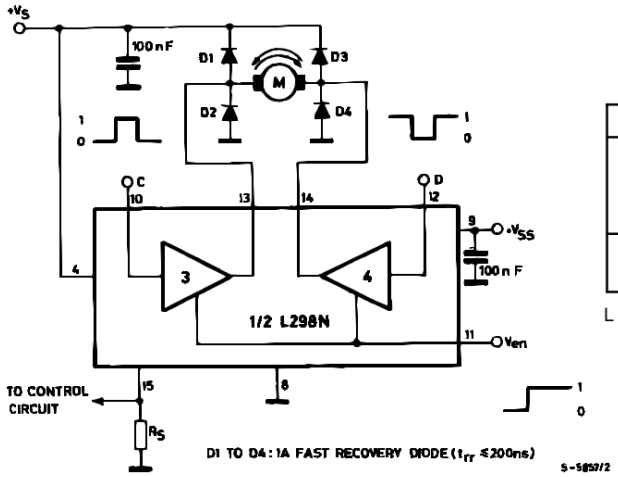



### 1.4 Питание

1. L289N - Логическое устройство - 5V
2. L289N - Моторы - 12V
3. Кнопка - 3.3V
4. Raspberry pi 2b+ - 5V

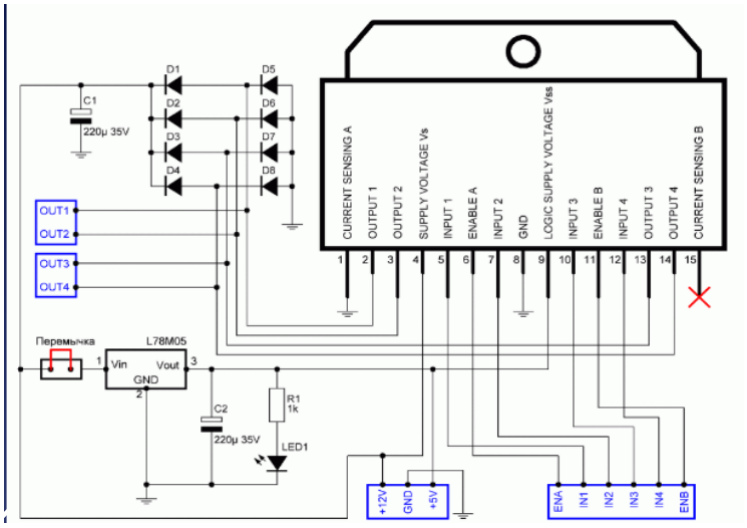
## 2 Схемы

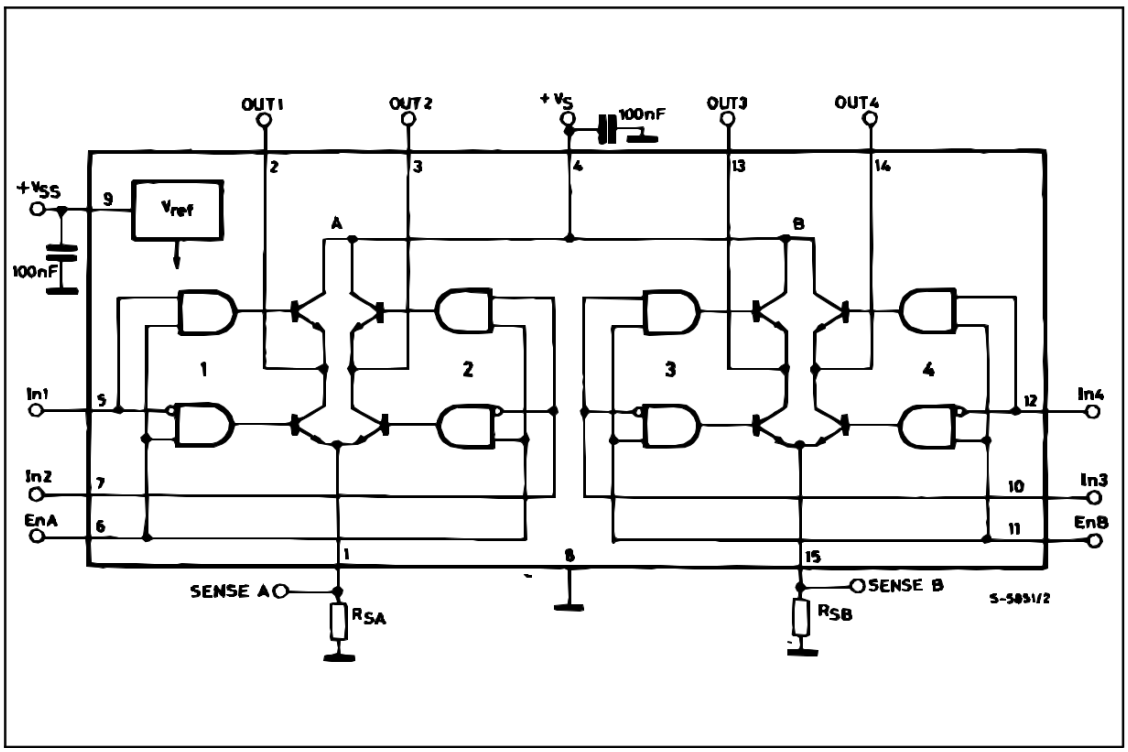
### 2.1 Принципиальная схема драйвера L289N



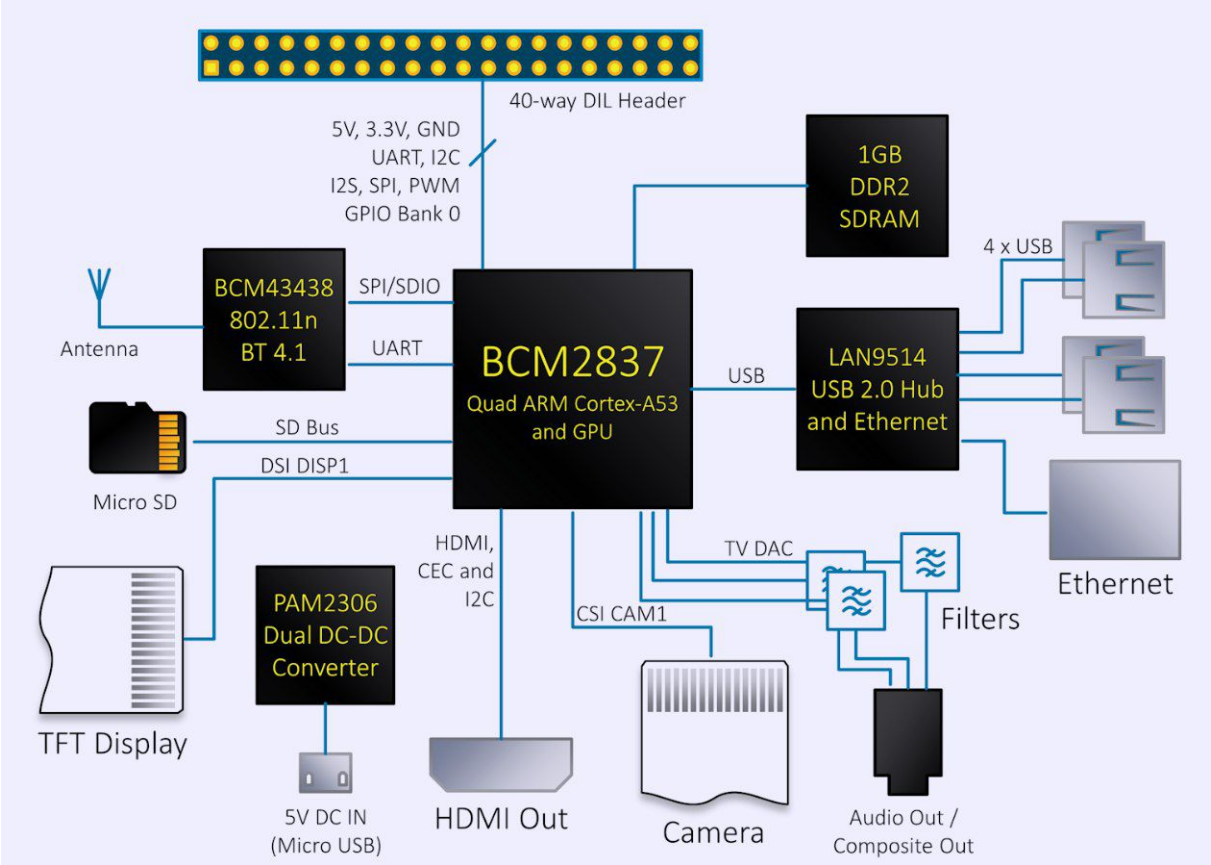

| Inputs | | Function |
|---|---|---|
| $V_{en}$ = H | C = H ; D = L | Forward |
| | C = L ; D = H | Reverse |
| | C = D | Fast Motor Stop |
| $V_{en}$ = L | C = X ; D = X | Free Running Motor Stop |

L = Low H = High X = Don't care


D1
D2
D3
D4
D5
D6
D7
D8
C1
220µ 35V
OUT1
OUT2
OUT3
OUT4
Перемычка
L78M05
Vin
Vout
GND
C2
220µ 35V
R1
1k
LED1
+12V
GND
+5V
ENA
IN1
IN2
IN3
IN4
ENB
CURRENT SENSING A
OUTPUT 1
OUTPUT 2
SUPPLY VOLTAGE Vs
INPUT 1
ENABLE A
INPUT 2
GND
LOGIC SUPPLY VOLTAGE Vss
INPUT 3
ENABLE B
INPUT 4
OUTPUT 3
OUTPUT 4
CURRENT SENSING B


OUT1
OUT2
+Vs
100nF
OUT3
OUT4
+Vss
Vref
100nF
A
B
1
2
3
4
In1
In2
EnA
In4
In3
EnB
SENSE A
SENSE B
RSA
RSB
S-5853/2

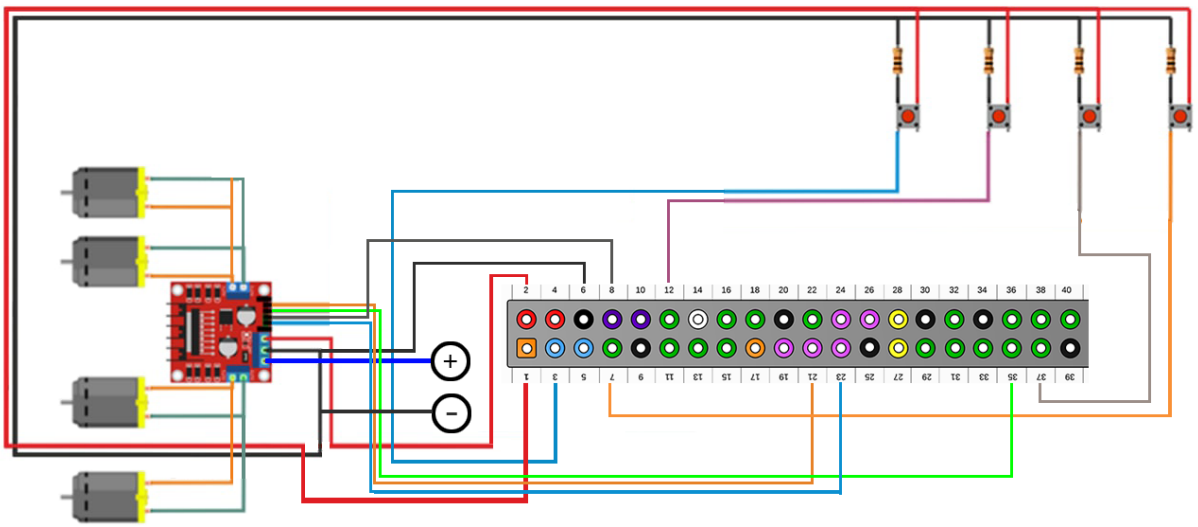
## 2.2 Принципиальная схема Raspberry Pi

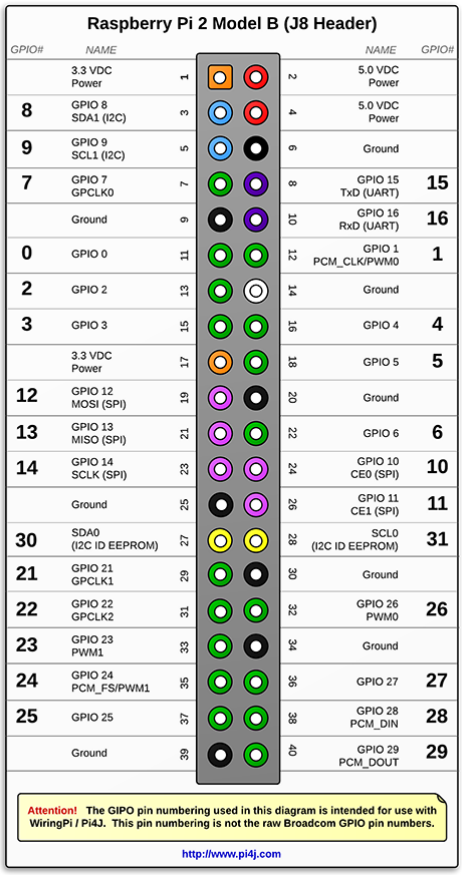



Полную принципиальную схему можно найти здесь.

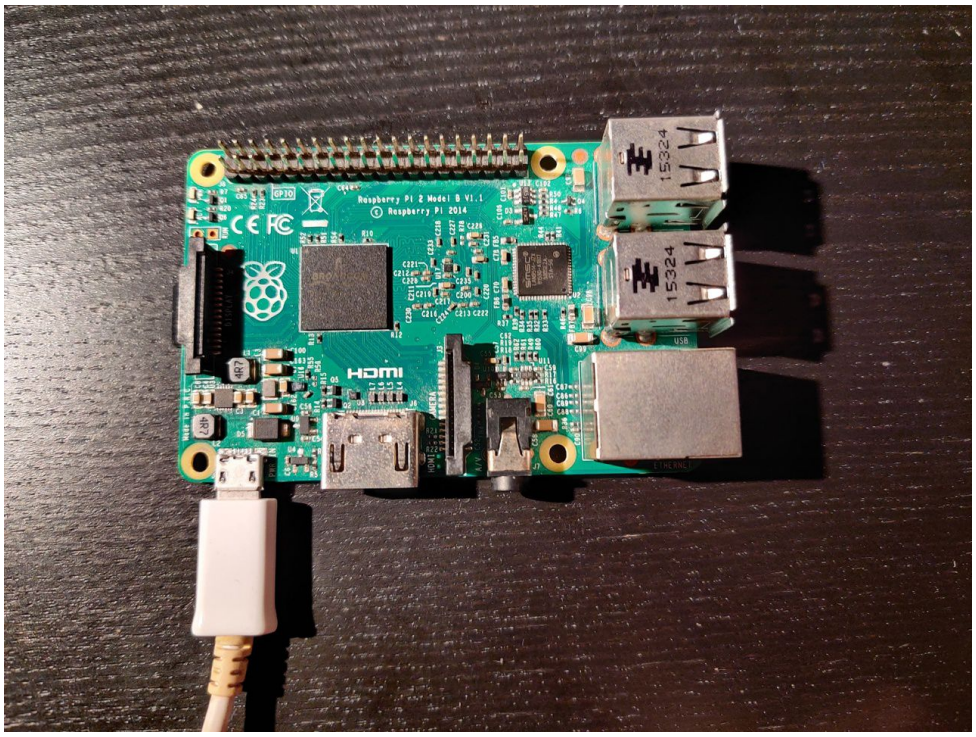
## 2.3 Схема подключений

Пин №1 подает 3.3 V на кнопки. Пин №2 подает 5 V питания на логическое устройство. Моторы питаются из внешнего зарядного устройства.






Raspberry Pi питает от micro-USB, который подает 5V.



# 3 Rust

Для написания программы нами был выбран молодой язык программирования Rust. Но почему мы его выбрали?

**Производительность** Rust невероятно быстр и эффективно использует память: без среды выполнения или сборщика мусора он может поддерживать критически важные для производительности службы, работать на встроенных устройствах и легко интегрироваться с другими языками.

**Надежность** Богатая система типов и модель владения Rust гарантируют безопасность памяти и потокобезопасность, что позволяет устранять многие классы ошибок во время компиляции.

**Удобство** У Rust отличная документация, удобный компилятор с полезными сообщениями об ошибках и первоклассные инструменты — интегрированный менеджер пакетов и инструмент сборки, интеллектуальная поддержка нескольких редакторов с автодополнением и проверкой типов, автоматический форматировщик и многое другое.

**Первая Версия** Первый стабильный релиз, Rust 1.0, был анонсирован в мае 2015.

**Владение** Владение — это набор правил, определяющих, как программа на Rust управляет памятью. Все программы так или иначе должны использовать память компьютера во время работы. В некоторых языках есть сборщики мусора, которые регулярно отслеживают неиспользуемую память во время работы программы; в других программист должен память явно выделять и освобождать. В Rust используется третий подход: управление памятью происходит через систему владения с набором правил, которые проверяются компилятором. При нарушении любого из правил программа не будет скомпилирована.

**правило 1** У каждого значения в Rust есть владелец

**правило 2** У значения может быть только один владелец в один момент времени

**правило 3** Когда владелец покидает область видимости, значение удаляется

**Многопоточность** Безопасное и эффективное управление многопоточным программированием — ещё одна из основных целей Rust. Поэтому он обеспечивает решение с наилучшей производительностью в любой конкретной ситуации и будут иметь меньше абстракций по сравнению с аппаратным обеспечением. Поэтому Rust предлагает множество инструментов для моделирования проблем любым способом, который подходит для вашей ситуации и требований.

**способ 1** потоки для одновременного запуска нескольких фрагментов кода

**способ 2** передачи сообщений, где каналы передают сообщения между потоками

**способ 3** Многопоточность для совместно используемого состояния, когда несколько потоков имеют доступ к некоторому фрагменту данных

# 4 Реализация на языке Python

В рамках дополнительного задания нужно было реализовать программу с тем же функционалом на языке Python и сравнить производительность с программой на языке Rust. Наша программа достаточно маленькая и в большинстве своем зависит от пользовательского ввода, так что произвести качественное сравнение представляется слишком трудным. Поэтому для сравнения языков была написана функция, которая считает значение числа $\pi$ и сравнивает скорость работы.

Программа на языке Rust:

```rust
use std::time::Instant;

const N: u64 = 100_000_000;

fn calculate_pi(n_terms: u64) -> f64 {
    let numerator = 4.0;
    let mut denominator = 1.0;
    let mut operation = 1.0;
    let mut pi = 0.0;
    for _ in 0..n_terms {
        pi += operation * (numerator / denominator);
        denominator += 2.0;
        operation *= -1.0;
    }
    pi
}

fn main() {
    let start = Instant::now();
    let pi = calculate_pi(N);
    let duration = start.elapsed();
    println!("pi = {}", pi);
    println!("Time elapsed: {:.2} seconds!", duration.as_secs_f64());
}
```

Программа на языке Python:

```python

from time import time

N = 100_000_000

def calculate_pi(n_terms: int) -> float:
    numerator = 4.0
    denominator = 1.0
    operation = 1.0
```

```
    pi = 0.0
    for _ in range(n_terms):
        pi += operation * (numerator / denominator)
        denominator += 2.0
        operation *= -1.0
    return pi


if __name__ == "__main__":
    start = time()
    pi = calculate_pi(N)
    end = time()
    print(f"pi = {pi}")
    print(f"Time elapsed: {round((end - start), 2)} seconds!")
```

После выполнения программ, мы получаем их время работы:

```
  N = 100,000,000

  Rust: 0.12 seconds
  Python: 9.33 seconds
```

По данным результатам можно увидеть что Rust почти в 80(!) раз быстрее чем Python.

# 5 Программное обеспечение

## 5.1 Используемые инструменты

1. Rust - 1.65.0 - Язык программирования
2. rppal - 0.14.0 - Raspberry Pi Peripheral Access Library. Предоставляет доступ к GPIO у Raspberry Pi.
3. Python -3.10.1 - Язык программирования
4. Rpi.GPIO - 0.7.0 - Raspberry Pi GPIO. Предоставляет доступ к GPIO у Raspberry Pi.

## 5.2 Код программы на Rust

```
/* main.rs */
use std::thread;
use std::time::Duration;
use color_eyre::Result;

use rppal::gpio::Gpio;
use rppal::pwm::{Pwm, Channel};
use rppal::system::DeviceInfo;

mod engine;
mod input;

fn main() -> Result<()> {
    println!("Working on a {}.", DeviceInfo::new()?.model());

    let mut engine_left = engine::Engine{
        forward_pin: Gpio::new()?.get(14)?.into_output(),
        backward_pin: Gpio::new()?.get(15)?.into_output(),
        pwm: Pwm::new(Channel::Pwm0)?,
    };

    let mut engine_right = engine::Engine{
        forward_pin: Gpio::new()?.get(26)?.into_output(),
        backward_pin: Gpio::new()?.get(13)?.into_output(),
        pwm: Pwm::new(Channel::Pwm1)?,
    };
    let mut ddc = engine::DigitalDirectionController{
        left: engine_left,
        right: engine_right,
    };
    let mut input = input::ButtonInput{
```

```
        forward: Gpio::new()?.get(8)?.into_input(),
        backward: Gpio::new()?.get(1)?.into_input(),
        left: Gpio::new()?.get(25)?.into_input(),
        right: Gpio::new()?.get(7)?.into_input(),
    };

    loop {
        let direction = input.get_direction();
        println!("Direction: {:?}", direction);
        ddc.direction(direction)?;
        thread::sleep(Duration::from_millis(50));
    }
}
```

```
/* input.rs */
use color_eyre::Result;
use rppal::gpio::InputPin;

use super::engine::Direction;

pub struct ButtonInput {
    pub forward: InputPin,
    pub backward: InputPin,
    pub left: InputPin,
    pub right: InputPin,
}

impl ButtonInput {
    pub fn get_direction(&mut self) -> Direction {
        if self.forward.is_high() {
            return Direction::Forward;
        };
        if self.left.is_high() {
            return Direction::Left;
        };
        if self.right.is_high() {
            return Direction::Right;
        };
        if self.backward.is_high() {
            return Direction::Backward;
        };
        Direction::Stop
    }
}
```

```rust
/* engine.rs */
use color_eyre::Result;
use rppal::gpio::OutputPin;
use rppal::pwm::Pwm;

pub struct Engine {
    pub forward_pin: OutputPin,
    pub backward_pin: OutputPin,
    pub pwm: Pwm,
}

impl Engine {
    pub fn set(&mut self, direction: f64) -> Result<()> {
        if !self.pwm.is_enabled()? {

            self.pwm.set_frequency(20.0, 0.0)?;
            self.pwm.enable()?;
        }
        let direction = direction.clamp(-1.0, 1.0);
        if direction >= 0.0 {
            self.forward_pin.set_high();
            self.backward_pin.set_low();
        } else {
            self.forward_pin.set_low();
            self.backward_pin.set_high();
        }
        self.pwm.set_duty_cycle(direction.abs())?;
        Ok(())
    }
}

impl Drop for Engine {
    fn drop(&mut self) {
        self.forward_pin.set_low();
        self.backward_pin.set_low();
    }
}

pub struct AnalogDirectionController {
    pub left: Engine,
    pub right: Engine,
}

impl AnalogDirectionController {
    pub fn direction(&mut self, gradient: f64, radial: f64) -> Result<()> {
        let gradient = gradient.clamp(-1.0, 1.0);
```

```
47          let radial = radial.clamp(-1.0, 1.0);
48          Ok(())
49      }
50  }
51
52  #[derive(Debug)]
53  pub enum Direction {
54      Stop,
55      Forward,
56      Backward,
57      Left,
58      Right
59  }
60
61  pub struct DigitalDirectionController {
62      pub left: Engine,
63      pub right: Engine,
64  }
65
66  impl DigitalDirectionController {
67      pub fn direction(&mut self, direction: Direction) -> Result<()> {
68          match direction {
69              Direction::Stop => {
70                  self.left.set(0.0)?;
71                  self.right.set(0.0)
72              },
73              Direction::Forward => {
74                  self.left.set(1.0)?;
75                  self.right.set(1.0)
76              },
77              Direction::Backward => {
78                  self.left.set(-1.0)?;
79                  self.right.set(-1.0)
80              },
81              Direction::Left => {
82                  self.left.set(-1.0)?;
83                  self.right.set(1.0)
84              },
85              Direction::Right => {
86                  self.left.set(1.0)?;
87                  self.right.set(-1.0)
88              },
89          }
90      }
91  }
```

## 5.3 Код программы на Python

```python
# main.py
import RPi.GPIO as GPIO
import engine
import input
import time

def main():
    GPIO.setmode(GPIO.BCM)
    engine_left = engine.Engine(
        forward_pin = GPIO.setup(14, GPIO.OUT),
        backward_pin= GPIO.setup(15, GPIO.OUT),
        pwm = GPIO.PWM(18, 20),
    )
    engine_left.pwm.Start(0)

    engine_right = engine.Engine(
        forward_pin = GPIO.setup(26, GPIO.OUT),
        backward_pin = GPIO.setup(13, GPIO.OUT),
        pwm = GPIO.PWM(19, 20)
    )
    engine_right.pwm.Start(0)

    ddc = engine.DigitalDirectionController(
        left = engine_left,
        right = engine_right
    )

    inp = input.ButtonInput(
        forward = GPIO.setup(8, GPIO.IN, pull_up_down=GPIO.PUD_UP),
        backward = GPIO.setup(1, GPIO.IN, pull_up_down=GPIO.PUD_UP),
        left = GPIO.setup(25, GPIO.IN, pull_up_down=GPIO.PUD_UP),
        right = GPIO.setup(7, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    )

    while True:
        direction = inp.get_direction()
        print(f"Direction: {direction}")
        ddc.direction(direction)
        time.sleep(0.050)

if __name__ == "__main__":
    main()
```

```python
# input.py
from dataclasses import dataclass
import RPi.GPIO as GPIO
from engine import Direction

@dataclass
class ButtonInput:
  forward: any
  backward: any
  left: any
  right: any

  def get_direction(self) -> Direction:
    if GPIO.input(self.forward):
      return Direction.Forward
    if GPIO.input(self.left):
      return Direction.Left
    if GPIO.input(self.right):
      return Direction.Right
    if GPIO.input(self.backward):
      return Direction.Backward
    return Direction.Stop
```

```python
# engine.py
from dataclasses import dataclass
from enum import Enum
import RPi.GPIO as GPIO

@dataclass
class Engine:
  forward_pin: any
  backward_pin: any
  pwm: GPIO.PWM

  def set(self, direction: any):
    if direction >= 0.0:
      GPIO.output(self.forward_pin, GPIO.HIGH)
      GPIO.output(self.backward_pin, GPIO.LOW)
    else:
      GPIO.output(self.forward_pin, GPIO.LOW)
      GPIO.output(self.backward_pin, GPIO.HIGH)
    self.pwm.ChangeDutyCycle(abs(direction))

  def drop(self):
    GPIO.output(self.forward_pin, GPIO.LOW)
```

```
23      GPIO.output(self.backward_pin, GPIO.LOW)
24
25  class Direction(Enum):
26    Stop = 1,
27    Forward = 2,
28    Backward = 3,
29    Left = 4,
30    Right = 5
31
32  @dataclass
33  class DigitalDirectionController:
34    left: Engine
35    right: Engine
36
37    def direction(self, direction: Direction):
38      match direction:
39        case Direction.Stop:
40          self.left.set(0)
41          self.right.set(0)
42        case Direction.Forward:
43          self.left.set(1)
44          self.right.set(1)
45        case Direction.Backward:
46          self.left.set(-1)
47          self.left.set(-1)
48        case Direction.Left:
49          self.left.set(-1)
50          self.left.set(1)
51        case Direction.Right:
52          self.left.set(1)
53          self.left.set(-1)
```