Десятое практическое занятие

Командная разработка

Введение

Сделаем небольшой перерыв!

(Ну, в смысле, это я сделаю :))

А Вы - поработаете в команде!

Сегодня не будет полностью разобранного материала, но будет набор небольших и интересных заданий, для решения которых пригодится как материал изученный ранее в рамках этого курса, так и ему сопутствующий.

В условиях ограниченного времени, в одиночку все это успеть решить сложно, но, если Вы сумеете договориться между собой, грамотно разделить задачу на отдельные элементы, будете работать вместе и постараетесь - то все получится!

Объект работы

Сегодня мы познакомимся с новым датчиком - ультразвуковым дальномером HC-SR04

Это популярный и недорогой датчик, используемый для измерения расстояния до объектов.

Он работает на основе принципа эхолокации: датчик излучает ультразвуковые волны и измеряет время, за которое они возвращаются, отразившись от объекта. На основе этого времени рассчитывается расстояние до объекта.

Основные характеристики HC-SR04:

- **Рабочее напряжение:** 5 В.
- **Рабочий ток:** 15 мА.
- **Измеряемое расстояние:** от 2 см до 400 см.
- **Точность измерения:** около 3 мм.
- **Угол обзора:** примерно 15 градусов.
- **Частота ультразвука:** 40 кГц.

Преимущества HC-SR04:

- Низкая стоимость.
- Простота подключения и использования.
- Достаточная точность для большинства задач.

Недостатки:

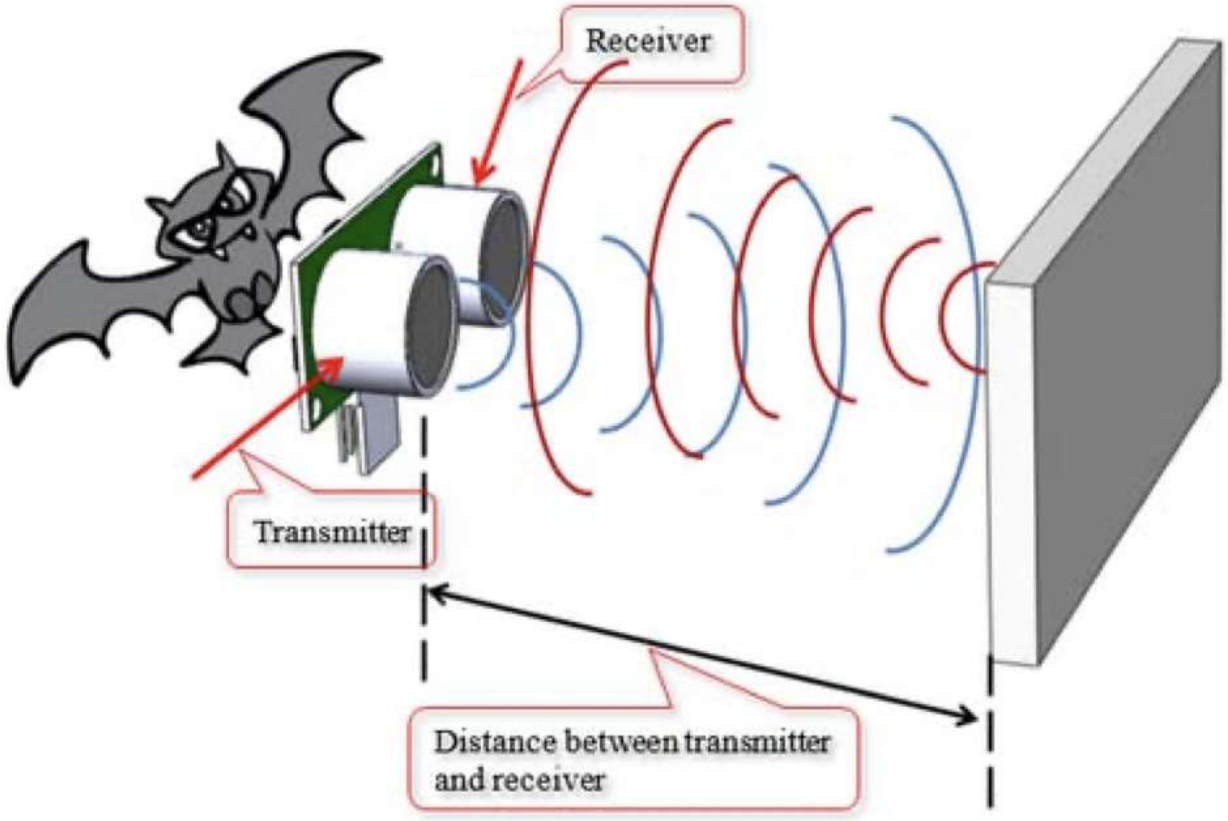
- Ограниченный диапазон измерения (до 4 метров).
- Зависимость от температуры и влажности воздуха (скорость звука меняется).
- Неэффективен для измерения расстояния до мягких или пористых объектов, которые плохо отражают ультразвук.

Применение:

HC-SR04 широко используется в робототехнике, системах автоматизации, умных устройствах и проектах, где требуется измерение расстояния без контакта с объектом. Например:

- Роботы, избегающие препятствий.
- Системы парковки.
- Измерительные устройства.

Принцип работы:



1. Датчик излучает ультразвуковой импульс (8 циклов на частоте 40 кГц) по сигналу на выводе Trig.
2. Импульс отражается от объекта и возвращается к датчику.
3. Датчик фиксирует время, за которое импульс вернулся, и выдает сигнал на выводе Echo.
4. Расстояние до объекта рассчитывается по формуле:
   Расстояние = (Время * Скорость звука) / 2
   Скорость звука в воздухе при комнатной температуре (20°C) составляет примерно 343 м/с.

Подключение к микроконтроллеру:

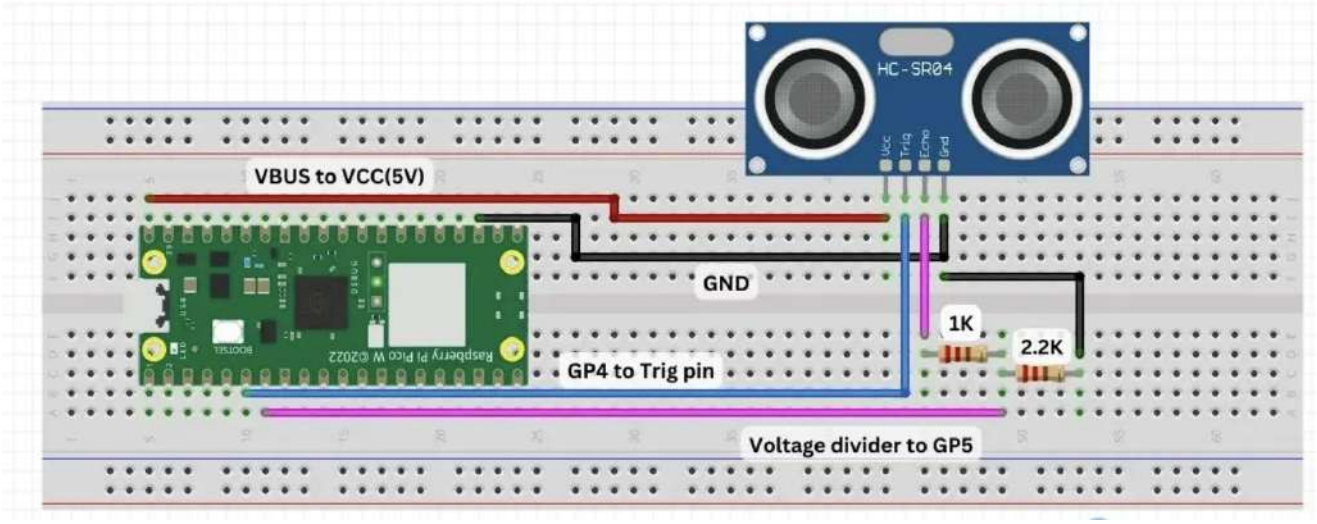
HC-SR04 имеет 4 вывода:

- **VCC:** Подключение к питанию 5 В.
- **GND:** Подключение к земле.
- **Trig:** Вход для запуска измерения (подключается к цифровому выводу микроконтроллера).
- **Echo:** Выход, на котором формируется сигнал с длительностью, пропорциональной расстоянию (подключается к цифровому выводу микроконтроллера).

Вроде ничего сложного, но есть один важный нюанс:

⚠ Так как датчик HC-SR04 работает от напряжения 5V, а пины RP Pico от 3.3V сигнал, считываемый с ECHO пина сигнал необходимо провести через делитель напряжения!

ⓘ Что такое делитель напряжения, мы подробно разбирали на третьем практическом занятии

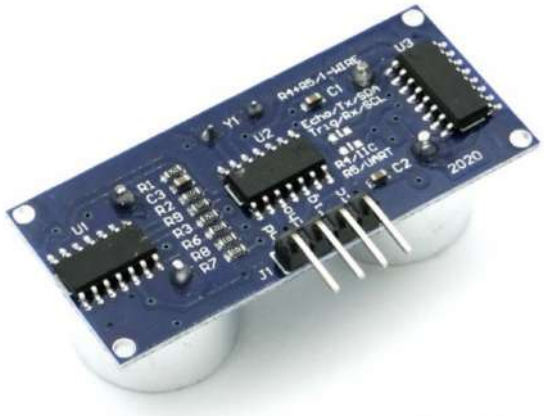


Пример расчета необходимых резисторов:

$$\frac{2.2k\Omega}{2.2k\Omega + 1k\Omega} \times 5V = 3.4V$$

Управление HC-SR04

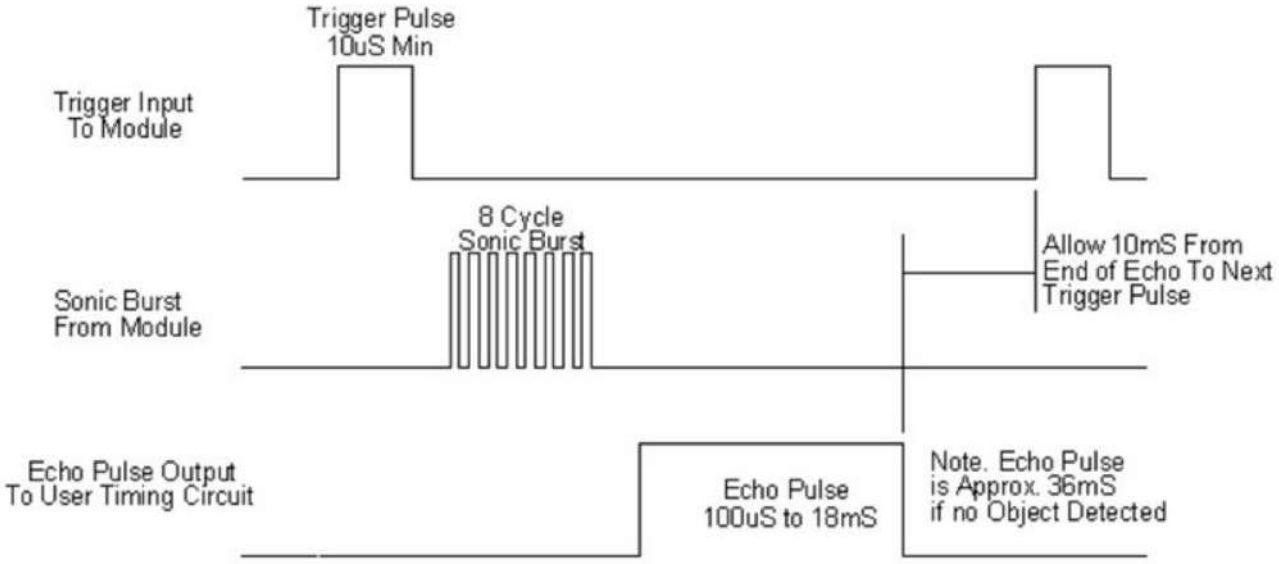
Взглянем на тыльную сторону датчика:



и применив несложную дедукцию придем к выводу, что все эти электронные компоненты на плате не для красоты...

Сам датчик берет на себя часть вспомогательных операций по генерации импульсов и сигналов для возможности точных измерений.

Принципиальная схема сигналов датчика представлена на рисунке:



Разберем ее!

- Для "включения" датчика необходимо подать на Trigger пин высокий сигнал длительностью в 10 микросекунд ($10uS$);
- Далее микроконтроллер датчика сгенерирует звуковой импульс из 8 коротких ультразвуковых "писков";
- Echo пин перейдет в режим "прослушивания" - он начнет передавать высокий логический сигнал на внешний микроконтроллер, который упадет в момент фиксации датчиком отразившегося от препятствия звука.

Примечания:

- Максимальное время ожидания отклика на Echo пине составляет 36 миллисекунд, после чего сигнал будет автоматически "опущен" независимо от того был ли зафиксирован вернувшийся звуковой сигнал. Это время определяет максимальную дальность измерений датчика.
- Следующее измерение становится возможным к запуску после прошествия 10 миллисекунд от последнего измерения.

Таким образом, продолжительность высокого сигнала на Echo пине после запуска измерения определяет время прохода звуковой волны от датчика до препятствия и обратно.

Приняв скорость звука как 343 м/с, определяемое расстояние в сантиметрах можно рассчитать по формуле:

$$S = \frac{1}{2}(pulse\_duration * 0.0343)\ см$$

Обобщим вышеизложенное, представив все эти операции в виде кода:

Базовый (плохой) код для работы с датчиком HC-SR04

```
from time import ticks_us, sleep_us, ticks_diff
from machine import Pin

TRIG_PIN = 4
ECHO_PIN = 5

trigger = Pin(TRIG_PIN, Pin.OUT)
echo = Pin(ECHO_PIN, Pin.IN)

def get_distance():
    trigger.low()
    sleep_us(2)
    trigger.high()
    sleep_us(10)
    trigger.low()
    signal_off = ticks_us()
    signal_on = ticks_us()
    while echo.value() == 0:
        signal_off = ticks_us()
    while echo.value() == 1:
        signal_on = ticks_us()
    time_passed = ticks_diff(signal_on, signal_off)
    distance_cm = (time_passed * 0.0343) / 2
    return round(distance_cm, 2)

while True:
    distance_cm = get_distance()
    print(f"{distance_cm} cm")
    sleep_us(500_000)
```

Разберем его:

1. Как всегда, начинаем с подключения необходимых пакетов:

```
from time import ticks_us, sleep_us, ticks_diff
from machine import Pin
```

С классом `Pin` все должно быть понятно, а вот с функции `ticks_us, sleep_us, ticks_diff` из модуля `time` заслуживают пары комментариев:

- `sleep_us()` - "усыпляет" микроконтроллер на переданное в нее количество миллисекунд (все недостатки усыпления микроконтроллера нами уже не раз обсуждались, но, так как время необходимого "сна" в режиме измерений крайне мало, используем эту функцию как допустимое зло);
- `ticks_us()` - возвращает текущее количество "тиков" - текущее значение счетчика времени в микросекундах.

Счетчик времени в `MicroPython` на `RP2040` (чип Raspberry Pi Pico) является **32-битным беззнаковым целым числом** (`uint32_t`). Это означает, что максимальное значение, которое может вернуть `ticks_us()`, составляет:

$$2^{32} - 1 = 4,294,967,295 \ микросекунд.$$

Когда счетчик достигает максимального значения (4,294,967,295 мкс), он переполняется и сбрасывается в 0. Это поведение называется **переполнением счетчика** (overflow). Переполнение происходит через:

$$4,294,967,295 \ мкс \approx 4295 \ секунд \approx 71.58 \ минут.$$

Таким образом, счетчик переполняется примерно каждые **71 минуту**.

Хотя в рамках продолжительности наших занятий риск появления проблемы переполнения переменных счетчиков времени невелик, для вычисления временных интервалов корректнее использовать функцию:

- `ticks_diff(end, start)` - которая вычисляет разницу между двумя значениями времени, учитывая переполнение.

2. Далее инициируем пины в нужных нам режимах:

```
TRIG_PIN = 4
ECHO_PIN = 5

trigger = Pin(TRIG_PIN, Pin.OUT)
echo = Pin(ECHO_PIN, Pin.IN)
```

Тут нет никаких проблем - самые базовые режимы работы цифровых пинов:

- trigger пин должен передавать сигнал включения на дальномер, поэтому работает "наружу";
- echo пин принимает сигнал от дальномера, поэтому работает на "чтение" сигнала.

3. Логика измерения расстояния заключена в функции:

```
def get_distance():
    trigger.low()
    sleep_us(2)
    trigger.high()
    sleep_us(10)
    trigger.low()
    signal_off = ticks_us()
    signal_on = ticks_us()
    while echo.value() == 0:
        signal_off = ticks_us()
    while echo.value() == 1:
```

```
        signal_on = ticks_us()
    time_passed = ticks_diff(signal_on, signal_off)
    distance_cm = (time_passed * 0.0343) / 2
    return round(distance_cm, 2)
```

Внутри которой сначала посылается сигнал на запуск измерения:

```
trigger.low()
sleep_us(2)
trigger.high()
sleep_us(10)
trigger.low()
```

далее идет определение границ высокого сигнала на echo пине:

```
signal_off = ticks_us()
signal_on = ticks_us()
while echo.value() == 0:
    signal_off = ticks_us()
while echo.value() == 1:
    signal_on = ticks_us()
```

после чего вычисляется время отклика:

```
time_passed = ticks_diff(signal_on, signal_off)
```

вычисляется расстояние:

```
distance_cm = (time_passed * 0.0343) / 2
```

и в заключение возвращается значение округленное до второго знака после запятой.

4. Запуск непосредственных измерений в цикле:

```
while True:
    distance_cm = get_distance()
    print(f"{distance_cm} cm")
    sleep_us(500_000)
```

ЗАДАНИЕ

И вот мы наконец дошли до непосредственных заданий!

Что же нужно сделать? Ну, во-первых, заинтересоваться!

Как проверить, что эта часть задания выполнена? Это легко - в голове должны возникнуть вопросы и идеи как это все можно использовать, проверить, улучшить...

- Возможно, кому-то будет интересен сам датчик и его метрологические свойства:
  - С какой точностью этот датчик измеряет расстояния?
  - Зависит ли его погрешность от величины измеряемой дальности?
  - А от угла к поверхности измерений?
  - А могу ли я количественно оценить и представить полученные закономерности?
  - А если через `matplotlib` или `seaborn`?
- Другие захотят исправить и улучшить разобранный ранее код:
  - исправить функцию применив к ней **принцип единственной ответственности**;
  - Усовершенствовать логику измерения временного интервала на echo пине;
  - Или вообще разработать свой класс `HcSr04` для работы с дальномером!
- Третьи решат повысить точность измерений, использовав свои знания теории математической обработки измерений:
  - Усредняя набор из нескольких данных;
  - Отфильтровывая грубые ошибки из измерений через допуски;
  - Или исключая грубые выбросы через **медианные фильтры** и т.п.
- Кто-то захочет включать измерения по нажатию кнопки, а кто-то соберет "парктроник" с лампочками как в машинах...
- **Постарайтесь все вместе успеть как можно больше и дерзайте!**