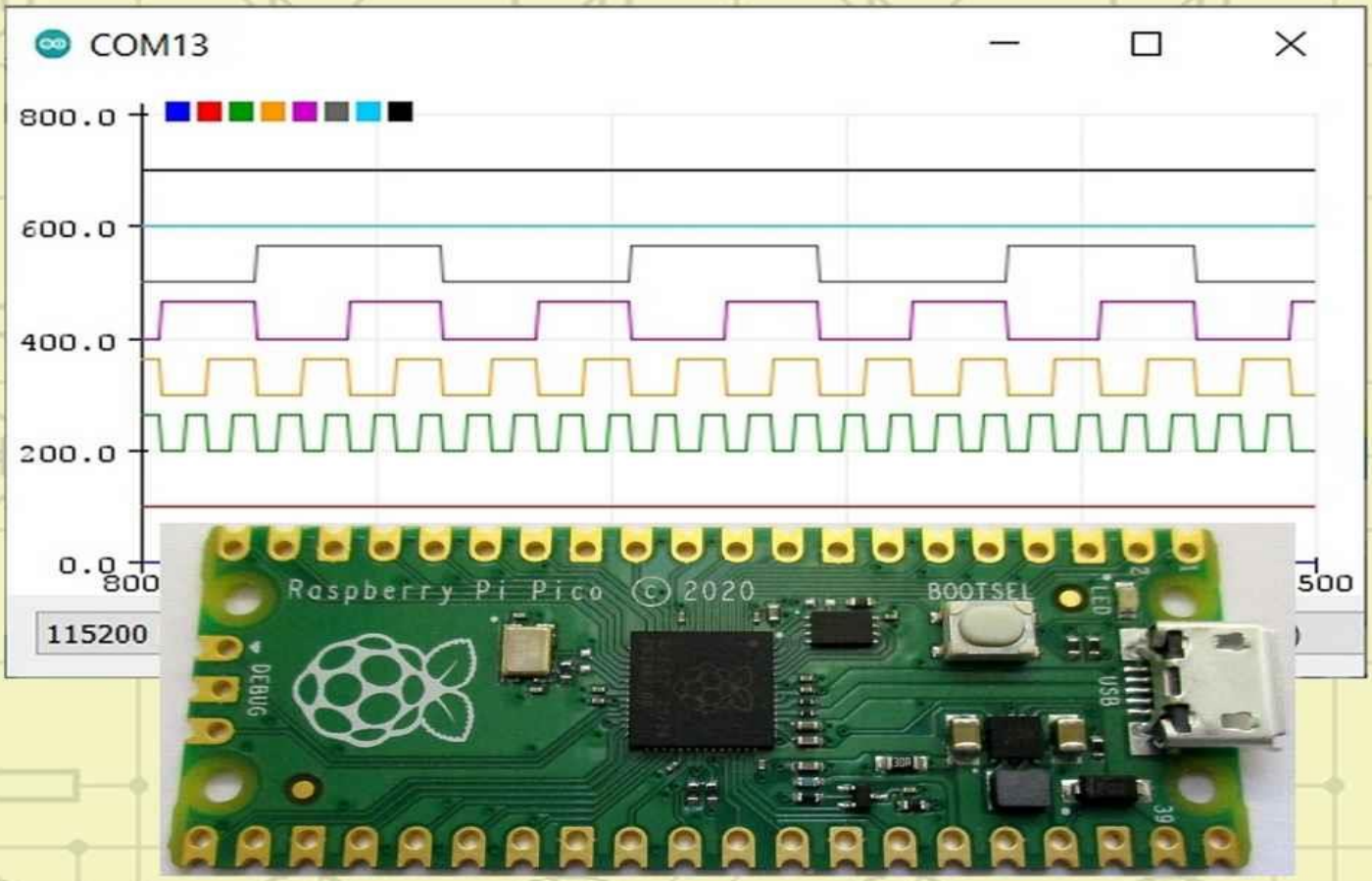


Проекты и схемы на RPi Pico

Буркхард Каинка

Проекты и схемы на RPi Pico

Буркхард Каинка

2022

Предисловие

Эта книга позволяет эффективно использовать RPi Pico в многочисленных практических приложениях. Представленные схемы раскрывают особенности Pico. Часто для выполнения задачи требуется на удивление мало внешних компонентов, если внутренние периферийные устройства RP2040 используются оптимально. Цель книги — показать самое простое и экономически эффективное решение для многих приложений. Минималистские схемы становятся полезными проектами при правильном программном обеспечении.

По сравнению с обычным 8-битным контроллером RPi Pico предлагает больше портов, большую скорость, специальные периферийные устройства и дополнительное ядро, которое можно использовать для выполнения срочных задач совершенно бесперебойно. В книге используется среда программирования Arduino, поскольку она широко доступна и проста в использовании. Вы не ограничены обычными функциями Arduino, но можете использовать все функции из комплекта разработки программного обеспечения Raspberry Pi Pico C/C++.

Оставайтесь креативными!

Буркхард Каинка

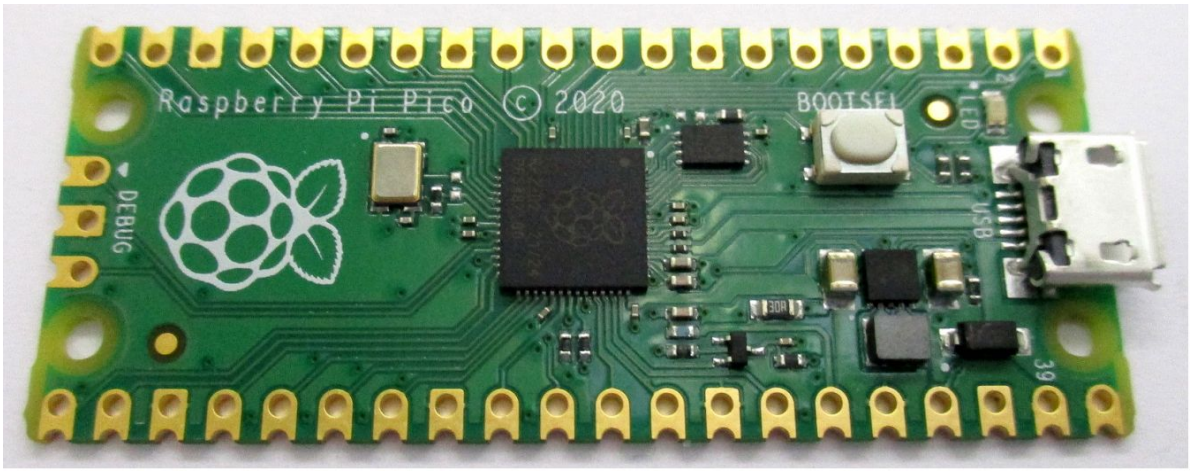
Программное обеспечение и дополнительная информация о книге:

https://www.elektronik-labor.de/Raspberry/Pico_Projects.html

СОДЕРЖАНИЕ

1 Подготовка

Устройства, описаные в книге требуют очень мало компонентов. Все,что вам действительно нужно, это Paspberry Pi Pico и кабель microUSB, и вы можете начинать прямо сейчас. Два впаянных 20-контактных разъема и макетная плата удобны для постановки простых экспериментов. Однако схемы можно собрать и пайкой.

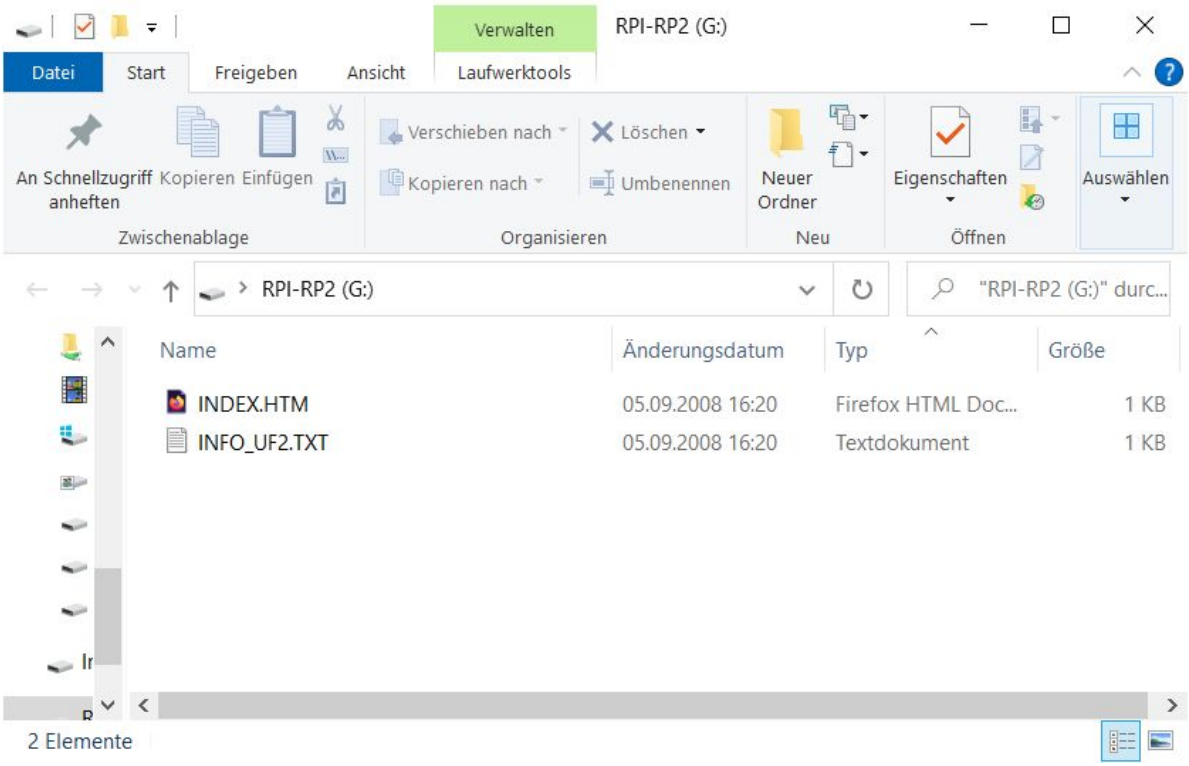


Если подключить RPi Pico к ПК с помощью USB-кабеля, сразу же запустится последняя загруженная программа. Альтернативно вы также можете использовать другой источник питания, например USB-блок питания.

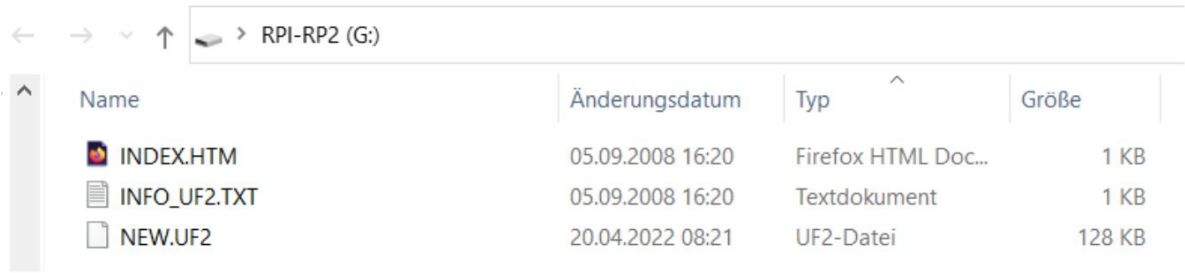
Если вы хотите загрузить новую программу, вам необходимо подключить USB-кабель, удерживая кнопку BOOT на плате Pico. Затем Pico образует диск, похожий на USB-накопитель. Внутри находится текстовый файл INFO_UF2.TXT, который можно открыть текстовым редактором:

UF2 Bootloader v1.0Model: Raspberry Pi RP2

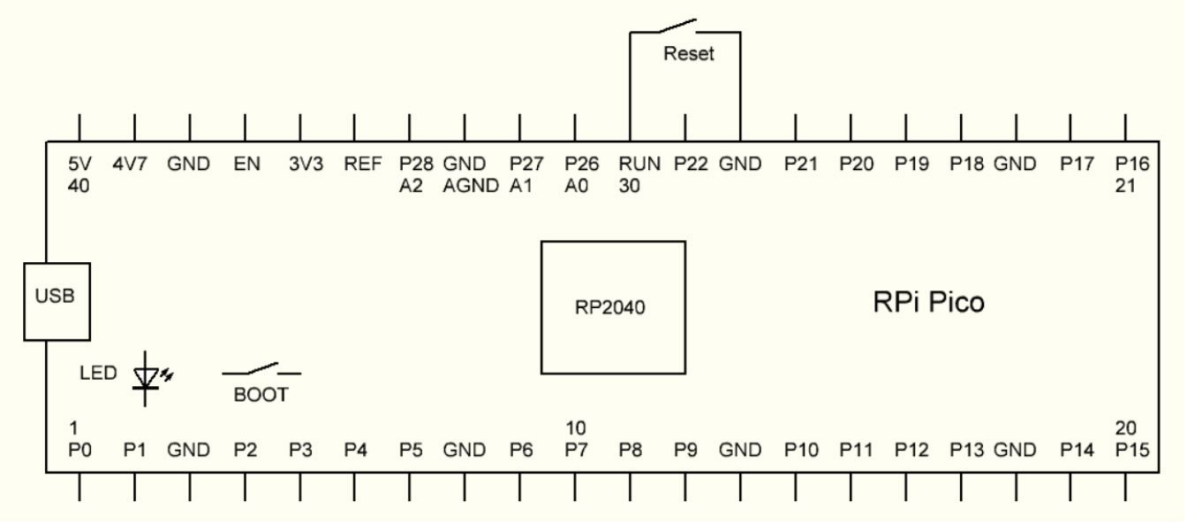
Идентификатор платы: RPI-RP2. Кроме того, существует HTML-файл, который можно открыть в браузере и который содержит ссылку на страницу Raspberry. Там вы найдете всю необходимую информацию:
https://raspberrypi.com/device/RP2



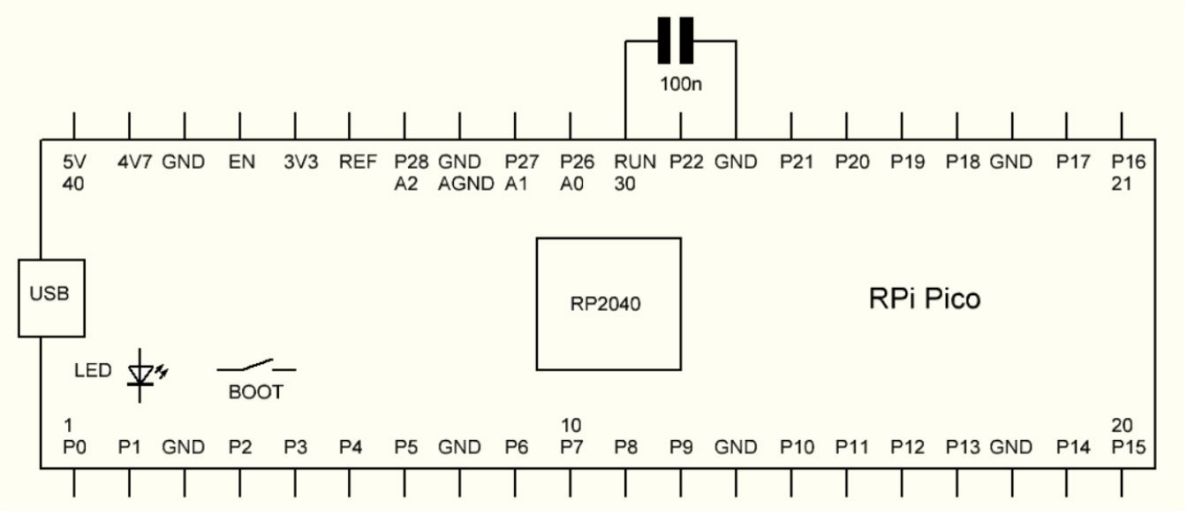
Если вы хотите загрузить новую программу, вам достаточно скопировать в каталог соответствующий файл с расширением UF2. Это происходит автоматически в Arduino IDE. Как только файл оказывается на диске, Pico Bootloader использует его и программирует сам.



Если вы пишете программу и вам часто приходится загружать новую версию, отключение USB-кабеля раздражает. Альтернативно вы можете использовать кнопку сброса, которую вы подключаете между RUN и GND. Для загрузки одновременно нажмите Reset и BOOT и отпустите Reset, удерживая кнопку Boot.



Расширение платы для Arduino IDE, представленное ниже, требует только первого ручного запуска загрузчика. После этого появляется виртуальный последовательный интерфейс, который можно использовать для остановки работающей программы и запуска загрузчика. Переключатель сброса понадобится только в случае ошибки, например, если установлен неправильный COM. Если все работает гладко, для компиляции и загрузки программы достаточно щелчка мыши.

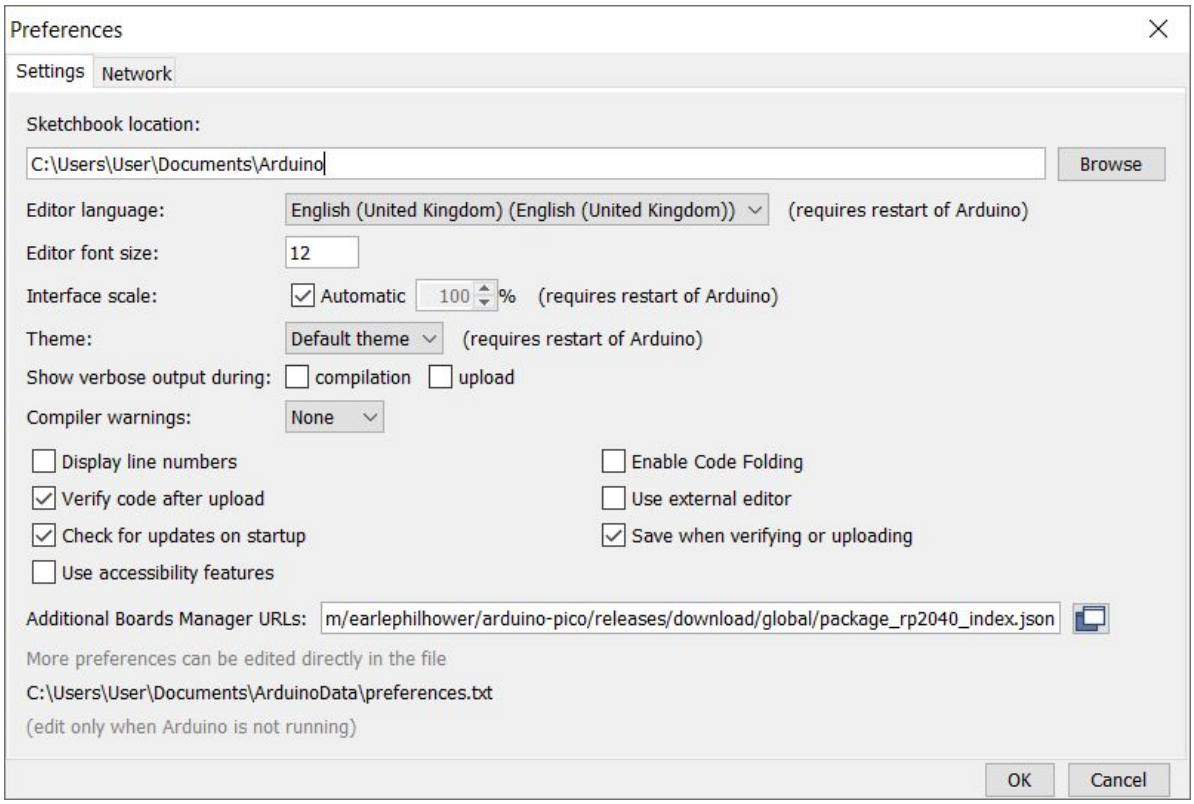


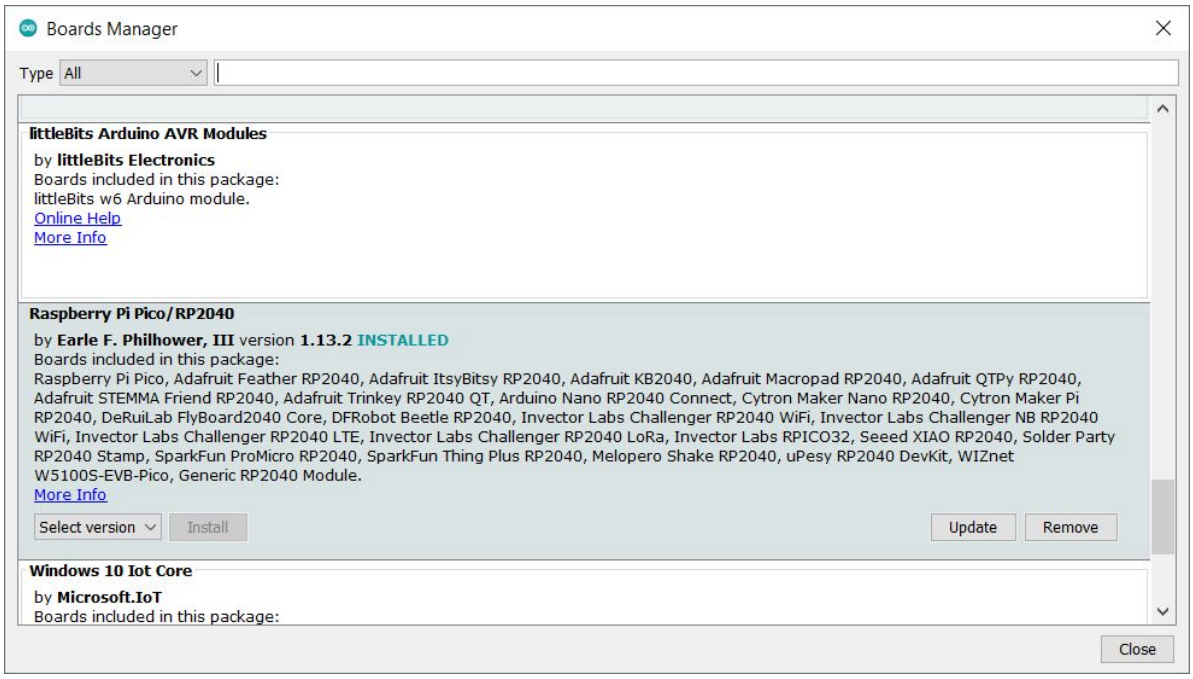
Если кнопка сброса больше не нужна, вместо нее можно вставить конденсатор емкостью 100 нФ, чтобы предотвратить случайный сброс. Если вы поэкспериментируете во время работы контроллера, случайного прикосновения может быть достаточно, чтобы вызвать небольшой помеховый импульс и, таким образом, вызвать сброс. Конденсатор предотвращает такие импульсы и делает эксплуатацию более безопасной.

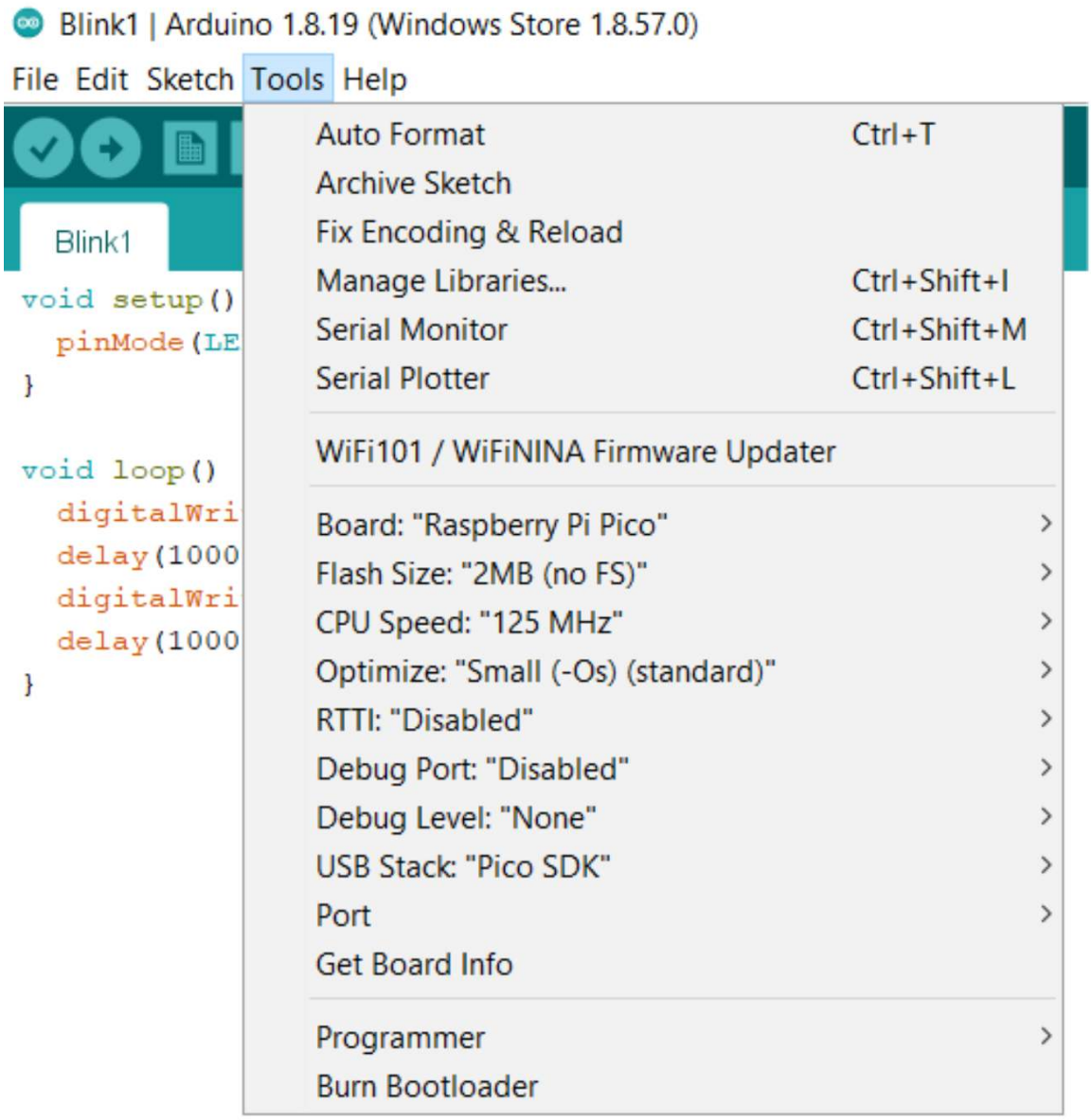
Большинство читателей, вероятно, уже работали с интерфейсом Arduino. Тогда Arduino IDE уже доступна. В противном случае последнюю версию можно загрузить с домашней страницы Arduino. Когда вы запускаете программу в первый раз, вы устанавливаете используемое оборудование в разделе «Инструменты/Плата». Помимо различных версий Arduino, часто уже установлены другие системы. Однако Пико обычно все еще отсутствует. Если вы выполните поиск в Инструменты/Плата/Диспетчер плат, вы найдете множество контроллеров, но не расширение платы Raspberry Pi Pico/RP2040 от Эрла Ф. Филхауэра, III. Это не одно из уже предложенных расширений, и его необходимо сначала сообщить в IDE. Расширение можно найти на Github:

https://github.com/earlephilhower/arduino-pico/releases/download/global/package_rp2040_index.json

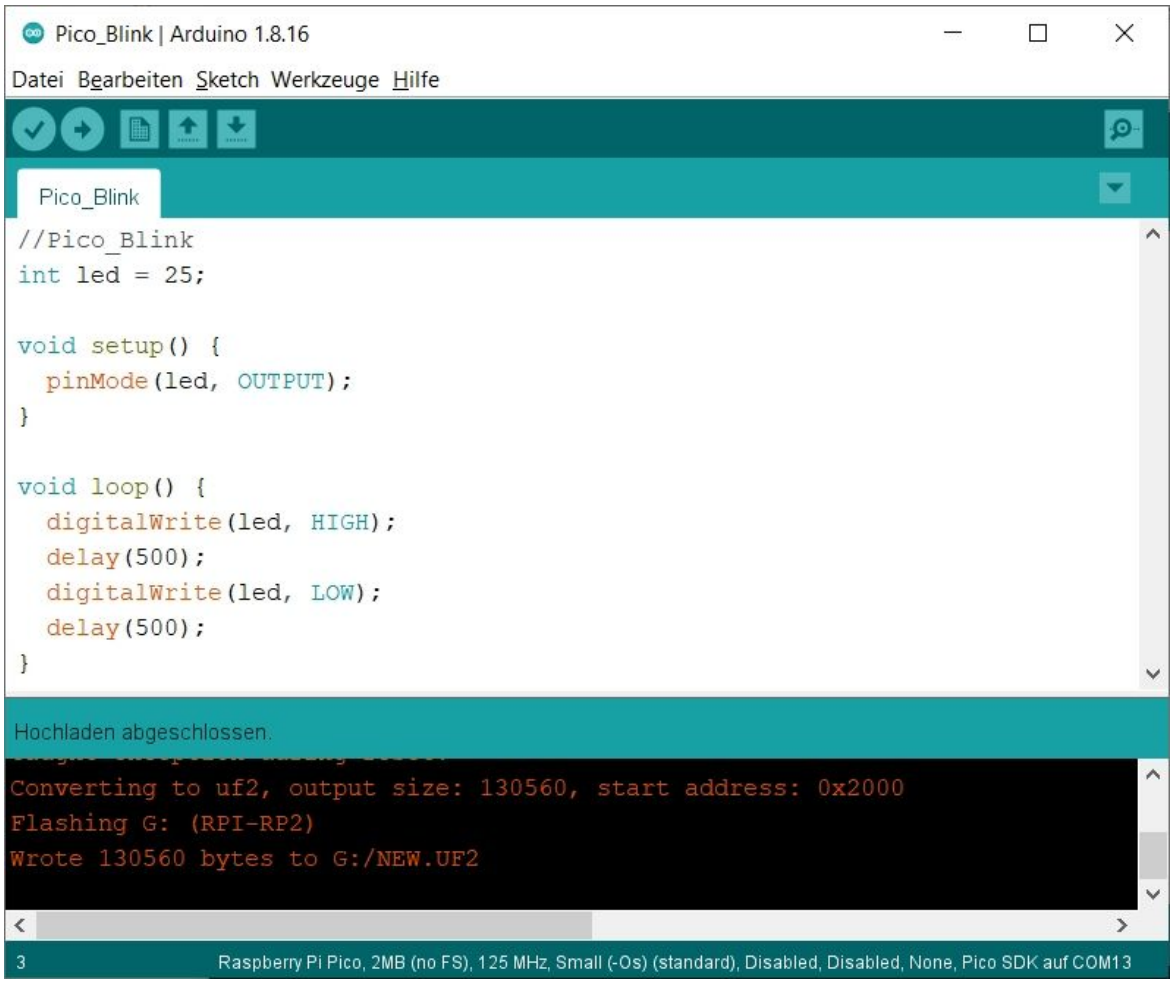
Эту ссылку необходимо ввести в разделе «Файл/Настройки». Об этом сообщает менеджер плат, который затем предлагает её для установки.


Preferences
Settings
Network
Sketchbook location:
C:\Users\User\Documents\Arduino
Browse
Editor language:
English (United Kingdom) (English (United Kingdom))
(requires restart of Arduino)
Editor font size:
12
Interface scale:
Automatic
100
%
(requires restart of Arduino)
Theme:
Default theme
(requires restart of Arduino)
Show verbose output during:
compilation
upload
Compiler warnings:
None
Display line numbers
Enable Code Folding
Verify code after upload
Use external editor
Check for updates on startup
Save when verifying or uploading
Use accessibility features
Additional Boards Manager URLs:
m/earlephilhower/arduino-pico/releases/download/global/package_rp2040_index.json
More preferences can be edited directly in the file
C:\Users\User\Documents\ArduinoData\preferences.txt
(edit only when Arduino is not running)
OK
Cancel


Boards Manager
Type
All
littleBits Arduino AVR Modules
by littleBits Electronics
Boards included in this package:
littleBits w6 Arduino module.
Online Help
More Info
Raspberry Pi Pico/RP2040
by Earle F. Philhower, III version 1.13.2 INSTALLED
Boards included in this package:
Raspberry Pi Pico, Adafruit Feather RP2040, Adafruit ItsyBitsy RP2040, Adafruit KB2040, Adafruit Macropad RP2040, Adafruit QTPy RP2040, Adafruit STEMMA Friend RP2040, Adafruit Trinkey RP2040 QT, Arduino Nano RP2040 Connect, Cytron Maker Nano RP2040, Cytron Maker Pi RP2040, DeRuiLab FlyBoard2040 Core, DFRobot Beetle RP2040, Invector Labs Challenger RP2040 WiFi, Invector Labs Challenger NB RP2040 WiFi, Invector Labs Challenger RP2040 LTE, Invector Labs Challenger RP2040 LoRa, Invector Labs RPICO32, Seeed XIAO RP2040, Solder Party RP2040 Stamp, SparkFun ProMicro RP2040, SparkFun Thing Plus RP2040, Melopero Shake RP2040, uPesy RP2040 DevKit, WIZnet W5100S-EVB-Pico, Generic RP2040 Module.
More Info
Select version
Install
Update
Remove
Windows 10 Iot Core
by Microsoft.IoT
Boards included in this package:
Close

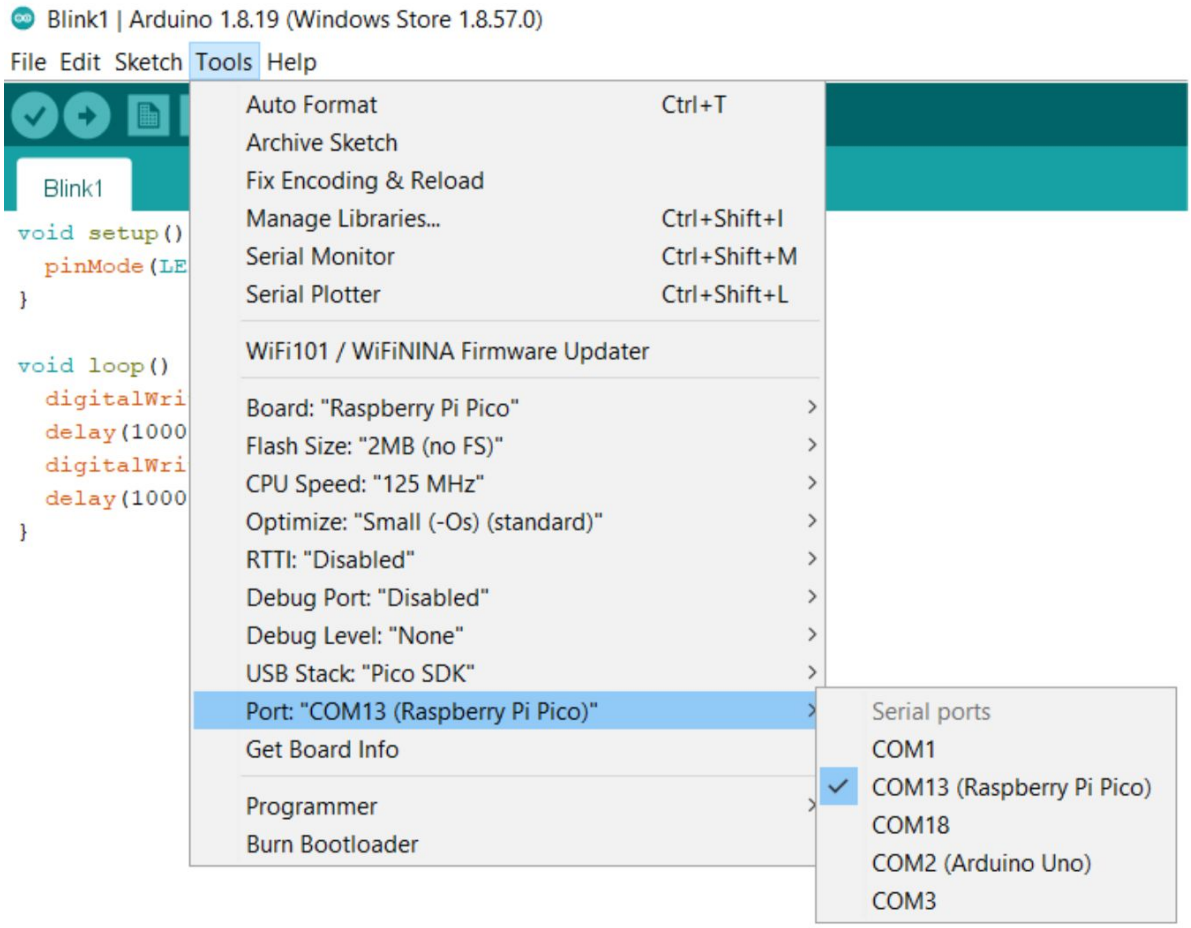


После установки Raspberry Pi Pico можно выбрать в разделе «Инструменты/Плата». Это приводит к правильным настройкам, за исключением используемого COM-порта, поскольку Pico еще не сформировал COM-интерфейс. Однако теперь уже можно загрузить первую программу: Pico_Blink.ino. Предполагается, что светодиод на плате начнет мигать. Исходный код компилируется и передается нажатием на символ стрелки (загрузить). При первой загрузке загрузчик все равно необходимо запустить вручную кнопкой BOOT и выполнить сброс или перезагрузку.



Если все работает правильно, светодиод платы теперь мигает каждую секунду. Исходный код содержит типичные для Arduino функции setup () и loop(), за которыми скрывается нечто большее, чем кажется на первый взгляд. Помимо прочего, создается виртуальный последовательный интерфейс, хотя в самой программе он пока не используется. Однако он нужен для автоматического запуска загрузчика при следующей загрузке программы.

Как только программа будет успешно запущена, вы услышите типичный звук при входе в систему USB-устройства. Теперь вы можете выбрать COM-порт пико в разделе «Инструменты/Порт». В моем случае это был COM13, но на других компьютерах обычно это другой COM.



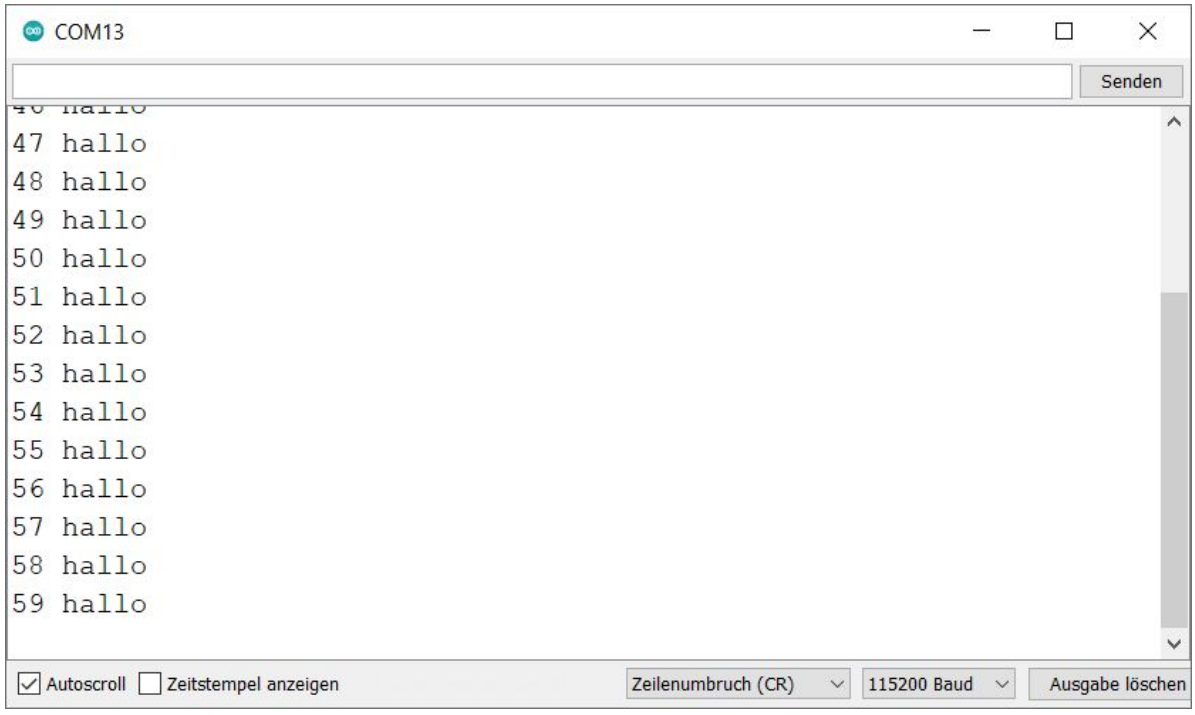
Внеся небольшое изменение в исходный код, вы теперь можете проверить, все ли работает так, как хотелось бы. Для этого измените, например, время ожидания для функций задержки. Теперь программа должна скомпилироваться и загрузиться одним щелчком мыши. Измененная скорость перепрошивки подтверждает успех.

Теперь виртуальный последовательный интерфейс также будет использоваться для отправки сообщений на ПК. Программа генерирует возрастающую последовательность чисел и текст. Оба записываются в одной строке, поскольку только вторая команда печати называется Serial.println и, таким образом, создает разрыв строки.

Последовательный интерфейс инициализирован со скоростью передачи 115200 бод. Соответственно, эта скорость также задается в последовательном мониторе. Однако это происходит только по формальным причинам, поскольку виртуальный COM-интерфейс на самом деле работает намного быстрее и работает с любыми настройками.

Это сильно отличается от Arduino, поскольку там микроконтроллер использует реальный последовательный интерфейс, где установленная скорость передачи данных должна быть правильной.

```
//Pico_Serial
int n;

void setup() {
  Serial.begin(115200);
}

void loop() {
  n++;
  Serial.print(n);
  Serial.println(" hello");
  delay(500);
}
```



Основная цель расширения заключалась в том, чтобы сделать возможным использование RPi Pico как обычного Arduino. Знакомые функции, такие как digitalWrite, digitalRead, AnalogWrite и AnalogRead, теперь также работают и на Pico.

Кроме того, однако, можно использовать и функции SDK с их значительно расширенными возможностями. Комплект разработки программного обеспечения (SDK) гораздо более обширен и сложен, чем обычное программирование Arduino. Полную документацию по всем функциям можно найти в этом документе:

raspberry-pi-pico-c-sdk.pdf

Её можно скачать со страницы Raspberry

https://raspberrypi.com/device/RP2

Новичку документация может показаться слишком обширной и сложной. Это также имело место и для меня. Я хотел решить определенные задачи и поначалу не мог понять, какие опции доступны и какие функции и настройки для них нужно использовать. Тогда помогли готовые примеры из сети. Теперь я смог найти в документации используемые там функции и постепенно правильно их классифицировать. Возможно, вы чувствуете то же самое. Вначале используйте готовые программы из книги. А там, где вы хотите что-то изменить или адаптировать для других целей, изучите документацию повнимательнее.

Если ваша цель — правильно выучить C и разработать хороший стиль программирования, эта книга не подойдет. Мой стиль программирования не является образцовым, программы не снабжены подробными комментариями, а также нуждаются в доработке по другим параметрам. Мне просто было интересно решить определенные задачи как можно более коротким путем. Книга написана с точки зрения инженера-электронщика, для которого программирование — лишь средство для достижения цели.

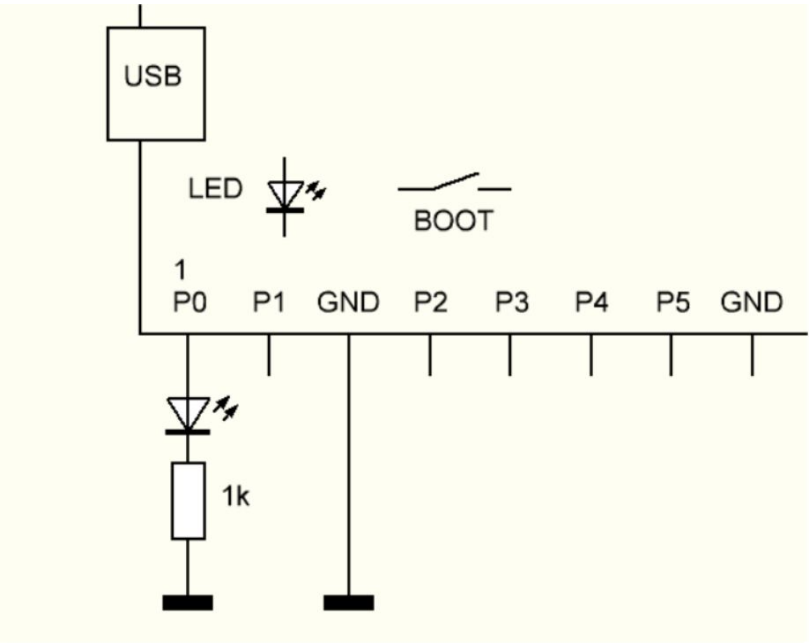
Когда вы увидите решение, которое я использую для точного расчета времени в микросекундах, некоторые программисты на C скажут, что вы можете узнать стиль старых программ на языке Basic.

Я запрашиваю микросекундный таймер и жду, пока не будет достигнуто определенное время. Профессионал, возможно, предпочтет использовать прерывание по таймеру. В других местах я заполняю области данных циклично, а профи для этого использовал бы функцию DMA (Direct Memory Access). По программам в этой книге вы можете сказать, что они не были написаны программистом на языке C. Но они компактны и выполняют свою работу. В некоторых местах я осознавал, что большего можно достичь другими средствами. Но исходные тексты стали намного длиннее и труднее для понимания.

Большинство проектов в этой книге относятся к области измерительной техники. Это опять-таки связано с моей точкой зрения как инженера-электронщика. Когда я просматриваю технические характеристики микроконтроллера, первое, что я вижу, — это возможные применения в измерительной технике. Pico может измерять частоты до 100 МГц - надо попробовать! Если я захочу решить эту задачу с помощью цифровых микросхем, это потребует огромных усилий. С Pico это очень просто и недорого. Если вас меньше интересует техника измерений, у вас могут быть совершенно другие задачи, но со схожими проблемами. Так что возможно вы сможете использовать представленные программы как карьер, брать на себя мелкие детали и решать с их помощью совершенно разные свои задачи.

2 Управление светодиодом

Первый тест уже показал, как можно управлять светодиодом на плате, но теперь должен мигать другой светодиод. Для этого к P0 (вывод 1) подключается светодиод с последовательно подключенным резистором сопротивлением 1 кОм.

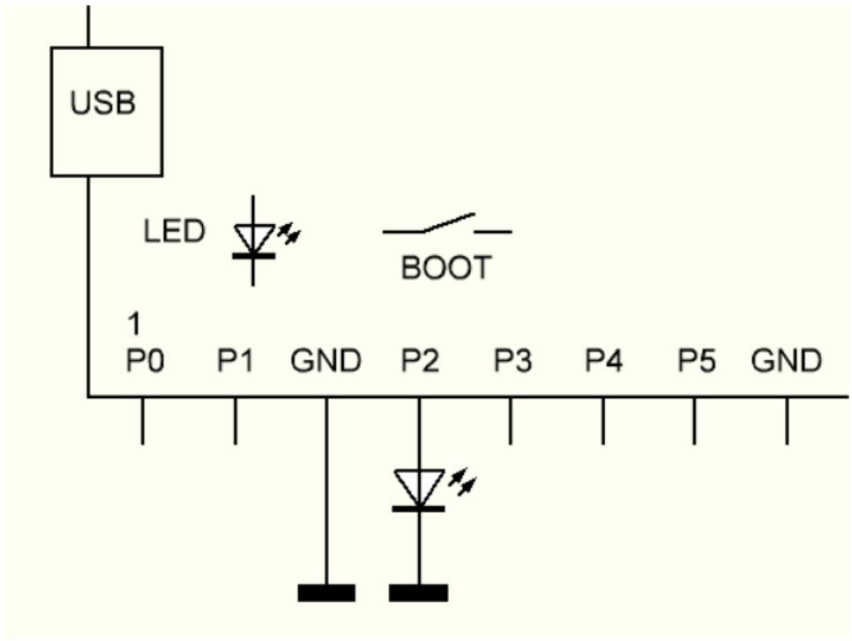



```
//Pico_Blink2
int led = 25;

void setup() {
  pinMode(led, OUTPUT);
  pinMode(0, OUTPUT);
}

void loop() {  digitalWrite(led,HIGH);
digitalWrite(0, 1);
delay(500);
digitalWrite(led, LOW);
digitalWrite(0, 0);
delay(500);
}
```

С pinMode(0, OUTPUT); вывод P0 становится выходом. В функции цикла он включается с помощью digitalWrite(0, 1); а затем выключился с помощью digitalWrite(0, 0);. Обозначения HIGH и 1, а также LOW и 0 эквивалентны. Таким образом, BoardLED и внешний светодиод теперь управляются в одной фазе. Командные строки выполняются одна за другой, но результирующие задержки составляют менее микросекунды и поэтому остаются невидимыми.

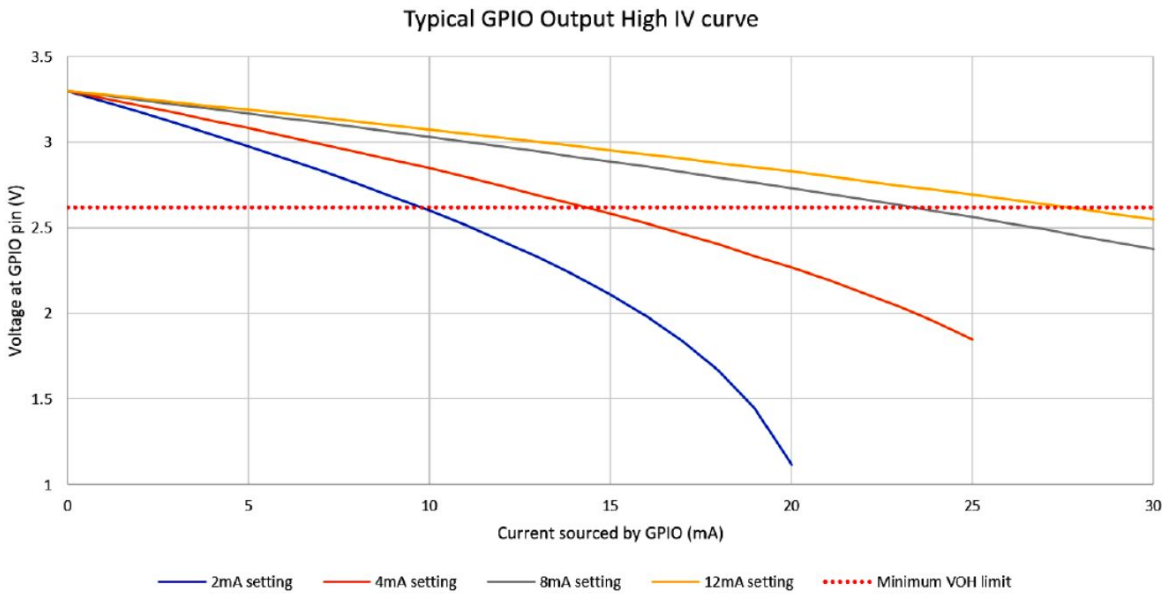
У вас есть совершенно другие возможности с функциями из SDK (C-Software Development Kit). Чтобы использовать их, вам необходимо включить заголовочный файл pico/stdlib.h и, в других случаях, другие заголовочные файлы. Функции теперь имеют немного другие имена. Чтобы использовать порт, вам необходимо инициализировать его с помощью gpio_init и переключить его как выходной с помощью gpio_set_dir. На этот раз используется P2.

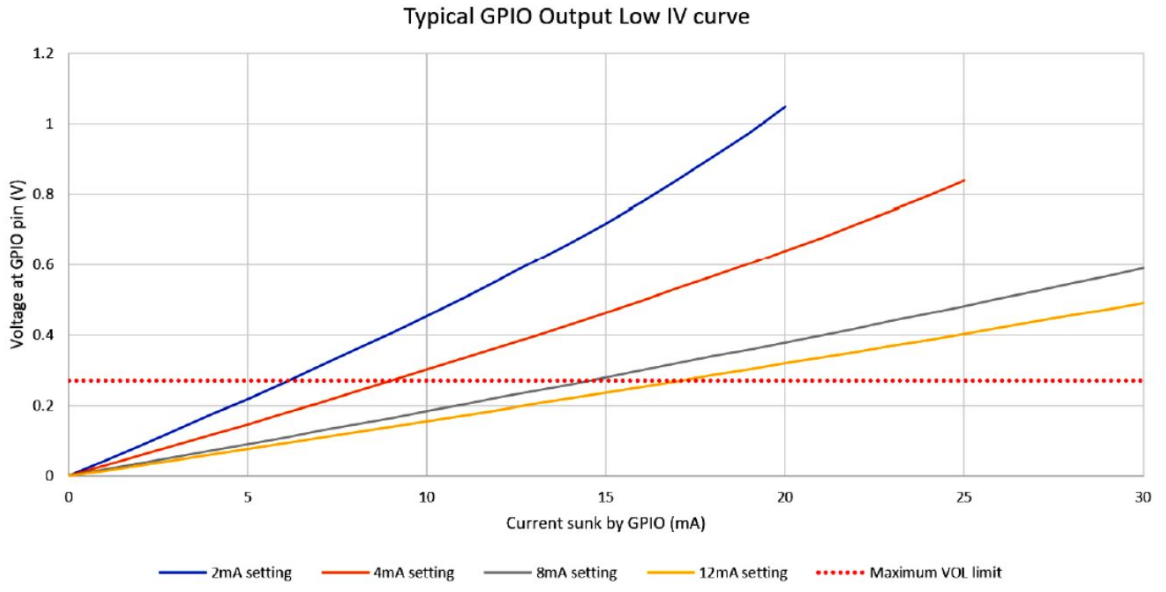
Кроме того, можно установить выходной ток порта. С помощью функции gpio_set_drive_strength(2, GPIO_DRIVE_STRENGTH_2MA) выходной ток снижается до такой степени, что можно подключить светодиод без последовательного резистора. Однако спецификацию 2 мА следует интерпретировать с осторожностью. Согласно техпаспорту, при указанном токе все равно гарантировано напряжение 2,62 В. Фактически при такой настройке у зеленого светодиода был измерен ток короткого замыкания 20 мА и рабочий ток 13 мА.




```
//Pico_Blink3
#include "pico/stdlib.h"

void setup() {
  gpio_init(2);
  gpio_set_dir(2, 1);
  gpio_set_drive_strength(2, GPIO_DRIVE_STRENGTH_2MA);
  while (true) {
    gpio_put(2, 1);
    sleep_ms(500);
    gpio_put(2, 0);
    sleep_ms(500);
  }
}

void loop() {}
```

В техпаспорте RP2040 есть графики ожидаемого тока в зависимости от напряжения на порту. Они подтверждают, что светодиод с настройкой 2 мА можно подключить напрямую.


Typical GPIO Output High IV curve
Voltage at GPIO pin (V)
3.5
3
2.5
2
1.5
1
0
5
10
15
20
25
30
Current sourced by GPIO (mA)
2mA setting
4mA setting
8mA setting
12mA setting
Minimum VOH limit


Typical GPIO Output Low IV curve
Voltage at GPIO pin (V)
1.2
1
0.8
0.6
0.4
0.2
0
0
5
10
15
20
25
30
Current sunk by GPIO (mA)
2mA setting
4mA setting
8mA setting
12mA setting
Maximum VOL limit

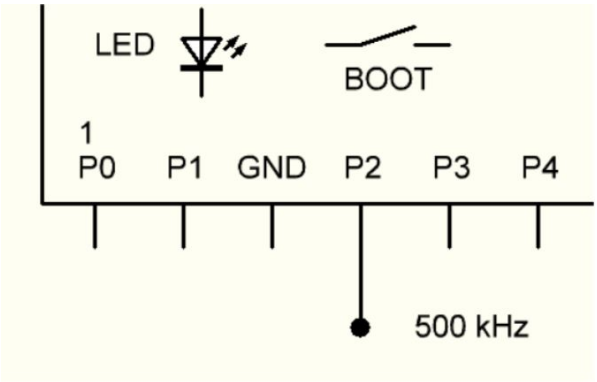
3 Быстрый выходной порт

До сих пор функция задержки Arduino или функция сна SDK sleep_ms использовались для создания определенных задержек в несколько миллисекунд или секунд. Также имеется sleep_us для задержки на микросекунды. Однако это не обеспечивает точное время выполнения, поскольку оно добавляется к другому, неизвестному времени выполнения.

Лучшим вариантом является использование микросекундного таймера в пико. Это 64-битный счетчик, который считает с циклом 1 мкс. Функция time_us_32() считывает младшие 32 бита, чего обычно достаточно.

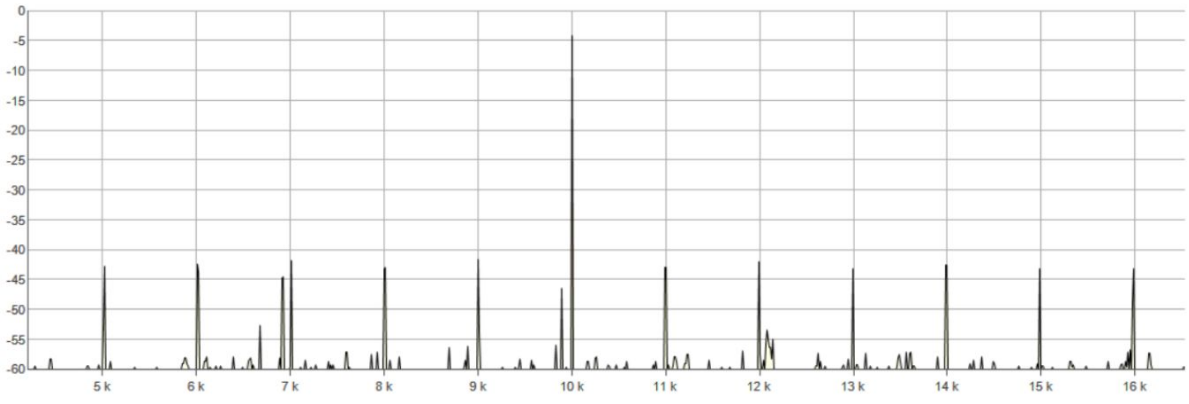
Следующая программа ждет следующей микросекунды, чтобы переключить состояние порта.

```
//Pico_time_us
#include "pico/stdlib.h"

void setup() {
  gpio_init(2);
  gpio_set_dir(2, true);
  uint32_t t = time_us_32();
  while (true) {
    t+=1; while (t>time_us_32());
    gpio_put(2, 1);
    t+=1; while (t>time_us_32());
    gpio_put(2, 0);
  }
}
```



Программа генерирует прямоугольный сигнал частотой 500 000 кГц на порту P2. Однако если вы внимательно изучите сигнал, вы заметите небольшие задержки в несколько микросекунд после каждой миллисекунды. Вероятно, они связаны с активным интерфейсом USB. Тот факт, что частотомер тем не менее измеряет ровно 500 кГц, обусловлен использованием микросекундного таймера. Если импульс пропущен, следующие импульсы следуют быстрее, пока часы снова не станут точными.

Если вы посмотрите на сигнал или его гармоники в спектре коротковолнового приемника, вы увидите, например, сильный сигнал на частоте 3500 кГц, но сопровождаемый на 40 дБ более слабым количеством побочных излучений с интервалом ровно 1 кГц каждое.



Далее ниже будет показано, как эту проблему можно решить за счет аутсорсинга второго ядра процессора. Другая возможность — генерировать сигналы с выходом ШИМ.

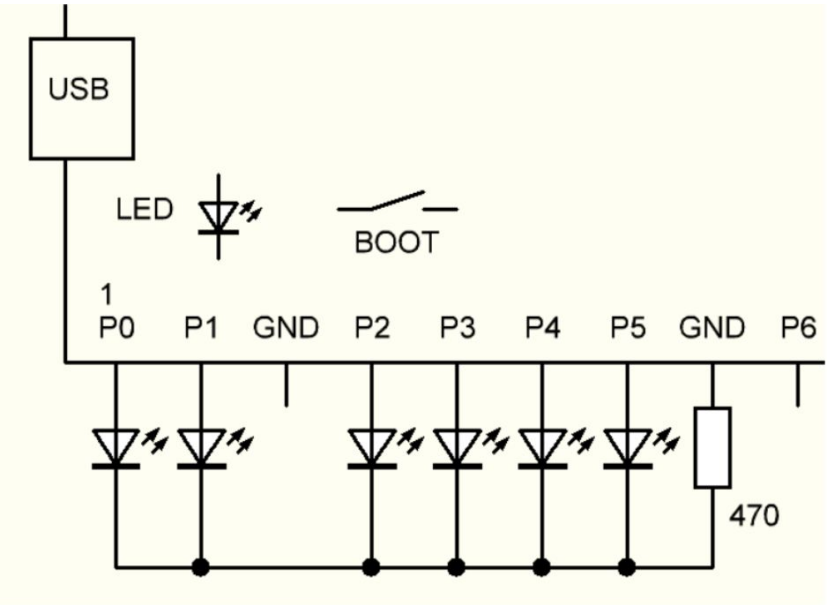
4 ШИМ-управление

Обычный язык Arduino включает в себя аналог Write. Это формирует выход ШИМ с разрешением 8 бит. Никакой специальной инициализации не требуется.

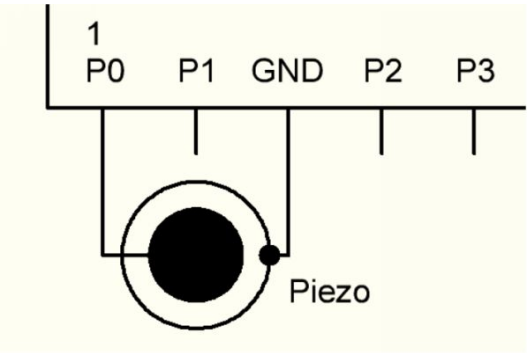
```
//Pico_PWM
unsigned char t =0;
void setup() {
  Serial.begin(115200);
}

void loop() {
  t++;
  unsigned int a1=120+110*sin(t/128.0*3.1415);
  Serial.println(a1);
  a1=(a1*a1)/256;
  analogWrite (0, a1);
  unsigned int a2=120+110*sin(t/128.0*3.1415-1);
  a2=(a2*a2)/256;
  analogWrite (1, a2);
  unsigned int a3=120+110*sin(t/128.0*3.1415-2);
  a3=(a3*a3)/256;
  analogWrite (2, a3);
  unsigned int a4=120+110*sin(t/128.0*3.1415-3);
  a4=(a4*a4)/256;
  analogWrite (3, a4);
  unsigned int a5=120+110*sin(t/128.0*3.1415-4);
  a5=(a5*a5)/256;
  analogWrite (4, a5);
  unsigned int a6=120+110*sin(t/128.0*3.1415-5);
  a6=(a6*a6)/256;
  analogWrite (5, a6);
```

```
  delay(5);
}
```

Здесь квазианалоговые выходы ШИМ используются для управления яркостью шести светодиодов. Через ряд должна пройти синусоидальная волна. Поскольку глаз не воспринимает яркость линейно, вычисленные синусоидальные значения возводятся в квадрат, а затем смещаются обратно в диапазон от 0 до 255.

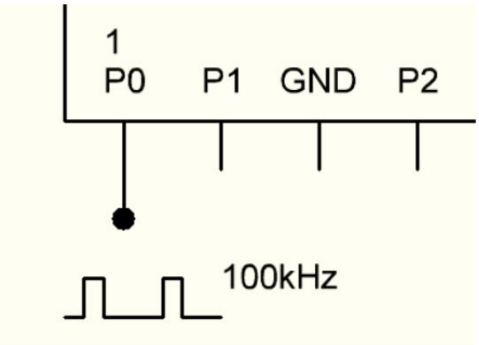


Для шести светодиодов достаточно общего резистора, с помощью которого можно регулировать общую яркость. В зависимости от типа светодиода может оказаться полезным резистор сопротивлением от 100 Ом до 1 кОм.



Кстати, выход ШИМ работает на частоте 1 кГц. К одному из выходов можно подключить пьезопреобразователь и услышать звук. Поскольку длина импульса постоянно меняется, звук также меняется. Короткие импульсы обеспечивают более высокую долю гармоник, в то время как сигнал с коэффициентом заполнения 50 % обеспечивает только нечетные гармоники, которые уменьшаются с частотой.

Здесь снова расширены возможности функций SDK. На этот раз также необходимо включить заголовочный файл hardware/pwm.h. Теперь вы можете устанавливать частоту ШИМ, разрешение и ширину импульса в широких пределах.



Модули ШИМ Pico состоят из 16-разрядных счетчиков прямого преобразования, которые работают на системной частоте или на частоте, деленной на нее. С помощью gpio_set_function вы определяете, какой порт должен стать выходом ШИМ, в данном случае порт P0. pwm_set_wrap определяет, до какого уровня должен работать счетчик, прежде чем он снова начнет работать с нуля. В этом случае счетчик должен работать от 0 до 1249, т.е. отсчитывать 1250 шагов. Поскольку пико работает на частоте 125 МГц, частота ШИМ составляет 100 кГц. pwm_set_gpio_level устанавливает ширину импульса, в данном случае 250 шагов счета, т.е. 20 % от общей ширины. Таким образом, порт включен на 2 мкс и выключен на 8 мкс. С помощью pwm_set_enabled вы включаете или выключаете выход ШИМ.

//Pico_PWM2

```
#include "pico/stdlib.h"
#include "hardware/pwm.h"

void setup() {
  gpio_set_function(0, GPIO_FUNC_PWM);
  pwm_set_wrap(0, 1249);
  pwm_set_gpio_level(0, 250);
  pwm_set_enabled(0, true);
}

void loop() {}
```

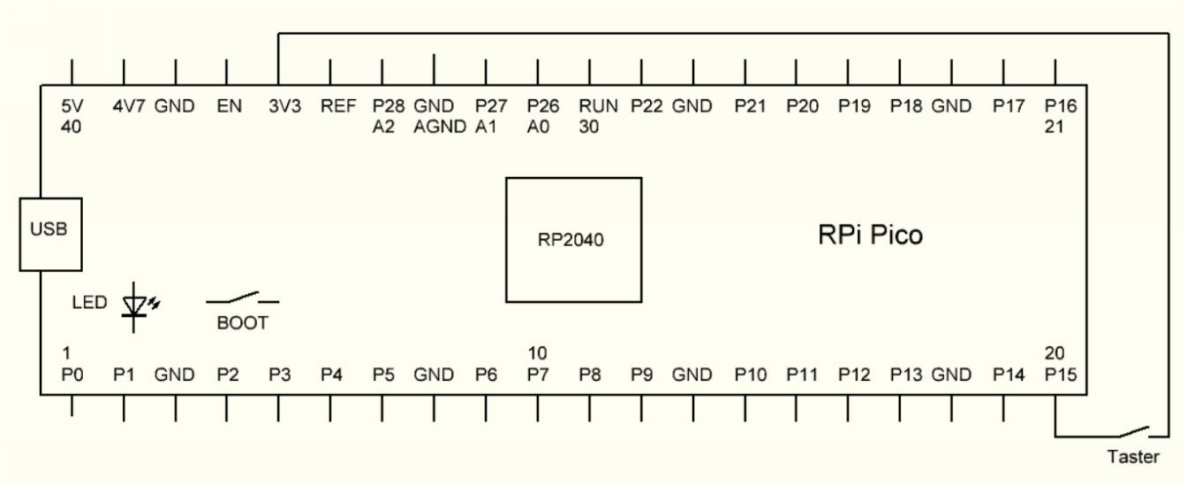
Выход ШИМ очень универсален благодаря функциям SDK. Вы можете выбрать генерацию определенной частоты или задать определенное разрешение, например 8-битное, 10-битное или 16-битное. Это дает вам очень гибкий генератор прямоугольных импульсов, работающий на частотах до нескольких МГц. Наименьшая частота без использования прескалера составляет 125 МГц/65 536 = 1907,35 Гц.

Весь программный код находится в функции setup(). Кроме того, необходимо использовать пустую функцию Loop(). В качестве альтернативы можно также использовать функцию int main(). Но тогда никакого COM не образуется и при следующей программе автоматический вызов загрузчика уже не работает, поэтому придется запускать его с ресетом и загрузочным ключом.

5 Цифровые входы

Каждый порт RPi Pico можно использовать как цифровой выход или как цифровой вход. Запрос с помощью digitalRead затем возвращает либо 1, либо 0, в зависимости от того, присутствует ли высокое напряжение (3,3 В) или низкое напряжение (0 В).

Особенностью RPi Pico являются его внутренние понижающие резисторы, которые активируются на каждом порту после сброса. Таким образом, в отличие от других контроллеров, у вас нет входов с высоким сопротивлением, которые могут принимать случайные состояния, но вы надежно считываете ноль (низкий уровень) на всех неиспользуемых входах. Соответственно, коммутатор должен быть подключен к 3V3. В первом примере порт P15 опрашивается на контакте 20. При нажатии переключатель поднимает порт до 3,3 В.

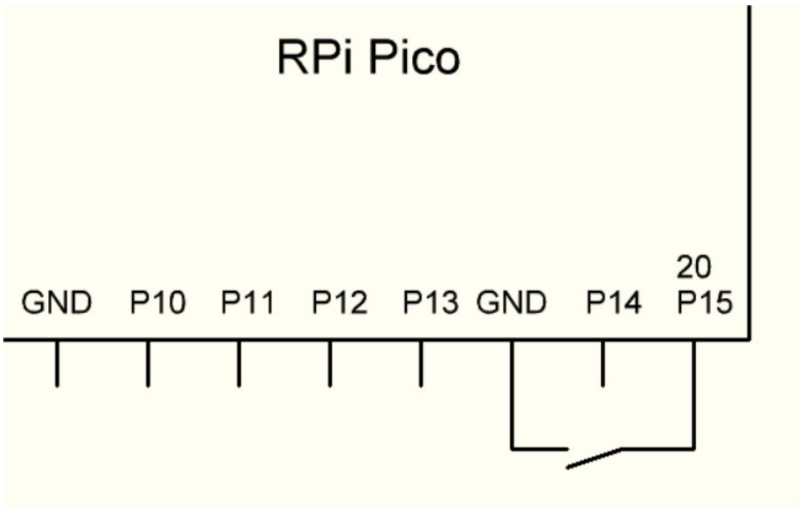



```
//Pico_Din

void setup() {
  pinMode(25, OUTPUT);
}

void loop() {
  int i = digitalRead (15);
  digitalWrite(25, i);
  }
```

Программа предназначена для включения светодиода платы на GPIO25 при нажатии кнопки. Поэтому порт 25 инициализируется как выход. Вход не обязательно должен быть инициализирован, поскольку каждый порт после сброса переключается как вход. В функции цикла порт 15 запрашивается снова и снова. Результат (1 или 0) попадает в i. Затем i выводится на порт 25 и, таким образом, светодиод на плате включается при нажатии переключателя.

Часто лучше размещать внешние кнопки напротив земли. Тогда порт должен быть снабжен подтягивающим резистором, который подтягивает его в состоянии ожидания. При инициализации P15 с помощью pinMode(15, INPUT_PULLUP), он выключается. Это приводит к состоянию ожидания порта 1. Теперь кнопка должен быть замкнута на GND, чтобы переключить состояние на 0. Возможна также инициализация полностью без подтягивающего резистора.

```
//Pico_Din2 Pullup

void setup() {
  pinMode(25, OUTPUT);
  pinMode(15, INPUT_PULLUP);
}

void loop() {
  int i = digitalRead (15);
  digitalWrite(25, i);
}
```



Таким образом, функция кнопки меняется на противоположную. В режиме ожидания на плате светится светодиод. При нажатии кнопки он гаснет. Однако при необходимости результат запроса можно инвертировать:

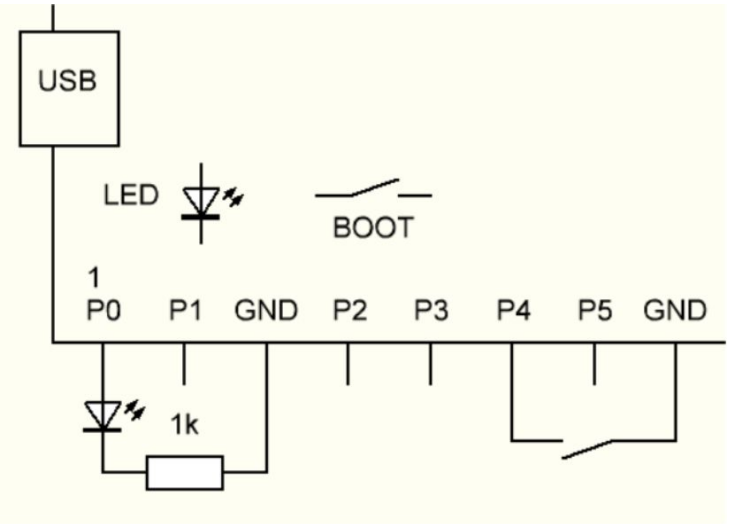
```
int i = !digitalRead(15);
```

Если вам нужен вход с высоким импедансом, уберите ограничение напряжения. Тогда инициализация называется pinMode(15, INPUT). Активное включение подтягивающего резистора в данной среде невозможно, для этого нужны расширенные функции SDK.

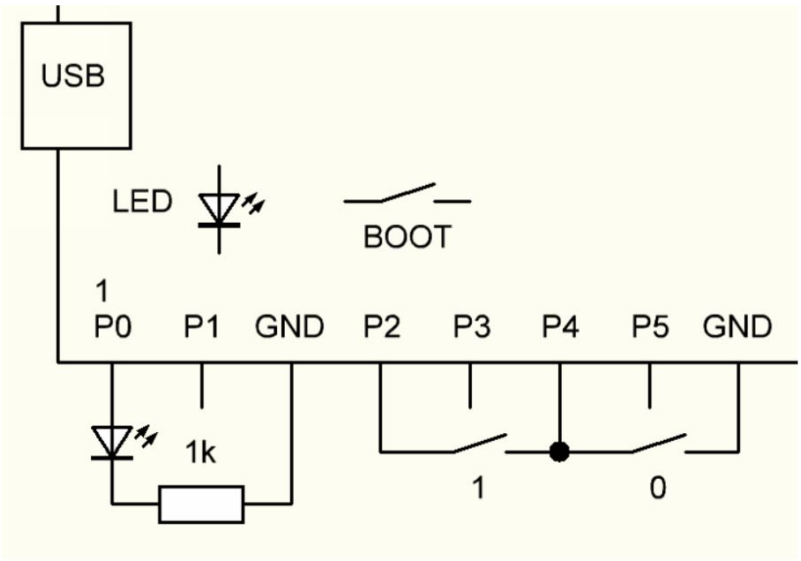
```
//Pico_Din3 Pulls
#include "pico/stdlib.h"

void setup() {
  gpio_init(0);
  gpio_set_dir(0, 1);
  gpio_init(4);
  gpio_set_dir(4, 0);
  gpio_pull_up(4);
  //gpio_pull_down(4);
  //gpio_set_pulls(4,0,0);
  while(true){
    gpio_put(0,gpio_get(4));
  }
```

```
}

void loop() {}
```

Здесь P4 запрашивается с помощью внутреннего подтягивания, и P0 переключается соответствующим образом. В исходном коде показаны закомментированные строки с дополнительными функциями для подтягивающих резисторов.



Когда одновременно включены как подтягивание, так и ограничение напряжения, вход имеет совершенно особое поведение: gpio_set_pulls (4,1,1) Фактически, одновременно активен только один из двух, но тот, который удерживает достигнутое цифровое состояние, включен. Тогда вход ведет себя как RS-триггер. Его можно поднять или опустить короткими контактными действиями, после чего порт останется в соответствующем состоянии.



Здесь используются две кнопки. Правая подтягивает порт к GND, другой — до 3,3 В. Для этого включается порт P2. С одной стороны, это имеет то преимущество, что соединение к положительному полюсу короче. А с другой стороны, это повышает безопасность при коротком замыкании. Ведь если обе кнопки случайно нажаты одновременно, порт P2 ограничивает ток прибл. 30 мА.

```
//Pico_Din4 Pullup+Pulldown
#include "pico/stdlib.h"

void setup() {
  gpio_init(0);
  gpio_set_dir(0, 1);
  gpio_init(2);
  gpio_set_dir(2, 1);
  gpio_put (2,1);
  gpio_init(4);
  gpio_set_dir(4, 0);
  gpio_set_pulls(4,1,1);
  while(true){
    gpio_put(0,gpio_get(4));
  }
}
void loop()
{
```

Фактически, Pico теперь ведет себя как триггер RS. Короткого нажатия на одну из двух кнопок достаточно, чтобы привести в нужное состояние. Это поведение не видно программе, поскольку это только поведение входа P4, при котором оба притягивающих резистора были активированы с помощью gpio_set_pulls(4,1,1).

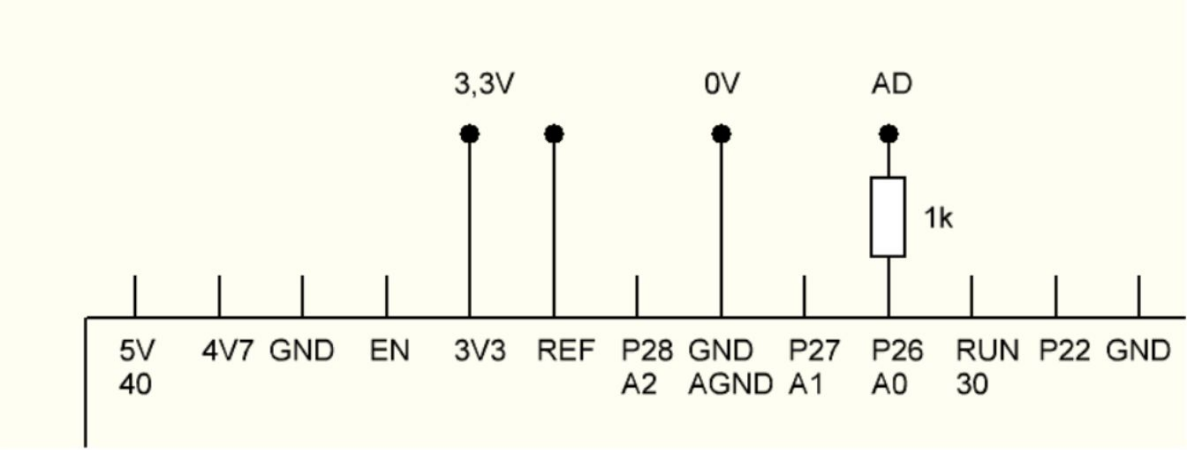
Просто для сравнения вы можете соединить два порта вместе, один из которых будет получать активное подтягивание, а другой — ограничение. Фактически, тогда у вас есть делитель напряжения с примерно половиной рабочего напряжения на обоих портах.

Вы можете измерить включенный подтягивающий илипонижающий резистор с помощью омметра и найтисопротивление около 20 кОм относительно GND или 3V3. Однаков паспорте указано, что она должна составлять от 50 до 80 тысяч. Если вы измерите ток относительно +3,3 В, вы обнаружитезначение около 58 мкА, что указывает на понижениесопротивления 57 кОм. Таким образом, понижение напряженияне является постоянным, а увеличивается с увеличениемнапряжения. Характеристическая кривая, кажется, больше соответствует полевому транзистору, который все больше ибольше приближается к источнику постоянного тока с болеевысоким напряжением.

6 Измерение напряжения

RPi Pico имеет три аналоговых входа от ADC0 до ADC2 и два других, которые недоступны снаружи. По линии REF подается опорное напряжение, которое подключается к 3,3 В через RC-фильтр нижних частот. Таким образом, диапазон измерения АЦП составляет от 0 В до 3,3 В.

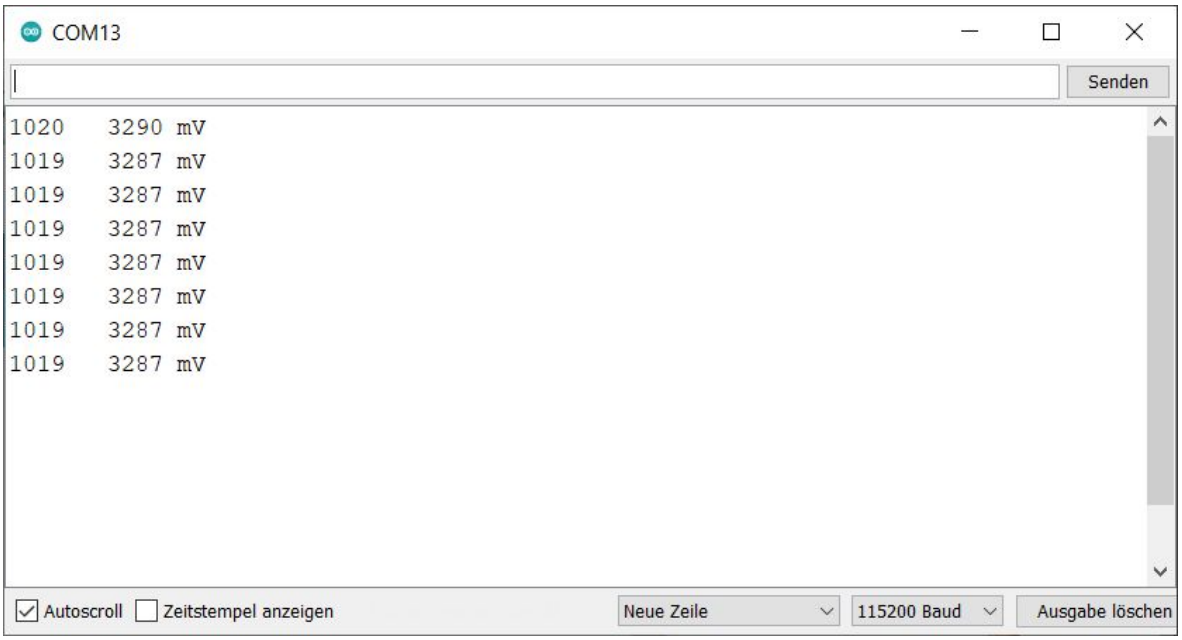
На этот раз контакт 33, AGND (аналоговое заземление) используется в качестве соединения GND, чтобы свести к минимуму помехи при измерениях регулятором напряжения на плате. В первом тесте измеряется рабочее напряжение 3,3 В. Резистор сопротивлением 1 кОм встроен в измерительный вход в качестве меры предосторожности и для предотвращения повреждения в случае, если кто-то случайно подаст более высокое или отрицательное напряжение.




```
//Pico_voltage AD0
unsigned int ad,u;

void setup() {
  Serial.begin(115200);
}

void loop() {
  ad =analogRead(A0);
```

```
  u = (unsigned int) ad*3300/1023;
  Serial.print(ad);
  Serial.print(" ");
  Serial.print(u);
  Serial.println(" mV ");
  delay(500);
}
```

Программа считывает напряжение на A0 с помощью AnalogRead(A0) и преобразует его в 10-битное числовое значение, т.е. в диапазон от 0 до 1023. Затем оно преобразуется в напряжение в диапазоне от 0 мВ до 3300 мВ. Поскольку результаты вычислений представлены десятичными знаками, результат преобразуется в целое число, т. е. округляется в меньшую сторону. На выходе отображается цифровое измеренное значение и напряжение в мВ.



Измерение на ADC_VREF показало цифровое значение 1019 и напряжение 3287 мВ, при этом небольшие отклонения являются нормальными. Непосредственно на 3V3 измеряется максимальное значение в диапазоне измерения 1023 или 3300 мВ. На GND можно было бы ожидать ноль, но цифровое значение около 2 и измеренное напряжение около 9 мВ. Таким образом, на обоих концах диапазона измерения имеются небольшие ошибки смещения. Наивысшая точность достигается при средних напряжениях.

С функциями SDK программирование АЦП выглядит несколько иначе. Сначала его необходимо инициализировать с помощью adc_init. adc_gpio_init переключает соответствующий вывод на высокий импеданс. Кроме того, канал АЦП должен быть выбран с помощью adc_select_input. Здесь был выбран ADC0 на P26. Фактическое измерение использует запрос adc_read() и возвращает 12-битное значение от 0 до 4095.

В этом примере также было включено подтягивание входного контакта P26 для измерения прямого напряжения на различных диодах. Подтягивающее напряжение обеспечивает низкий ток около 50 мкА без необходимости использования дополнительного резистора.

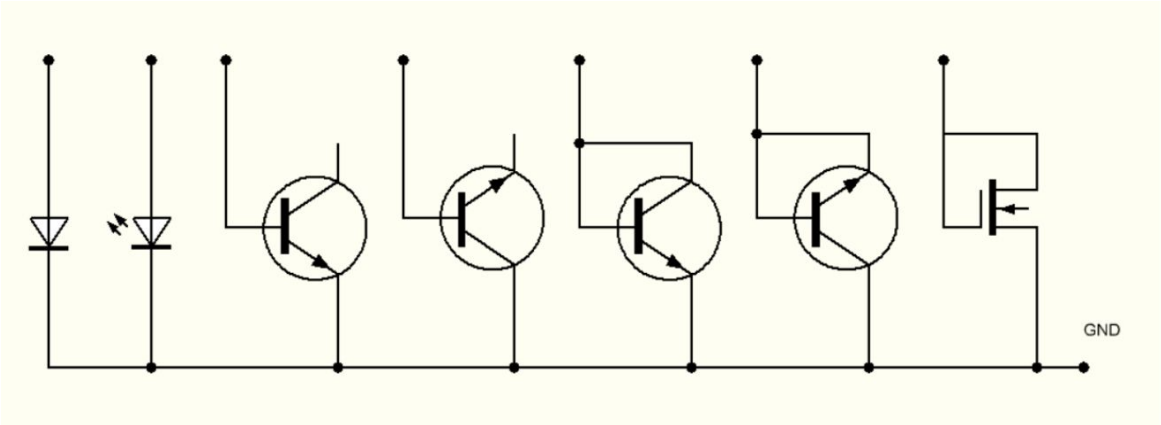
Для обеспечения стабильного отображения десять измеренных значений усредняются. Для этого формируется IIR-фильтр (бесконечная импульсная характеристика). Последнее измеренное значение влияет на результат измерения только на 10 %. При несколько колеблющихся результатах АЦП это позволяет добиться значительного увеличения разрешения.

Аналогичную функцию измерения можно найти в некоторых цифровых мультиметрах, которые также отображают прямое напряжение диодов в диапазоне измерения сопротивления. Таким образом можно распознать тип диода и, например, отличить диоды Шоттки от обычных кремниевых диодов.

```
//Pico_AD2 diode test

#include "pico/stdlib.h"
#include "hardware/adc.h"

void setup() {
  int iir=0;
  Serial.begin(115200);
  adc_init();
  adc_gpio_init(26);
  gpio_pull_up(26);
  adc_select_input(0);
  uint32_t t = time_us_32();
  while(true){
    uint32_t ad = adc_read()*3300/4095;
    for (int n=0; n<10; n++){
      ad = adc_read()*3300/4095;
      iir = (iir*9+ad)/10;
    }
    Serial.println(iir);
    sleep_ms(500);
  }
}

void loop() {
}
```

Светодиоды невозможно проверить с помощью цифровых приборов, поскольку их прямое напряжение (светодиодов) слишком велико. Однако с Pico это работает, поскольку напряжение может достигать 3,3 В. При подключении желтого светодиода было измерено напряжение 1770 мВ. При этом видно очень слабое свечение. Для сравнения: диод 1N4148 показал всего 490 мВ. Поразительно низкое напряжение обусловлено большим сопротивлением подтяжки. При сопротивлении 1 кОм против VCC напряжение выросло до 670 мВ.

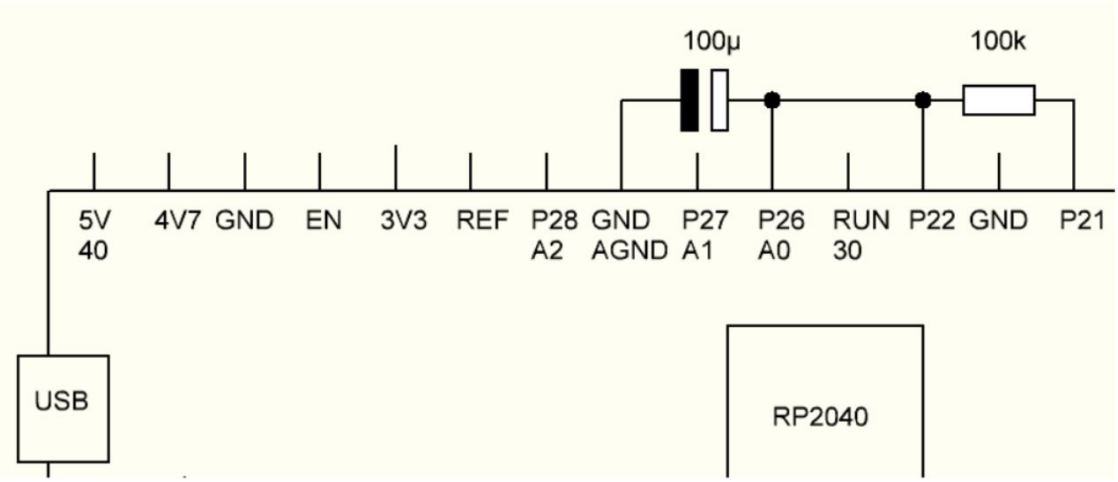
Помимо диодов можно исследовать и транзисторы. На неизвестном транзисторе это можно использовать для идентификации PN-переходов. В случае NPN-транзистора BC547B на переходе БЭ было измерено напряжение 620 мВ, а на переходе БК — 618 мВ. Однако в основном прямое напряжение зависит от температуры и уменьшается со скоростью 2 мВ/к. Просто прикоснувшись рукой, напряжение падает прибл. 10 мВ.



Если коллектор также подключен к базе, напряжение падает прибл. 570 мВ. Транзистор имеет 300-кратное усиление тока. Следовательно, через базу протекает только в 300 раз меньший ток, что приводит к соответственно меньшему прямому напряжению. Инвертированный транзистор с коллектором на отрицательном полюсе тоже работает, но с гораздо меньшим усилением тока. Тем не менее, напряжение ок. 560 мВ было измерено даже в инвертированном режиме с соединенными базой и эмиттером. В аналогичной схеме можно исследовать и полевой транзистор. При подключенных затворе и стоке BS170 показал напряжение 1770 мВ. Это дает вам представление о напряжении, при котором полевой транзистор начинает проводить ток, и о том, как им нужно управлять.

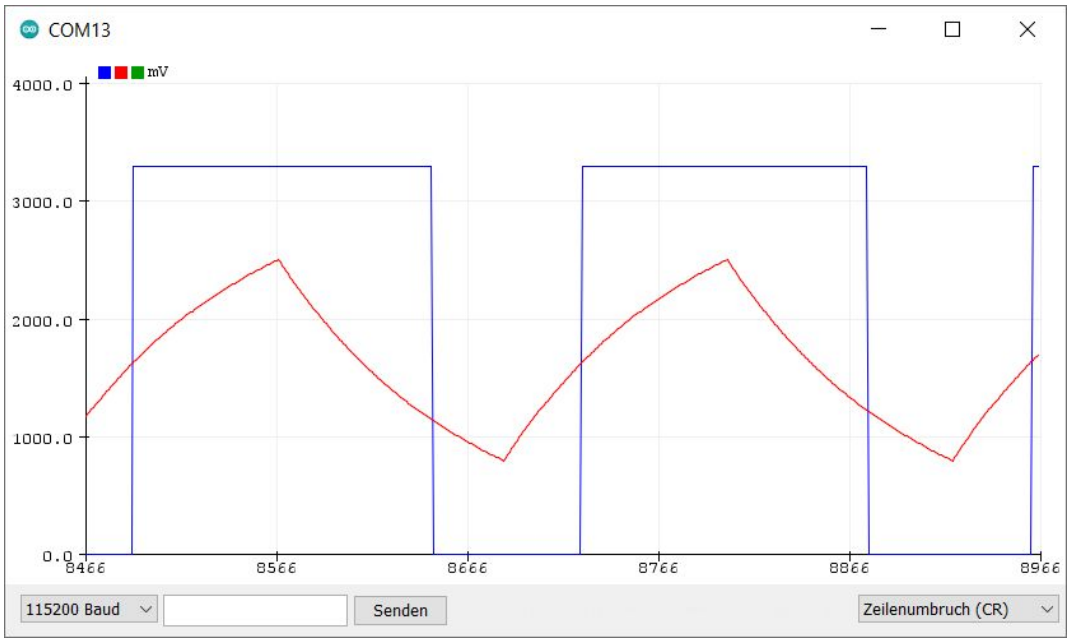
8 Входные пороги и гистерезис

Какое входное напряжение цифрового порта читается как 1, а какое как 0? Чтобы рассмотреть это более подробно, напряжение теперь измеряется одновременно с помощью аналогового входа A0. Электролитический конденсатор емкостью 100 мкФ обеспечивает медленное изменение. Он подключается через выход P21 и 100 кОм либо к 3V3, либо к GND.



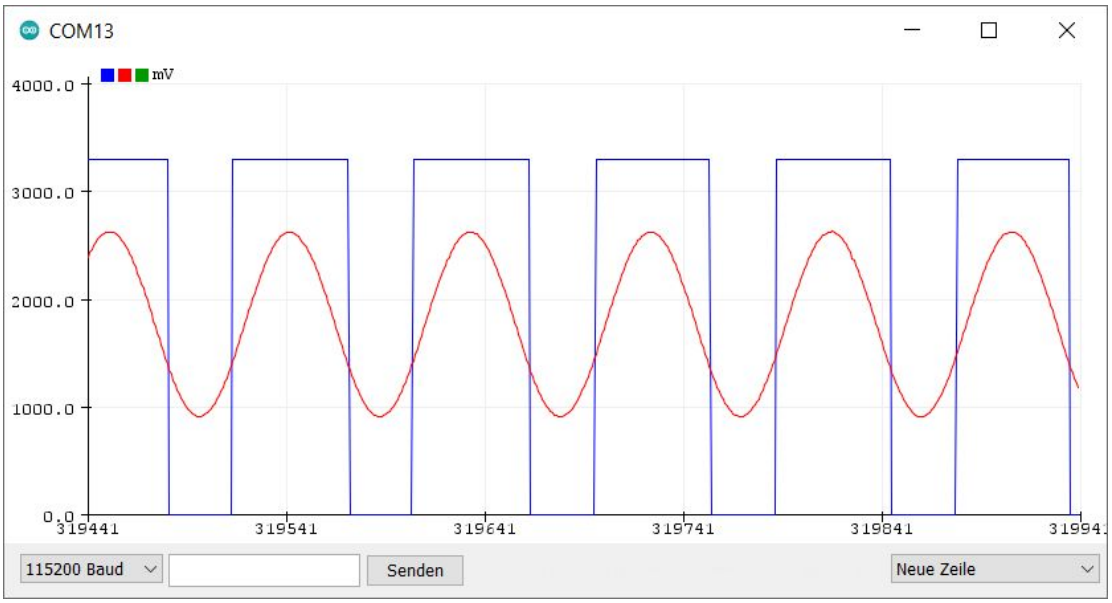

```
//Pico_Din5 Hyteresis
void setup() {
  pinMode(22, INPUT);
  pinMode(21, OUTPUT);
  Serial.begin(115200);
}

void loop() {
  int u =analogRead(A0)*3300/1023;
  if (u<800) digitalWrite(21,1);
  if (u>2500) digitalWrite(21,0);
  Serial.print(digitalRead(22)*3300);
  Serial.print(" ");Serial.print(u);
  Serial.println(" mV ");
  delay(500);
  }
```

Состояние на Din на этот раз выводится как напряжение 0 мВ или 3300 мВ и, таким образом, имеет тот же диапазон, что и измерение напряжения. Это позволяет отображать реакцию входа на цифровом плоттере.



При измерении зарядный резистор всегда подключался к земле, когда электролитический конденсатор заряжался до напряжения 2,5 В, и наоборот к 3 В3, когда он разряжался до 0,8 В. Измерение напряжения показывает типичные экспоненциальные кривые зарядки конденсатора. Цифровой вход реагирует на небольшое смещение и включается прибл. 1,5 В, но выключается только ниже приблиз. 1,2 В. Расстояние между двумя точками переключения представляет собой гистерезис. В этом диапазоне всегда сохраняется последнее считанное состояние.

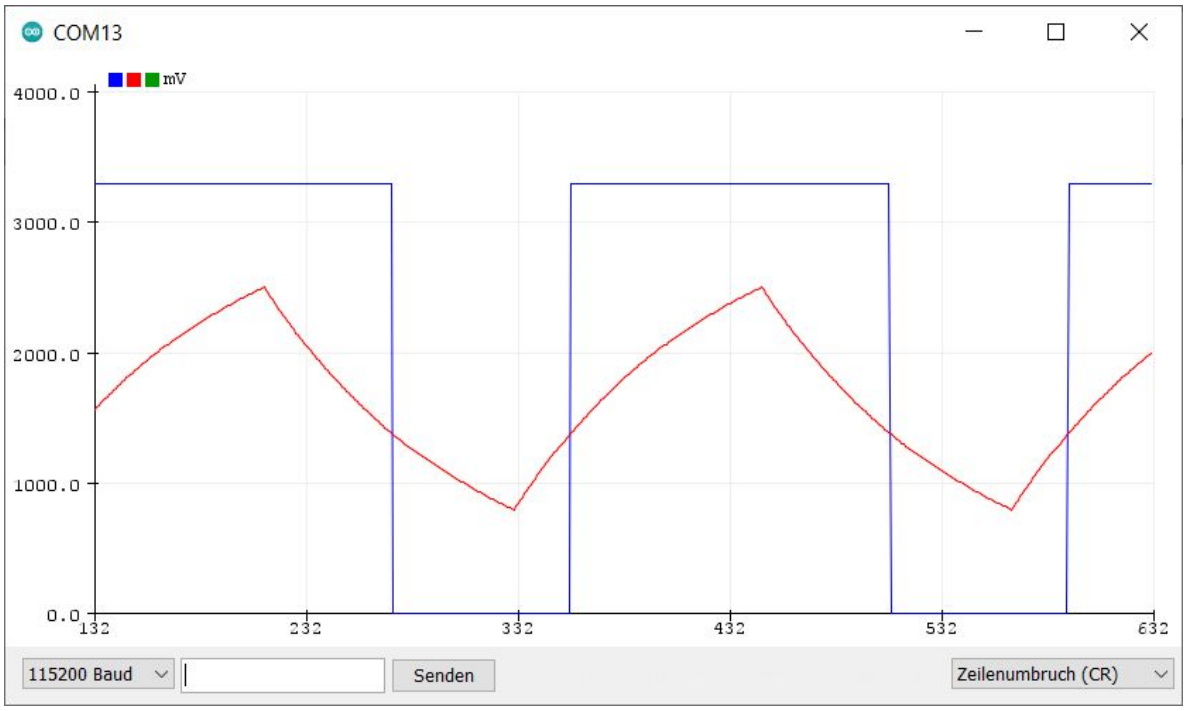
Гистерезис цифрового входа полезен для постоянного считывания уникальных состояний. Это означает, что напряжение на цифровом входе не может быть настолько близко к порогу, чтобы состояние иногда считывалось как 1, а иногда как 0. Если вход управляется синусоидальнымнапряжением,гистерезиспревращаетегов уникальный прямоугольный сигнал.



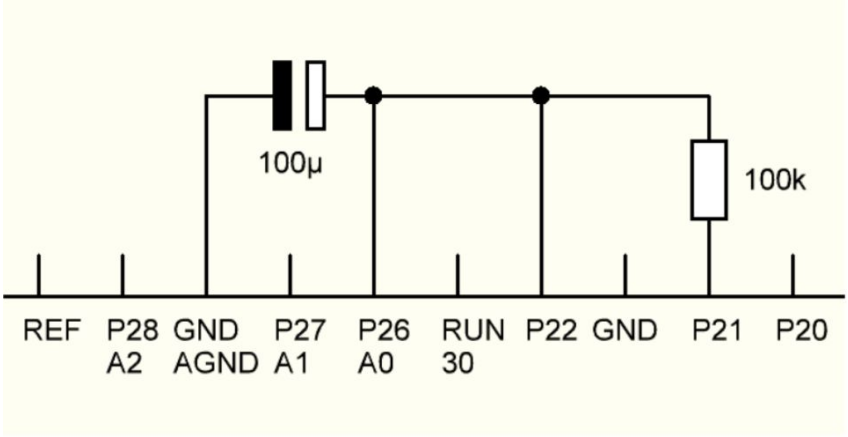
У RP2040 есть возможность отключения гистерезиса, но для этого требуется функция SDK. С помощью функции gpio_set_input_hysteresis_enabled(22,0) гистерезис на входе P22 отключается. Измерение ясно показывает, что точки переключения теперь находятся на одном уровне.

```
//Pico_Din6 Hyteresis off
#include "pico/stdlib.h"

void setup() {
  pinMode(22, INPUT);
  gpio_set_input_hysteresis_enabled(22,0);
  pinMode(21, OUTPUT);
  Serial.begin(115200);
}

void loop() {
  int u =analogRead(A0)*3300/1023;
  if (u<800) digitalWrite(21,1);
  if (u>2500) digitalWrite(21,0);
```

```
  Serial.print(digitalRead(22)*3300);
  Serial.print(" ");
  Serial.print(u);
  Serial.println(" mV ");
  delay(500);
}
```



Для некоторых задач полезно более точно знать точки гистерезиса. Поэтому теперь необходимо выполнить автоматическое измерение, которое выводит только точки переключения, но не всю кривую напряжения. Порт 21 управляет процессом зарядки и разрядки электролитического конденсатора в качестве выхода. Напряжения измеряются и выводятся в обеих точках переключения.




```
//Pico_Din7 Hysteresis points
unsigned int u;

void setup() {
  Serial.begin(115200);
  pinMode(22, INPUT);
  pinMode(21, OUTPUT);
}

void loop() {
  digitalWrite (21,1);
  while (!digitalRead(22)){
    u= (int) analogRead(A0)*3300/1023;
  }
  Serial.print(u);
  Serial.print(" mV ");
  digitalWrite (21,0);
  while (digitalRead(22)){
    u= (int) analogRead(A0)*3300/1023;
  }
  Serial.print(u);
  Serial.println(" mV ");
}
```

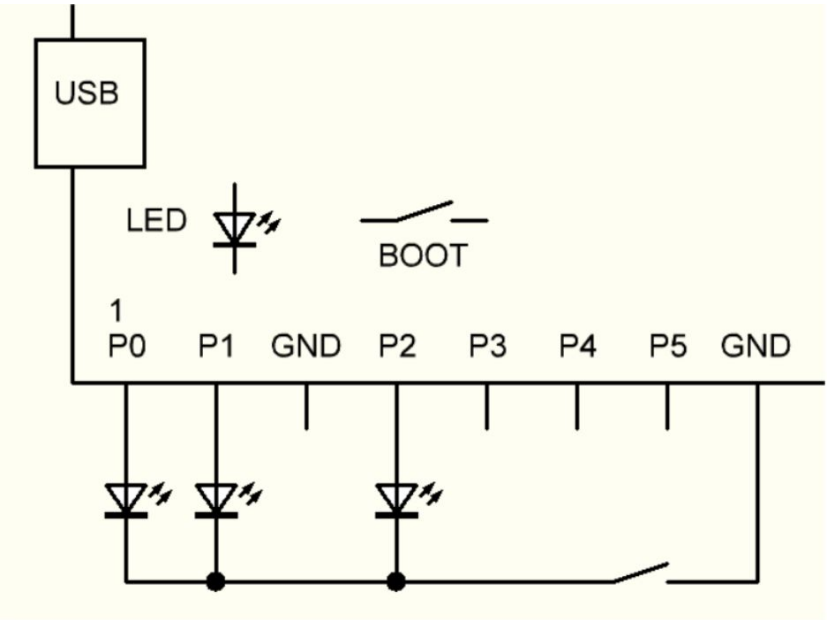
1587 mV 1177 mV
1590 mV 1180 mV
1590 mV 1180 mV
1587 mV 1180 mV
1587 mV 1177 mV
1590 mV 1180 mV
1583 mV 1180 mV
1590 mV 1180 mV
1593 mV 1180 mV
1593 mV 1183 mV

Слегка округленный результат: верхняя точка переключения составляет 1,59 В, а нижняя 1,18 В. Таким образом, существует гистерезис 0,41 В.

9 Измерение температуры

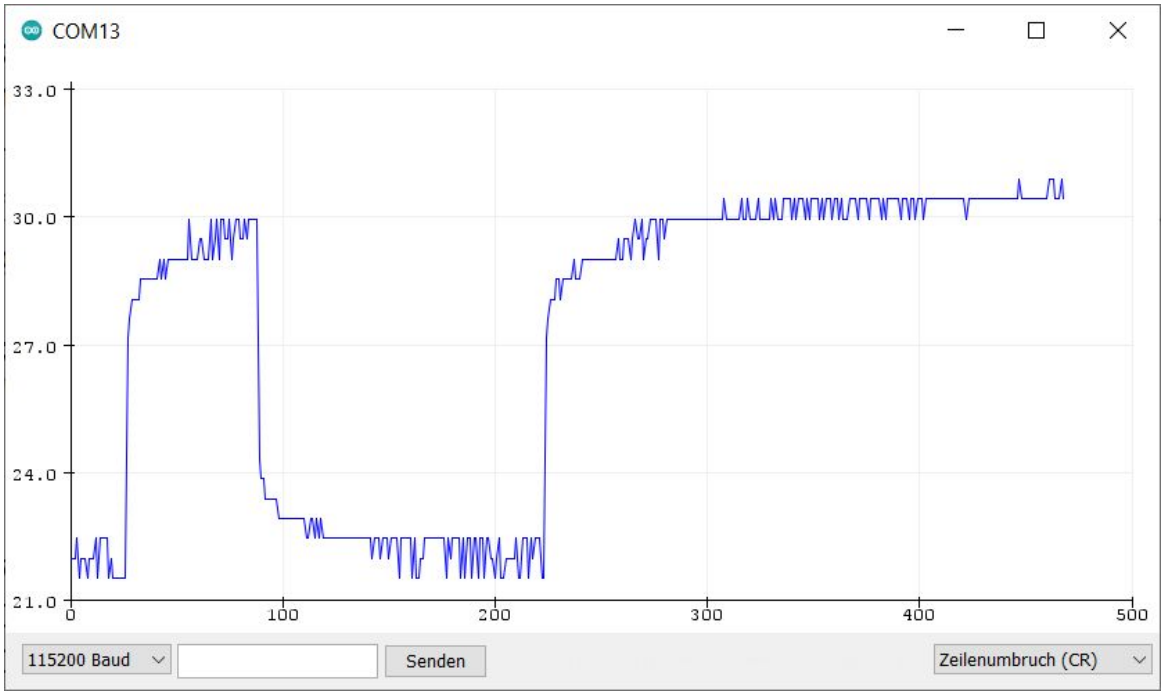
Всего RP2040 имеет пять аналоговых входов, три из которых доступны снаружи на плате PICO PI. Четвертый вход используется для измерения напряжения на плате. А пятый подключен к внутреннему датчику температуры в контроллере. Расширение платы pico поддерживает измерение температуры с помощью функции AnalogReadTemp().

Цель здесь — выяснить, насколько сильно нагревается контроллер, когда он подвергается более тяжелым нагрузкам на порты. Три выхода переключаются на высокий уровень и управляют тремя желтыми светодиодами без последовательного резистора. Поскольку заданный ток составляет 4 мА, через каждый светодиод протекает около 20 мА. Переключатель на линии GND может включать и выключать всю нагрузку.



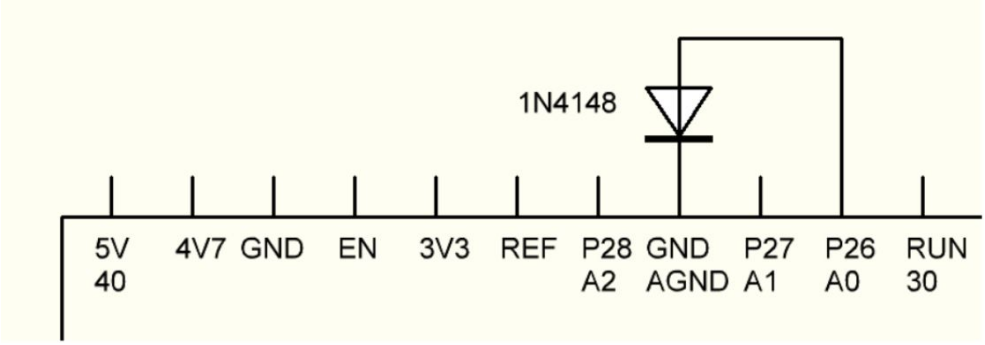

```
//Pico_ADtemp

#include "pico/stdlib.h"
#include "hardware/adc.h"
```

```
void setup() {
  Serial.begin(115200);
  pinMode(0, 1);
  pinMode(1, 1);
  pinMode(2, 1);
  digitalWrite(0, 1);
  digitalWrite(1, 1);
  digitalWrite(2, 1);
}

void loop() {
  Serial.println(analogReadTemp());
  delay(1000);
}
```

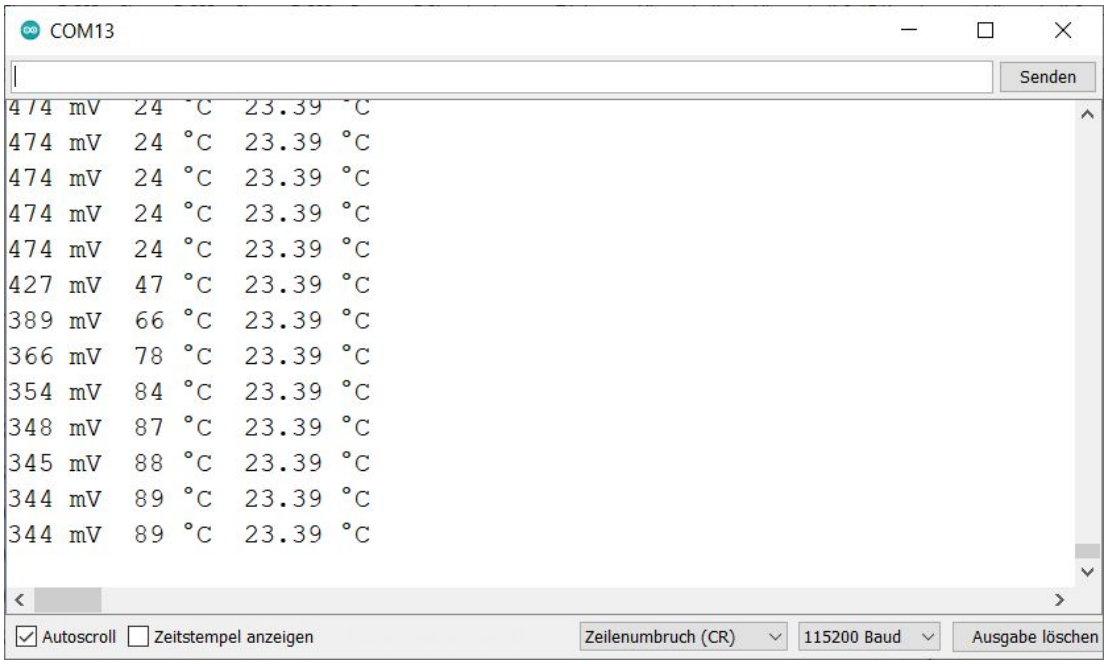


Измерение показывает разогрев прибл. 10 градусов. Однако можно увидеть скачки на 6 градусов при включении и выключении нагрузки, а затем типичный экспоненциальный диапазон нагрева или охлаждения. Таким образом, предполагается, что большие токи в портах приводят к фальсификации измерений температуры. Однако в любом случае общая нагрузка 60 мА не может привести к тепловой перегрузке контроллера.

Как уже было показано на примере диодного тестера, кремниевый диод также является хорошим датчиком температуры. Схема остается прежней. К аналоговому входу A0 подключается диод 1N4148, и включается внутренний подтягивающий резистор. При комнатной температуре измеряется прямое напряжение в диапазоне 500 мВ. Ток диода ок. 50 мкА достаточно мал, чтобы избежать электрического нагрева. Теперь температуру диодного датчика можно рассчитать по точному напряжению.



Программа отображает измеренное напряжение диода, расчетную температуру диода и внутреннюю измеренную температуру микроконтроллера. Если требования к точности не слишком высоки, температуру контроллера можно использовать в качестве сравнения для оптимизации преобразования. При 23 градусах получилось напряжение 476 мВ. Наклон составляет 0,5 мВ/градус. Таким образом, очень высокая точность не достигается. Но преимущество диода в том, что с его помощью можно измерять температуру до 150 градусов. Пример измерения показывает повышение до 89 градусов.

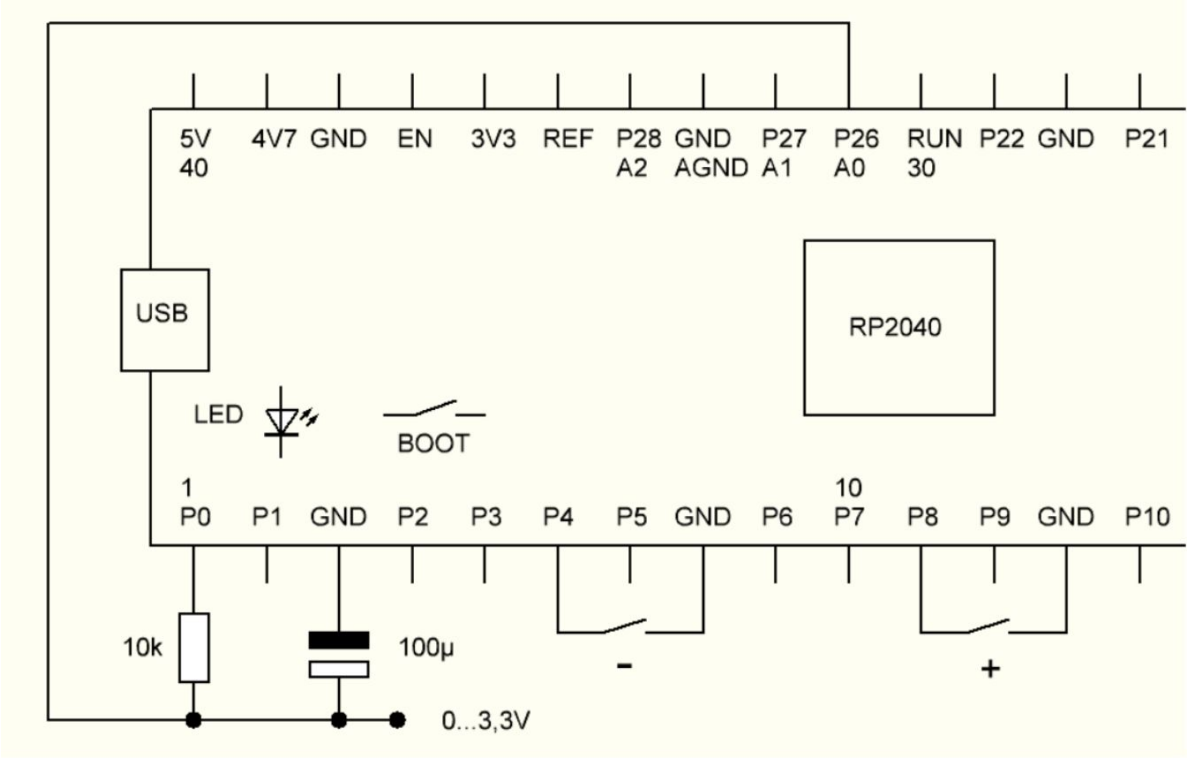



```
//Pico_ADtemp2 Si diode

#include "pico/stdlib.h"
#include "hardware/adc.h"

void setup() {
  int iir=0;
  int ad, temp;
  Serial.begin(115200);
  adc_init();
  adc_gpio_init(26);
  gpio_pull_up(26);
  adc_gpio_init(26);
  gpio_pull_up(26);
  adc_select_input(0);
  while(true){
    adc_select_input(0);
    for (int n=0; n<10; n++){
      ad = adc_read()*3300/4095;
      iir = (iir*9+ad)/10;
```

```
    }
    temp = 23+(476-iir)/2;
    Serial.print (iir);
    Serial.print (" mV ");
    Serial.print (temp);
    Serial.print (" °C ");
    Serial.print (analogReadTemp());
    Serial.println (" °C ");
    sleep_ms(5000);
  }
}

void loop() {
```

10 Регулируемый источник напряжения

Для испытаний электронных компонентов часто требуется точно регулируемое напряжение. Здесь выход ШИМ Pico используется вместе с фильтром нижних частот в качестве источника напряжения. Напряжение можно постепенно увеличивать или уменьшать с помощью двух кнопок.

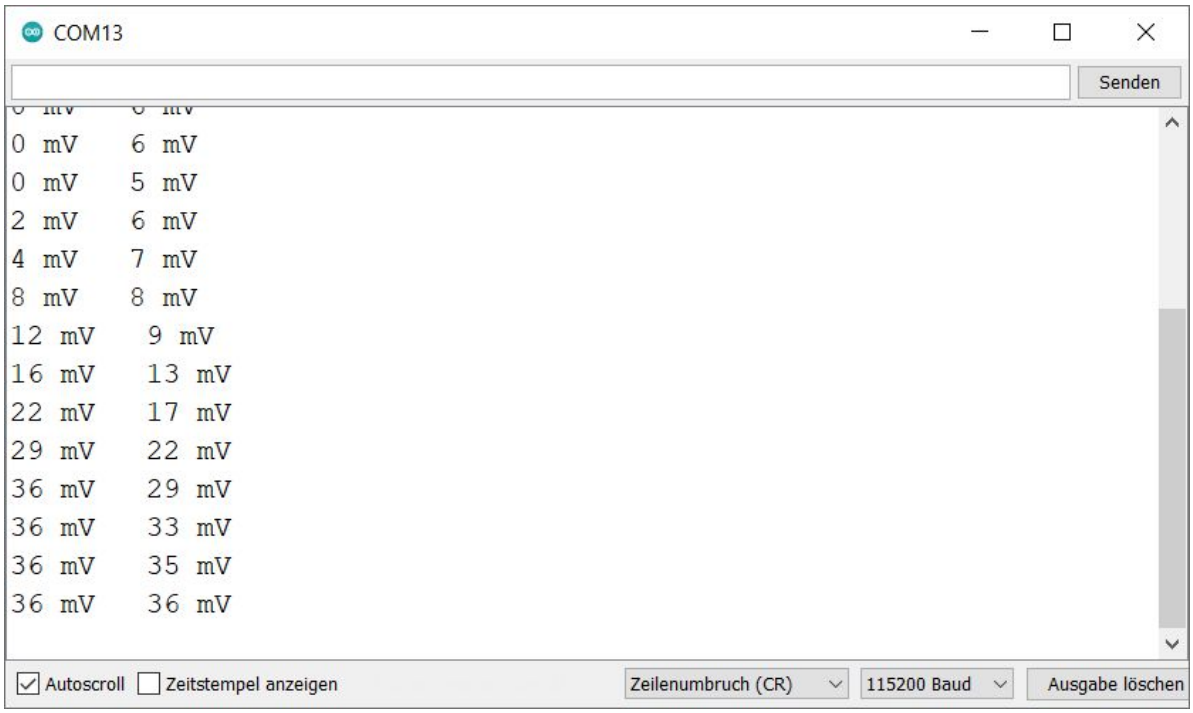



```
//Pico_ADpwm

#include "pico/stdlib.h"
#include "hardware/adc.h"
#include "hardware/pwm.h"

void setup() {
  int ad=0;
  int iir=0;
  int pwm=0;
  int n, d;
  Serial.begin(115200);
  adc_init();
  adc_gpio_init(26);
  adc_select_input(0);

  gpio_set_function(0, GPIO_FUNC_PWM);
  pwm_set_wrap(0, 4095);
```

```
  pwm_set_gpio_level(0, 0);
  pwm_set_enabled(0, true);
  gpio_pull_up(4);
  gpio_pull_up(8);

  while(true){
    for (n=0; n<50; n++){
      ad = adc_read();
      iir = (iir*9+ad)/10;
      sleep_ms(10);
    }
    Serial.print(pwm*3300/4095);
    Serial.print(" mV ");
    Serial.print(iir*3300/4095-5);
    Serial.println(" mV");
    if (gpio_get(4)==0) pwm-=d;
    if (pwm<0) pwm=0;
    if (gpio_get(8)==0) pwm+=d;
    pwm_set_gpio_level(0, pwm);
    d++;
    if ((gpio_get(4)==1) & (gpio_get(8)==1)) d=1;
  }
}
```

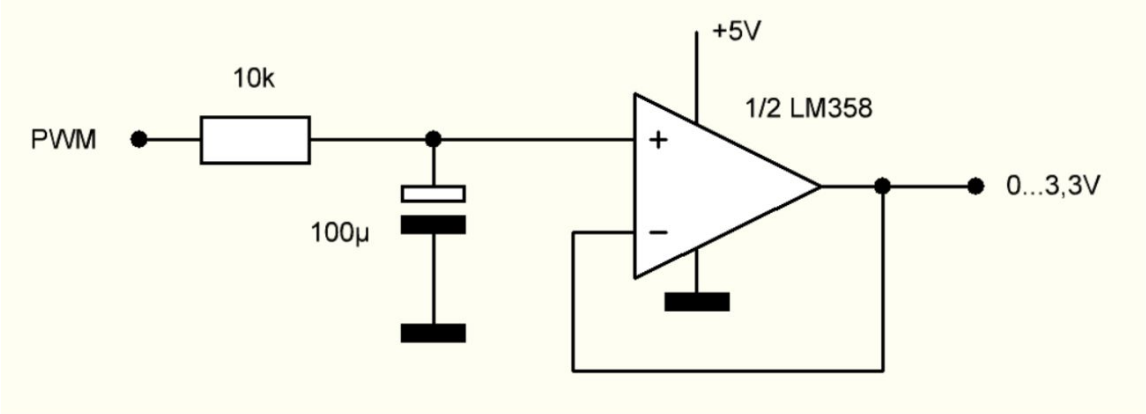
Поскольку АЦП имеет разрешение 12 бит, выход ШИМ также установлен в диапазоне 0...4095. В результате частота ШИМ составляет 125 МГц/4096 = 30,5 кГц. Таким образом, сигнал можно очень хорошо сгладить с помощью RC фильтра нижних частот с сопротивлением 10 кОм и 100 мкФ.

Две кнопки начинают с одиночных шагов, которые затем постепенно увеличиваются, так что можно было удобно устанавливать еще большие изменения напряжения. Отпустив кнопки, вы вернетесь к одному шагу.

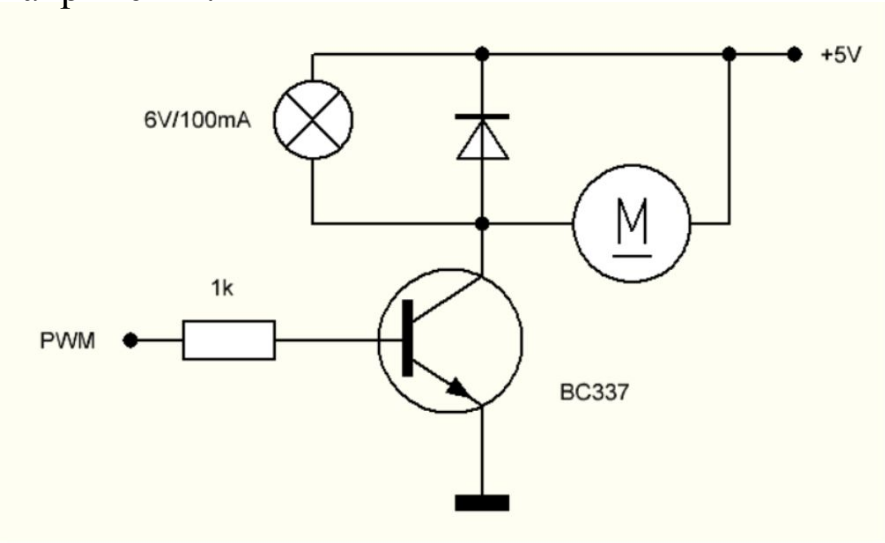


Последовательный монитор показывает заданное напряжение и измеренное напряжение. Тест показывает ошибку смещения около 5 мВ в нижнем конце шкалы. После изменения напряжения проходит около двух секунд, пока не будет достигнуто измеренное конечное значение. Это связано, с одной стороны, с постоянной времени RC-фильтра, равной одной секунде, а с другой стороны, со встроенным БИХ-фильтром, который должен предотвращать колебания показаний.

Сглаженный выходной сигнал имеет очень высокое сопротивление — 10 кОм, поэтому напряжение падает даже при небольших нагрузках. Это может быть преимуществом, когда необходимо исследовать чувствительные компоненты. Например, вы можете определить, при каком напряжении начинает загораться тот или иной светодиод. В других случаях желателен более высокий выходной ток. Следующий буферный усилитель с LM358 обеспечивает ток до 40 мА. Это можно даже использовать для привода небольших электродвигателей. Таким образом, управление ШИМ становится контролем скорости.



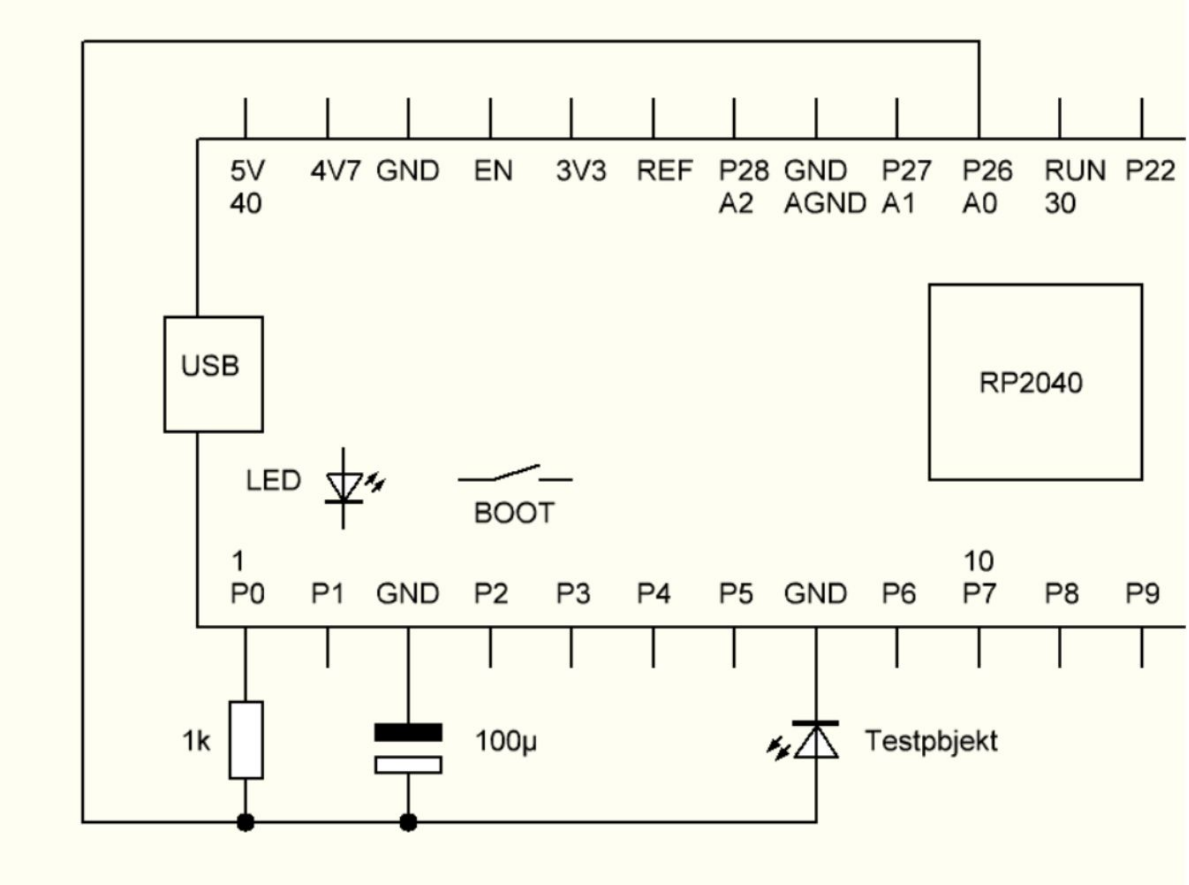
В других случаях можно обойтись без сглаживания и управлять силовым драйвером напрямую ШИМ-сигналом. Часто достаточно одного транзистора. Затем сигнал ШИМ можно использовать, например, для управления лампой накаливания или двигателем. При использовании индуктивных нагрузок необходимо использовать обратноходовой диод для подавления пиков индуктивного напряжения.



11 Регистратор характеристических кривых

Характеристические кривые представляют собой диаграммы U/I, описывающие поведение компонента. Напряжение на компоненте увеличивают и изменяют ток. Выход ШИМ с фильтром нижних частот сноваиспользуетсядляпостоянногонапряжения,которое автоматически проходит через него. Напряжение на компоненте измеряется напрямую. Ток получается из разницы между только что установленным напряжением холостого хода ШИМ и измеренным напряжением.

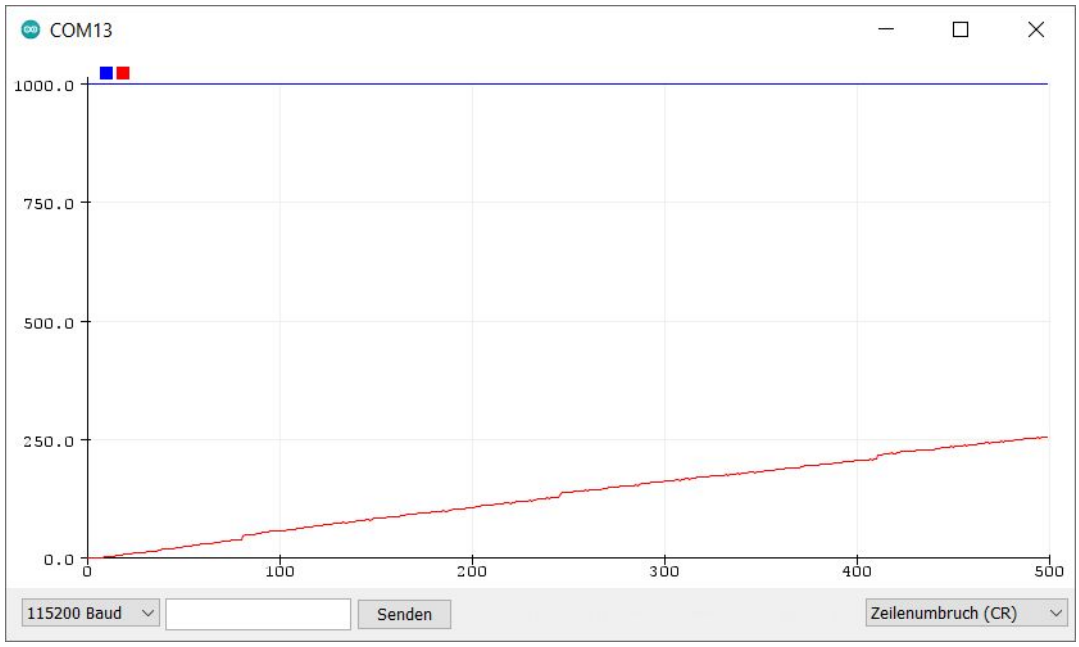
Чтобы кривая была плавной, измеренный сигнал проходит через БИХ-фильтр (IIR-фильтр). Ток может достигать почти 3,3 мА с целью с очень низким сопротивлением, но в других случаях остается намного меньшим. Для сравнения проведена вспомогательная линия на уровне 1 мА.



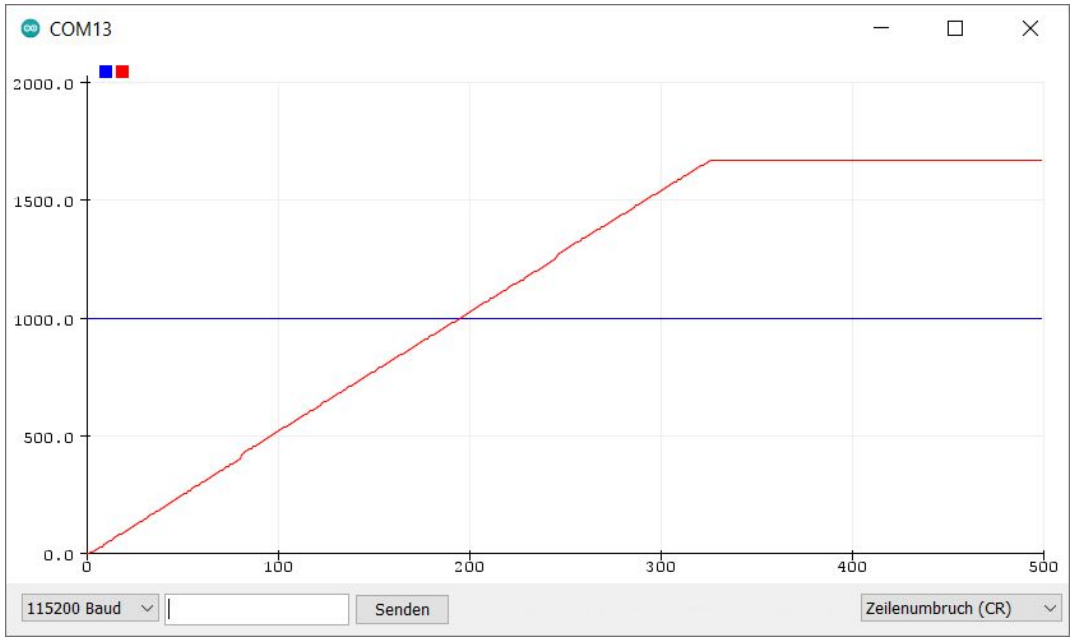
```
//Pico_ADpwm2 Characteristic curve

#include "pico/stdlib.h"
#include "hardware/adc.h"
#include "hardware/pwm.h"

void setup() {
  int ad=0;
  int iir=0;
  int pwm=0;
  int n, u, u2, i, i2, j, t;
  Serial.begin(115200);
  adc_init();
  adc_gpio_init(26);
  adc_select_input(0);

  gpio_set_function(0, GPIO_FUNC_PWM);
  pwm_set_wrap(0, 4095);
  pwm_set_gpio_level(0, 0);
  pwm_set_enabled(0, true);
  gpio_pull_up(4);
  gpio_pull_up(8);
  sleep_ms(1000);
  while(true){
    pwm_set_gpio_level(0, 0);
    sleep_ms(1000);
    u2=0;
    t=0;
    iir = 5;
    for (j=0; j<4096; j++){
      pwm_set_gpio_level(0, j);
      sleep_ms(2);
      for (n=0; n<5; n++){
        ad = adc_read();
        iir = (iir*3+ad)/4;
        sleep_us(500);
```

```
      }
     u=iir*3300/4095;
     if (u>2500) u=2500;
     if (u<2500) i=(j-iir)*3300/4095;
     if (i<0)i=0;
     u=u/5;
     if (u>u2){
      Serial.print(1000);
      Serial.print (" ");
      Serial.println(i);
      t++;
     }
     if(u>u2) u2=u;
    }
    pwm_set_gpio_level(0, 0);
    for (j=t; j<500; j++){
     Serial.print(1000);
     Serial.print (" ");
     Serial.println(i);
     sleep_ms(5);
    }
  }
}

void loop() {}
```

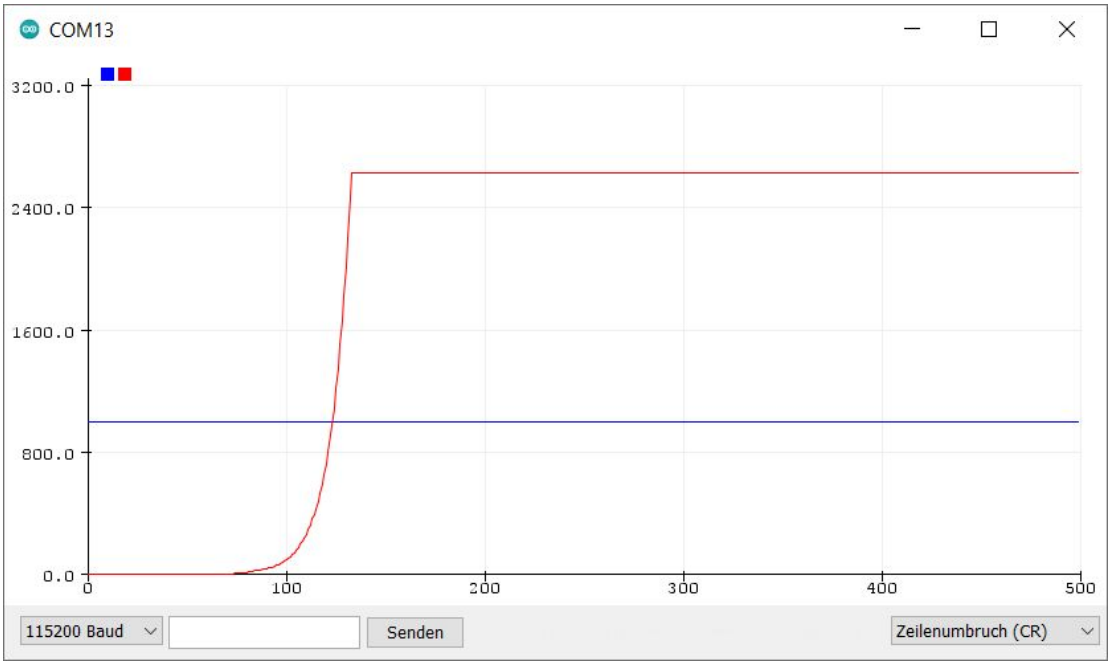
По оси X напряжение достигает 2,5 В, так что одно деление шкалы последовательного плоттера соответствует 0,5 В. Первое измерение показывает кривую резистора сопротивлением 10 кОм на холостом ходу. Видно, что ток возрастает линейно и достигает 250 мкА при 2,5 В. Небольшие скачкообразные отклонения от линейности на диаграмме обусловлены конечной точностью АЦП.



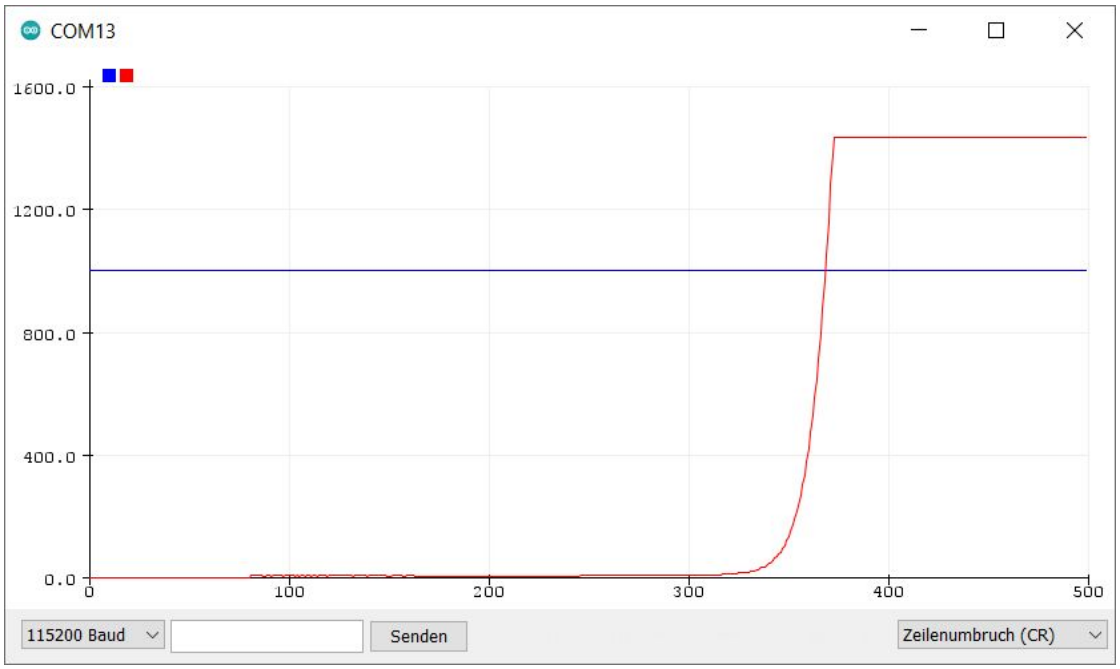
Измерение на резисторе сопротивлением 1 кОм также показывает линейное увеличение, при этом ток около 1 мА измеряется при напряжении 1 В. Характеристическая кривая расширяется примерно до 1,65 В и 1,65 мА, поскольку здесь делитель напряжения с двумя одинаковыми резисторами достигает максимума 3,3 В / 2. Горизонтальный участок больше не принадлежит самой характеристической кривой, а служит для заполнения серийного плоттера 500 измеренными значениями. Следующее измерение снова точно вписывается в окно.



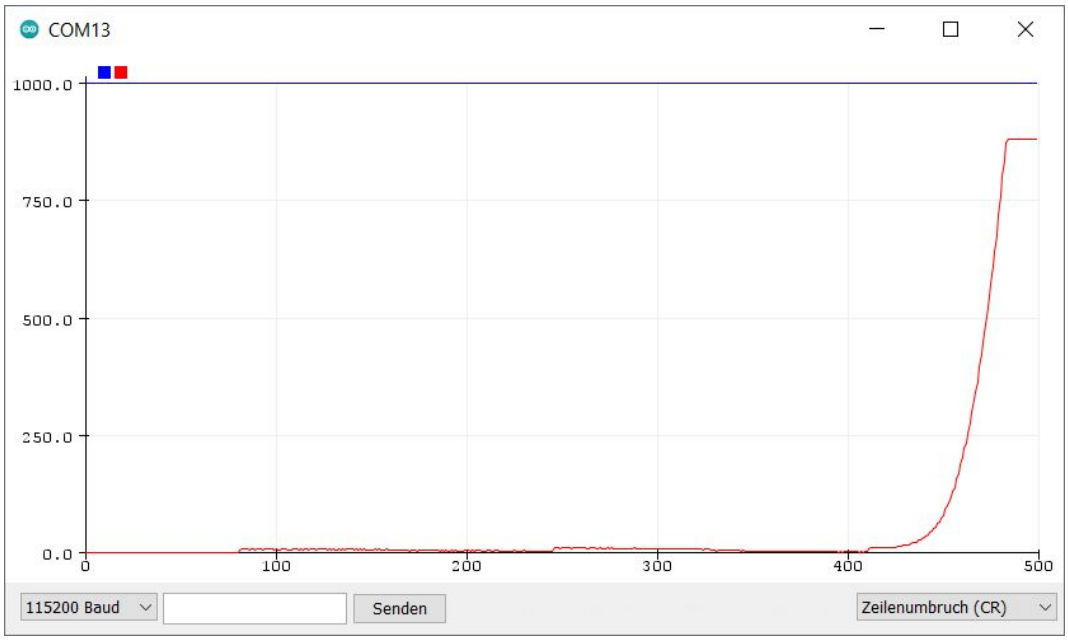
Характеристическая кривая диода 1N4148 показывает экспоненциальный рост тока, который в этом масштабе начинает расти примерно с 0,4 В от нулевой линии. Резкий подъем подтверждает впечатление, что прямое напряжение диода находится примерно в диапазоне от 0,6 до 0,7 В. Однако это вопрос масштаба. На схеме уже можно увидеть ток порядка 10 мкА примерно от 0,4 В.



Характеристическая кривая красного светодиода поразительно похожа на кривую кремниевого диода, хотя подъем сдвинут в сторону значительно более высоких напряжений. Едва заметный ток течет от величины напряжения ок. 1,6 В. При токе 1 мА прямое напряжение составляет ок. 1,8 В.

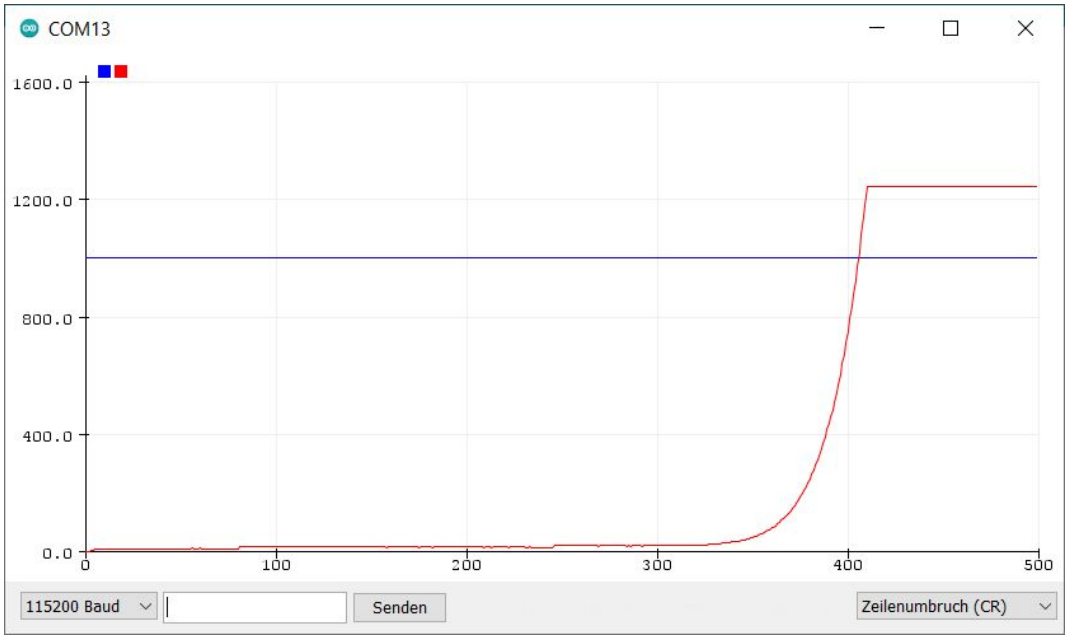


Еще более высокие напряжения до прибл. 2,4 В индицируются зеленым светодиодом.



Для исследования N-канального МОП-транзистора BS170 были соединены затвор и сток. Характеристическая кривая менее крутая, чем характеристика диода. При напряжении затвора 2 В ток стока около 1 мА.

По характеристической кривой можно оценить, что BS170 уже можно хорошо управлять с помощью управляющего напряжения 3,3 В.

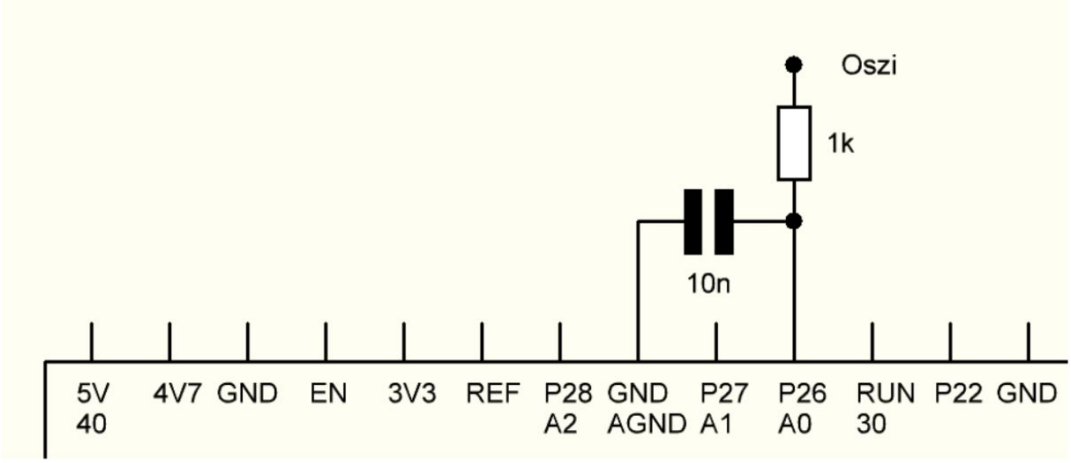


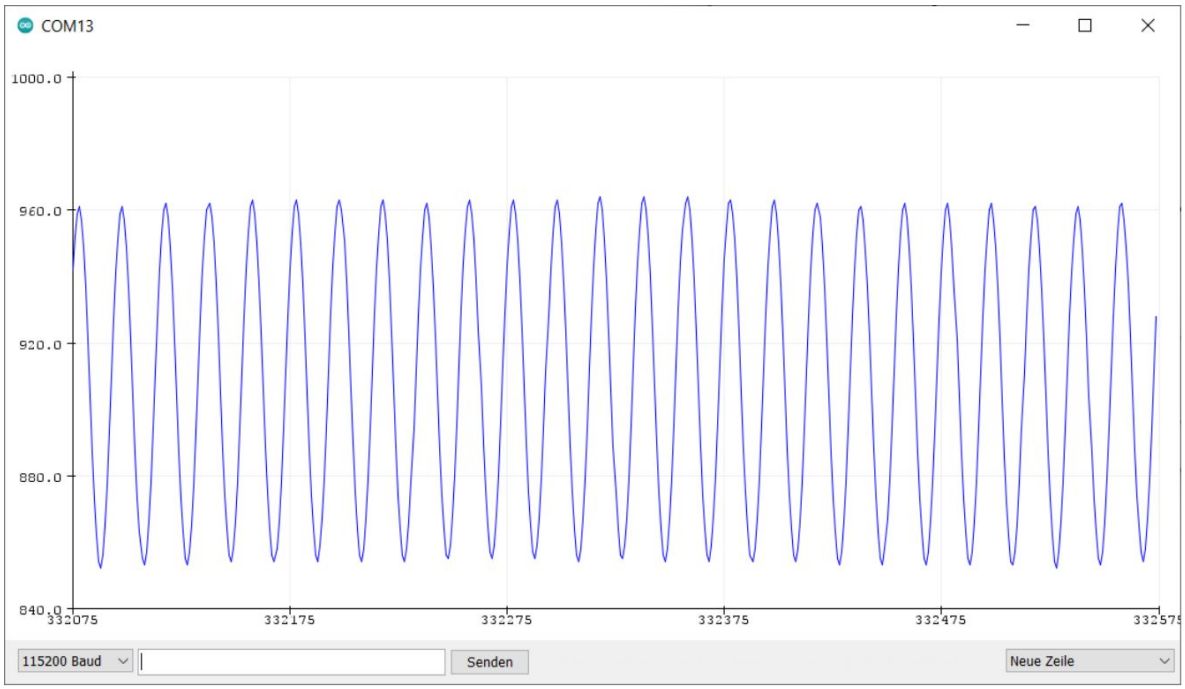
12 Простой осциллограф

Благодаря более быстрым измерениям и последовательному плоттеру можно реализовать простой осциллограф. Здесь снова использовалась синхронизация с микросекундным таймером, чтобы остальные задержки не складывались в установленный интервал в одну миллисекунду.

```
//Pico_AD3

#include "pico/stdlib.h"
#include "hardware/adc.h"

void setup() {
  uint32_t ad=0;
  uint32_t iir=0;
  Serial.begin(115200);
  adc_init();
  adc_gpio_init(26);
  //gpio_pull_up(26);
  adc_select_input(0);
  uint32_t t = time_us_32();
  while(true){
    ad = adc_read()*3300/4095;
    iir = (iir*3+ad)/4;
    Serial.println(iir);
    t+=1000;
    while (t>time_us_32());
  }
}

void loop() {
}
```

Кроме того, для получения более плавных осциллограмм был включен простой БИХ-фильтр (IIR-фильтр) нижних частот. Для каждого измерения только 25 % переменной iir определяется текущим измерением, а 75 % — предыдущими измерениями. Поведение этого фильтра соответствует поведению RC-фильтра нижних частот.



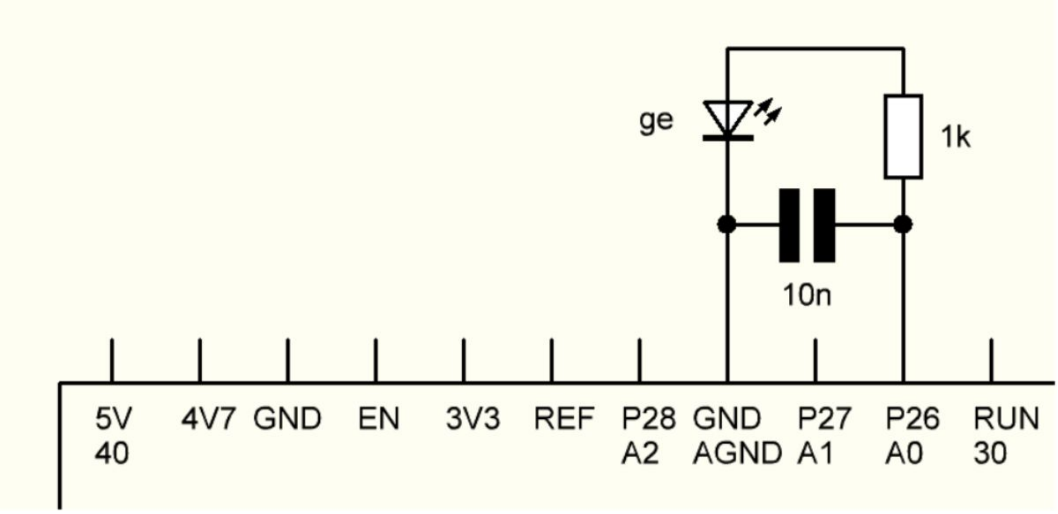


Этот результат измерения был получен путем прикосновения пальца к измерительному входу, в результате чего был измерен сигнал частотой 50 Гц. Кроме того, к GND был подключен конденсатор емкостью 10 нФ. Вместе с программным IR-фильтром это подавляет высокочастотные помехи.

При правильно установленном размере окна плоттера вы получаете пять частей шкалы по оси X, каждая из которых представляет 100 точек измерения, т.е. в данном случае 100 мс. Поскольку период сетевого напряжения составляет 20 мс, в каждом сегменте шкалы вы видите ровно 5 колебаний.

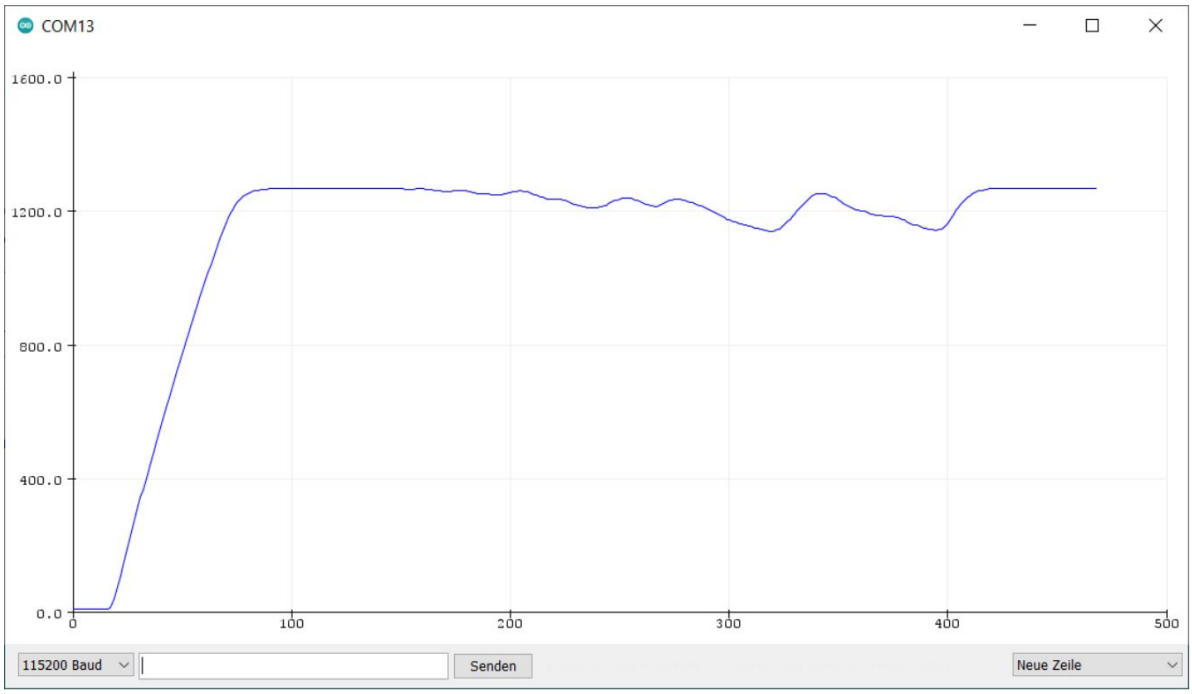
13 Измерение освещенности с помощью светодиода

Аналоговый вход Pico имеет чрезвычайно высокий импеданс. Это делает возможными измерения, которые невозможны с помощью обычных измерительных приборов с внутренним сопротивлением 10 МОм. В качестве фотодиода можно использовать обычный светодиод. В зависимости от цвета он выдает напряжение до 1,5 В и более, хотя максимальный ток находится лишь в диапазоне наноампер.



Для измерения использовался желтый светодиод. Программа осциллографа Pico_AD3 работала, замедлилась до периода выборки 100 мс.

```
t+=100000;
while (t>time_us_32());
```



Измерения проводились при нормальном комнатном освещении, когда светодиод был направлен на окно, улавливая естественный свет в пасмурную погоду. Начало измерения показывает рост прибл. 2 В/с. Отсюда при емкости 10 нФ можно получить фототок светодиода 20 нА. Напряжение возрастает примерно до 1,2 В и представляет собой логарифмическую меру освещенности. Колебания яркости, зафиксированные в среднем диапазоне, были вызваны частичными тенями, создаваемыми человеком, ходящим по комнате.

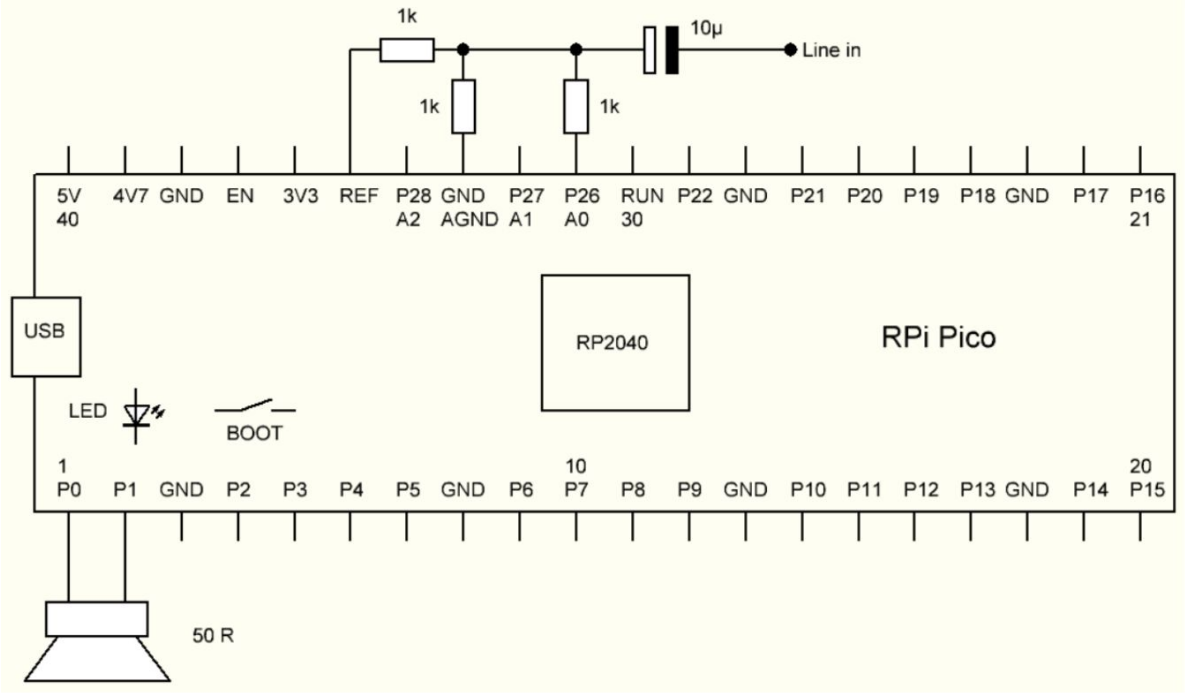
14 Усилитель класса D

Усилитель класса D отличается высоким КПД и низкими искажениями. Выходной каскад состоит из цифрового мостового усилителя. В состоянии ожидания оба выхода генерируют один и тот же сигнал ШИМ, поэтому между двумя соединениями громкоговорителя нет разницы потенциалов. При модуляции длины импульсов сдвигаются относительно друг друга.

Такой усилитель можно реализовать с помощью RPi Pico, если не требуется более высокая выходная мощность. Мостовой усилитель формируется из двух выходов ШИМ, принадлежащих одному блоку ШИМ, т.е. они находятся в фазе.

Сигнал AF поступает на аналоговый вход A0, при этом потенциал покоя устанавливается на половину диапазона измерения с помощью делителя напряжения. Измеренный сигнал напрямую управляет шириной импульса P0 и инвертированной шириной импульса P1. Громкоговоритель должен иметь относительно высокое сопротивление, предпочтительно 50 Ом. Для первоначальных экспериментов также можно использовать громкоговоритель сопротивлением 8 Ом с последовательным резистором сопротивлением 47 Ом. Как и порты, выходы ШИМ работают с настройкой 4 мА и, следовательно, выдают максимум 30 мА. К сожалению, изменить диапазоны тока в режиме ШИМ невозможно. При использовании громкоговорителя сопротивлением 50 Ом выходная мощность достигает 20 мВт.

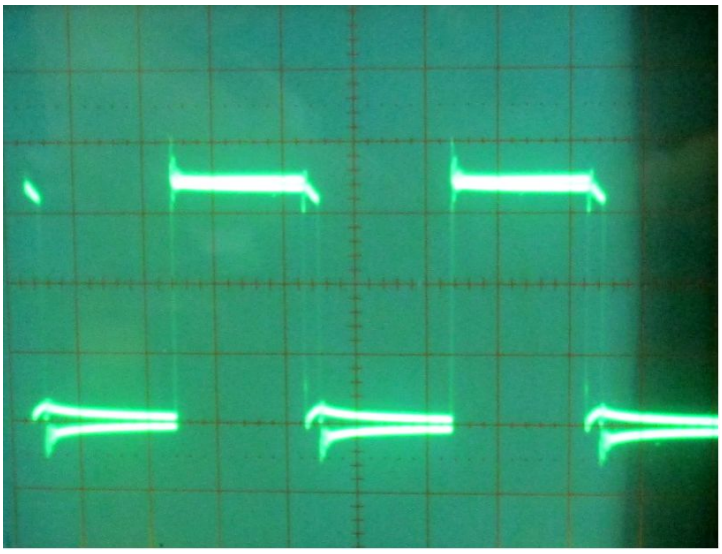
Разрешение измерения и сигнала ШИМ установлено на 10 бит, т.е. в диапазоне от 0 до 1023. Это обеспечивает высокую частоту дискретизации 122 кГц (125 МГц/1024 = 122 кГц). Делитель напряжения с двойным сопротивлением 1 кОм устанавливает центральное напряжение. Последовательный резистор сопротивлением 1 кОм на входе A0 предназначен для предотвращения перегрузки входа в случае возможной перегрузки.



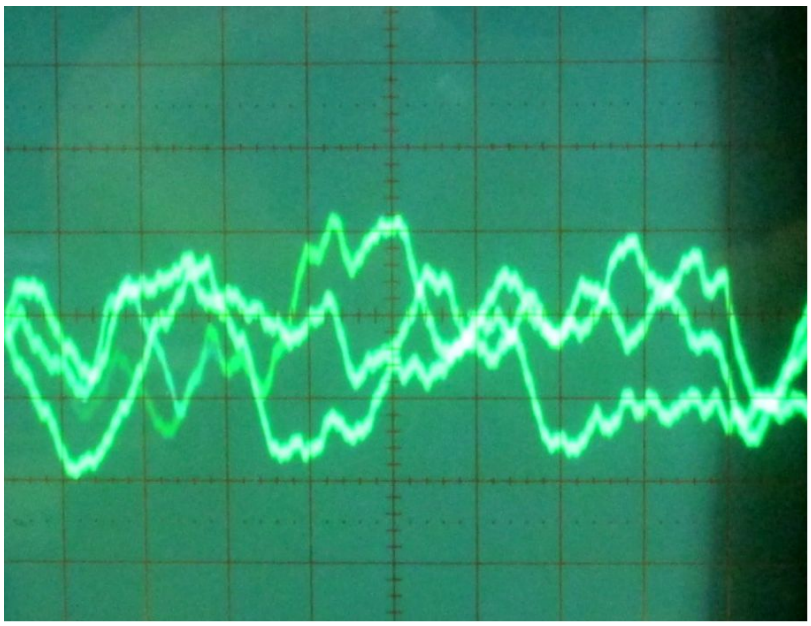

```
//Pico_PWM5 Class D amplifier
#include "pico/stdlib.h"
#include "hardware/pwm.h"
#include "hardware/adc.h"

void setup() {
}

void loop() {
}

void pwm_int() {
  pwm_clear_irq(0);
  int u =adc_read()/4;
  pwm_set_gpio_level(0, u);
  pwm_set_gpio_level(1, 1023-u);
}

void setup1() {
  adc_init();
```

```
  adc_gpio_init(26);
  adc_select_input(0);

  gpio_set_function(0, GPIO_FUNC_PWM);
  gpio_set_function(1, GPIO_FUNC_PWM);
  pwm_clear_irq(0);
  pwm_set_irq_enabled(0, true);
  irq_set_exclusive_handler(PWM_IRQ_WRAP, pwm_int);
  irq_set_enabled(PWM_IRQ_WRAP, true);
  pwm_set_wrap(0, 1023);
  pwm_set_gpio_level(0, 512);
  pwm_set_enabled(0, true);
}

void loop1() {
}
```

В программе используется функция прерывания, которая вызывается каждый раз при переполнении счетчика ШИМ. Таким образом, дискретизация входного сигнала АЦП и обновление ширины импульса ШИМ происходит синхронно с частотой ШИМ 122 кГц. Чтобы гарантировать, что плавный процесс не будет нарушен другими прерываниями, все выполняется во втором ядре процессора.



На осциллограмме виден общий нарастающий фронт обоих выходов и слегка смещенные спадающие фронты. Амплитуда импульса соответствует полному рабочему напряжению. В области спадающего фронта ток, текущий через громкоговоритель, можно распознать по наклонам падения напряжения на выходных внутренних резисторах, вызванному индуктивностью звуковой катушки.

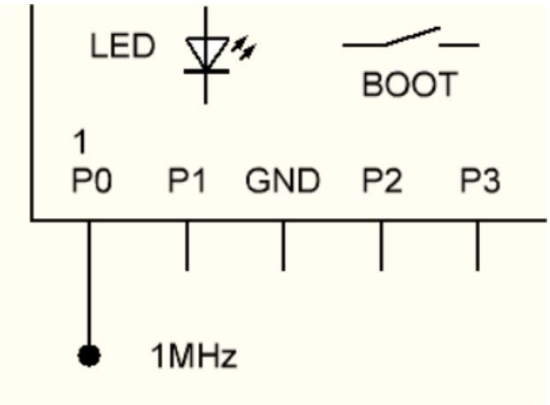


Усилитель обеспечивает хороший звук и удобное воспроизведение музыки при средней громкости в помещении. Вторая осциллограмма показывает усиленный сигнал ЗЧ на выходе ФНЧ сопротивлением 1 кОм и 100 нФ.

Использование прерывания не всегда приводит к абсолютно ровному времени. Если другие прерывания активны, они должны разделить время. Решение этой и других критичных по времени проблем заключается в использовании второго ядра процессора в RPi Pico. Для этого вам необходимо включить заголовочный файл pico/multicore.h. Функции setup() и Loop() тогда принадлежат первому ядру (Core0), функции setup1() и Loop1() принадлежат Core1, т.е. второму ядру.

15 Частотомер от 1 Гц до 1 МГц

Простой частотомер может наблюдать за состоянием порта и подсчитывать все положительные фронты в течение определенного временного окна в одну секунду. В первом эксперименте в качестве источника сигнала используется выход ШИМ с симметричным прямоугольным сигналом частотой 1 МГц. Выход ШИМ — это контакт P0, который также используется в качестве цифрового входа для частотомера. Таким образом, никакая внешняя схема не требуется. Проверенный микросекундный таймер снова используется для измерения времени стробирования.




```
//Pico_Freq1

#include "pico/stdlib.h"
#include "hardware/pwm.h"

void setup() {
  Serial.begin(115200);
  gpio_set_function(0, GPIO_FUNC_PWM);
  pwm_set_wrap(0, 124); //PWM 1 MHz, 50%
  pwm_set_gpio_level(0, 63);
  //pwm_set_wrap(0, 1249); //PWM 100 kHz, 50%
  //pwm_set_gpio_level(0, 639);
  pwm_set_enabled(0, true);
```

```
  while (true) {
    uint32_t f = 0;
    bool in_old = 0;
    while (!gpio_get(0));
    while (gpio_get(0));
    uint32_t t = time_us_32() + 1000000;
    while (t>time_us_32()){ //1 MHz measure
      bool in_new =(gpio_get(0));
      if (in_new>in_old) f++;
      in_old=in_new;
    }
    Serial.print(f);
    Serial.println(" Hz ");
  }
}

void loop() {
}
```

Последовательный монитор показывает слегка колеблющийся результат — 990,021 МГц. Это означает, что отображаемая частота занижена примерно на 1 %.

```
990021 Hz
990021 Hz
990020 Hz
```

Если выход ШИМ установлен на 100 кГц, сохраняется погрешность измерения 1 %. Уже при 10 кГц ошибка исчезает полностью.

```
10000 Hz
10000 Hz
10000 Hz
```

Наблюдения показывают, что на пико идет другой процесс, который съедает столько времени, что 1% положительных фронтов считывается на более высоких частотах. Задержки должны быть менее длины одного импульса сигнала частотой 10 кГц, т.е. менее 50 нс. Подозрение падает на интерфейс USB, который должен работать постоянно.

Решение проблемы снова заключается в использовании второго ядра процессора в RPi Pico. Виртуальный последовательный интерфейс инициализируется в первом ядре. Фактическое измерение частоты перенесено на второе ядро. Доступ к интерфейсу с Serial.print есть, но передача данных происходит в Core0 (в первом ядре).

```
//Pico_Freq2

#include "pico/stdlib.h"
#include "hardware/pwm.h"
#include "pico/multicore.h"

void setup() {                //Core0
  Serial.begin(115200);
}

void loop() {}

void setup1() {               //Core1
  gpio_set_function(0, GPIO_FUNC_PWM);
  pwm_set_wrap(0, 124); //PWM 1 MHz, 50%
  pwm_set_gpio_level(0, 63);
  pwm_set_enabled(0, true);

  while (true) {
    uint32_t f = 0;
    bool in_old = 0;
    while (!gpio_get(0));
    while (gpio_get(0));
    uint32_t t = time_us_32() + 1000000;
    while (t>time_us_32()){ //1 MHz measure
      bool in_new =(gpio_get(0));
      if (in_new>in_old) f++;
      in_old=in_new;
    }
    Serial.print(f);
    Serial.println(" Hz ");
```

```
  }
}

void loop1() {}
```

Результат радует: при ШИМ-сигнале 1 МГц отображается ровно 1000 000 Гц. Обычные частотомеры показывают колебания в последнем десятичном знаке, поскольку окно измерения не начинается в фазе с входным сигналом. Однако здесь перед началом измерения ожидается положительный фронт входного сигнала. Это синхронизирует начало, в результате чего дисплей становится постоянным.

```
1000000 Hz
1000000 Hz
1000000 Hz
```

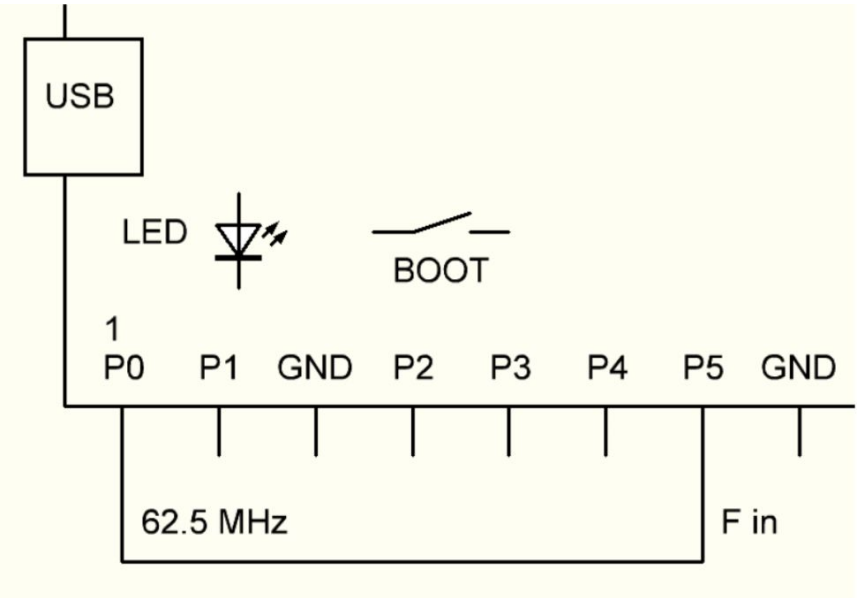
16 Частотомер от 1 кГц до 65 МГц

Для более высоких частот запроса состояний порта уже недостаточно, поэтому необходим аппаратный счетчик. Здесь блок ШИМ 2 (срез 2) используется как 16-битный счетчик. Канал B также может считать внешние часы вместо системных часов. Теперь он подключен к P5.

С помощью pwm_get_default_config() вы получаете структуру по умолчанию со всеми настройками модуля ШИМ. С помощью config_set_clkdiv_mode настройка изменяется таким образом, что внешние импульсы подсчитываются на входе B по нарастающим фронтам. Эта конфигурация затем назначается модулю ШИМ 2 с помощью pwm_init, где параметр false указывает, что выход ШИМ не требуется. Вместо этого на этот раз показания счетчика считываются с помощью pwm_get_counter.

```
//Pico_Freq3
#include "pico/stdlib.h"
#include "hardware/pwm.h"
#include "pico/multicore.h"

void setup() {
  Serial.begin(115200);
}

void loop() {}

void setup1() {
  gpio_set_function(0, GPIO_FUNC_PWM);
  pwm_set_wrap(0, 1); //PWM 62.5 MHz
  pwm_set_gpio_level(0, 1);
  pwm_set_enabled(0, true);

  while (true) {
    uint32_t f = 0;
    pwm_config cfg = pwm_get_default_config();
    pwm_config_set_clkdiv_mode(&cfg, PWM_DIV_B_RISING);
```

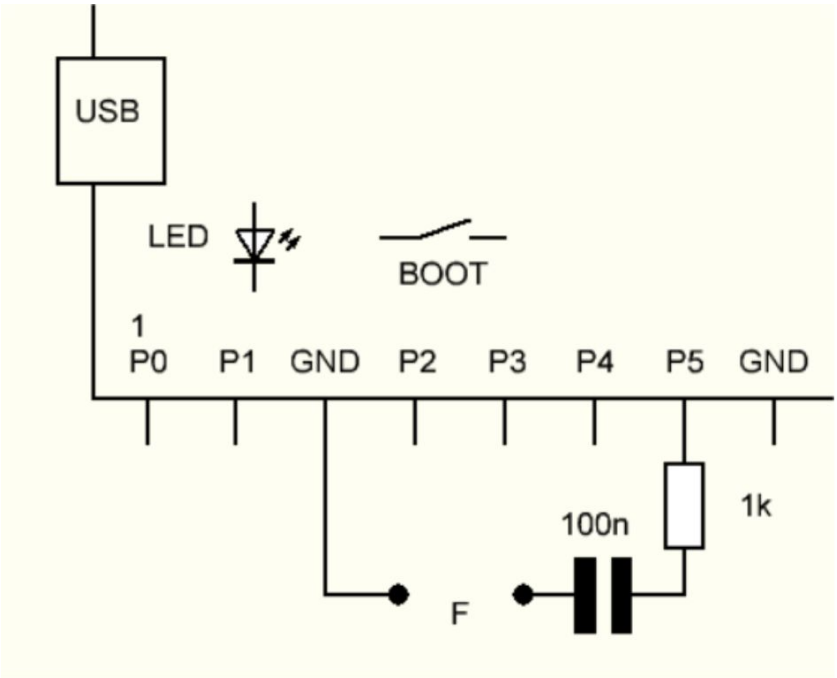
```
    pwm_init(2, &cfg, false);
    gpio_set_function(5, GPIO_FUNC_PWM);
    uint32_t t = time_us_32()+2;
    while (t>time_us_32());
    pwm_set_enabled(2, true);
    t += 1000;
    while (t>time_us_32());
    pwm_set_enabled(2, false);
    f= pwm_get_counter(2);

    Serial.print(f);
    Serial.println(" kHz ");
  }
}

void loop1() {}
```



Опять же, выход ШИМ на порту GP0 используется в качестве источника сигнала. Здесь генерируется максимально возможная частота ШИМ 62,5 МГц. В качестве базы времени используется микросекундный таймер. Чтобы гарантировать, что измерение начнется точно в начале микросекунды, заранее ожидается начало второй микросекунды. Еще через 1000 мкс измерение заканчивается. Результат измерения подтверждает частоту измерения 62500 кГц с обычными колебаниями на последней цифре.

62500 kHz
62499 kHz
62501 kHz

GP5 теперь можно использовать в качестве входа для универсального счетчика. Для защиты входа от отрицательного или слишком высокого напряжения сигнала резистор и конденсатор следует соединить последовательно.

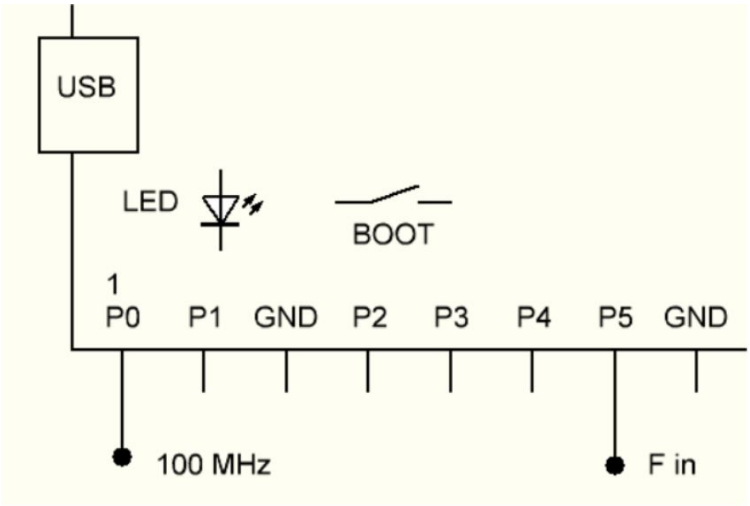


17 Частотомер от 1 Гц до 100 МГц

Счетчик ШИМ имеет ширину 16 бит и поэтому может считать только до 65535. Для еще большего разрешения необходимо обнаруживать переполнение счетчика. Здесь для этой цели используется прерывание ШИМ. При каждом переполнении увеличивается старшее слово f_hi. Время стробирования теперь можно увеличить до 1 000 000 мкс. На этот раз Rpi Pico разогнан до 200 МГц. Следовательно, на вывод P0 можно вывести тестовый сигнал частотой 100 МГц.

```
//Pico_Freq4

#include "pico/stdlib.h"
#include "hardware/pwm.h"
#include "pico/multicore.h"
uint32_t f_hi;

void setup() {
  set_sys_clock_khz(200000, true);
  Serial.begin(115200);
}

void loop() {}

void pwm_int() {
  pwm_clear_irq(2);
  f_hi++;
}

void setup1() {
  uint32_t f = 0;
  gpio_set_function(0, GPIO_FUNC_PWM);
  pwm_set_wrap(0, 1); //PWM 62.5 MHz
  pwm_set_gpio_level(0, 1);
  pwm_set_enabled(0, true);
```

```
  while (true) {
    pwm_config cfg = pwm_get_default_config();
    pwm_config_set_clkdiv_mode(&cfg, PWM_DIV_B_RISING);
    pwm_init(2, &cfg, false);
    gpio_set_function(5, GPIO_FUNC_PWM);
    pwm_set_irq_enabled(2, true);
    irq_set_exclusive_handler(PWM_IRQ_WRAP, pwm_int);
    irq_set_enabled(PWM_IRQ_WRAP, true);
    f_hi=0;
    uint32_t t = time_us_32()+2;
    while (t>time_us_32());
    pwm_set_enabled(2, true);
    t += 1000000;
    while (t>time_us_32());
    pwm_set_enabled(2, false);
    f= pwm_get_counter(2);
    f += f_hi<<16;
    Serial.print(f);
    Serial.println(" Hz ");
  }
}

void loop1() {}
```



При прямом соединении P0 и P5 тестовая частота 100 МГц измеряется с высокой точностью. Однако абсолютная точность ниже, поскольку зависит от допуска кварцевого генератора в пико. Если доступен подходящий эталонный кварц, счетчик можно откалибровать, установив время стробирования 1 000 000 мкс.

99999998 Hz
1000000 Hz
99999997 Hz

18 Генератор сигналов от 7 Гц до 50 МГц

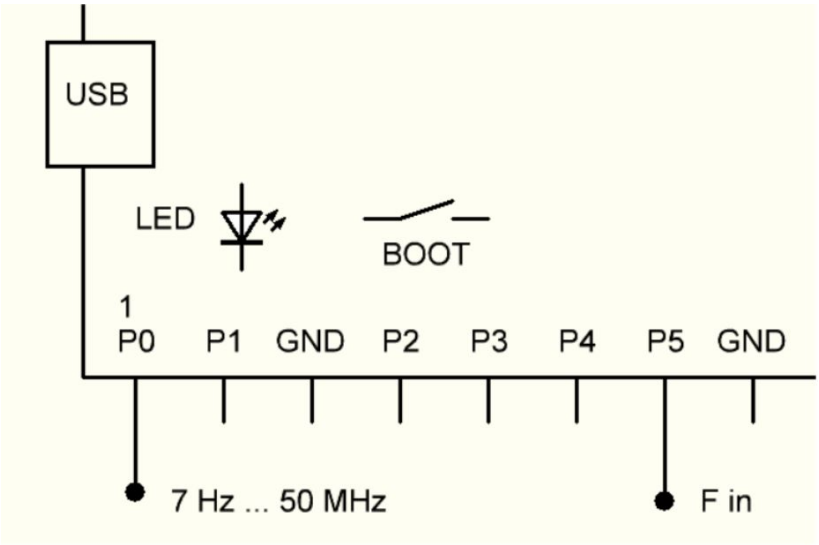
Этот генератор прямоугольных импульсов использует выход ШИМ на выводе P0. Частота устанавливается по команде с последовательного монитора в Гц или в кГц. С помощью f100 выбирается частота 100 Гц. F100 же дает 100 кГц. Выходная частота получается путем деления частоты системы на целочисленный делитель. Чтобы обеспечить плавные частоты, такие как 10 МГц или 50 МГц, частота системы устанавливается на 100 МГц.

Наименьшая частота выхода ШИМ раньше составляла 125 МГц/65536 = 1,907 кГц, а теперь будет 1,525 кГц. Однако еще есть неиспользованный ранее прескалер шириной 8 бит. В целях плавных делителей при необходимости устанавливается коэффициент делителя 10, 100 или 250. Таким образом можно снизить частоту до 7 Гц. Максимальная частота составляет 50 МГц с коэффициентом делителя 2.

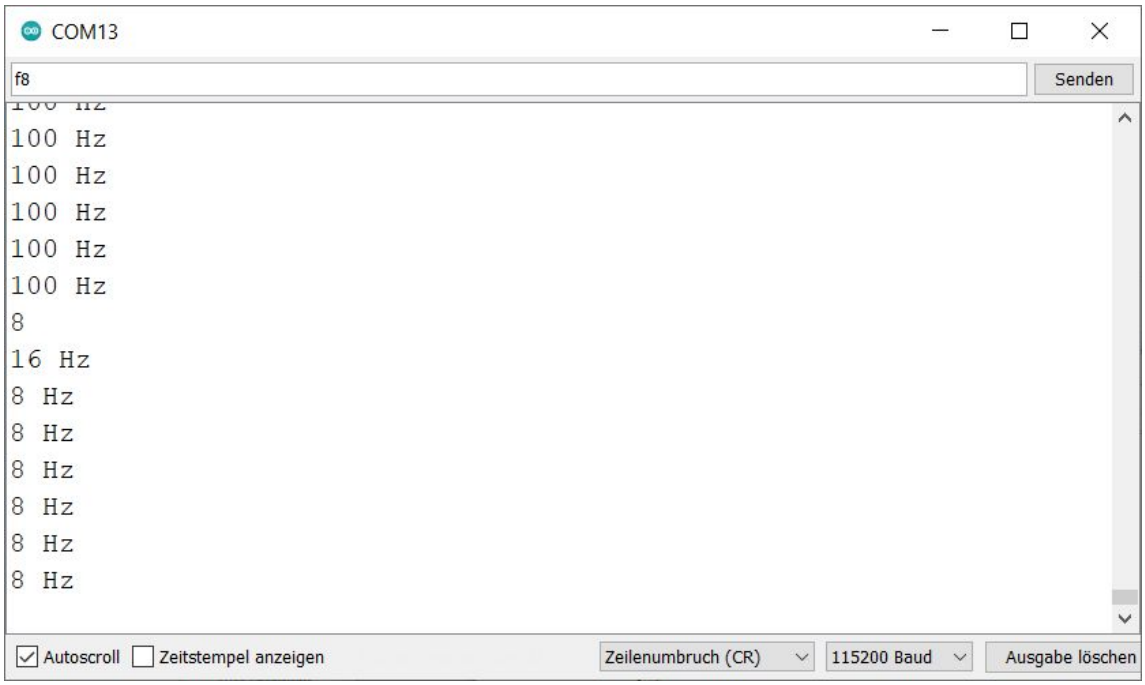
```
//Pico_PWM4 8 Hz ... 50 MHz

#include "pico/stdlib.h"
#include "hardware/pwm.h"
#include "pico/multicore.h"

uint32_t f_hi =0;

void setup() {
  set_sys_clock_khz(100000, true);
  Serial.begin(115200);
  while(1){
    if (Serial.available()) {
      char ch = Serial.read();
      uint32_t f = Serial.parseInt();
      if (ch == 70) f*=1000; //F
      if (ch == 102) f*=1; //f
      Serial.println (f);
      ch = Serial.read();
```

```
      uint16_t pre = 1;
      if (f<1908) pre = 10;
      if (f<191) pre = 100;
      if (f<20) pre = 250;
      gpio_set_function(0, GPIO_FUNC_PWM);
      pwm_config cfg = pwm_get_default_config();
      pwm_config_set_clkdiv_int (&cfg, pre);
      pwm_init(0, &cfg, true);
      uint32_t wrap = 100000000/pre/f;
      pwm_set_wrap(0, wrap-1);
      pwm_set_gpio_level(0, wrap/2);
      pwm_set_enabled(0, true);
    }
  }
}

void pwm_int() {
  pwm_clear_irq(2);
  f_hi++;
}

void loop() {}

void setup1() {
  uint32_t f = 0;
  while (true) {
    pwm_config cfg = pwm_get_default_config();
    pwm_config_set_clkdiv_mode(&cfg, PWM_DIV_B_RISING);
    pwm_init(2, &cfg, false);
    gpio_set_function(5, GPIO_FUNC_PWM);
    pwm_set_irq_enabled(2, true);
    irq_set_exclusive_handler(PWM_IRQ_WRAP, pwm_int);
    irq_set_enabled(PWM_IRQ_WRAP, true);
    f_hi=0;
    uint32_t t = time_us_32()+2;
    while (t>time_us_32());
    pwm_set_enabled(2, true);
```

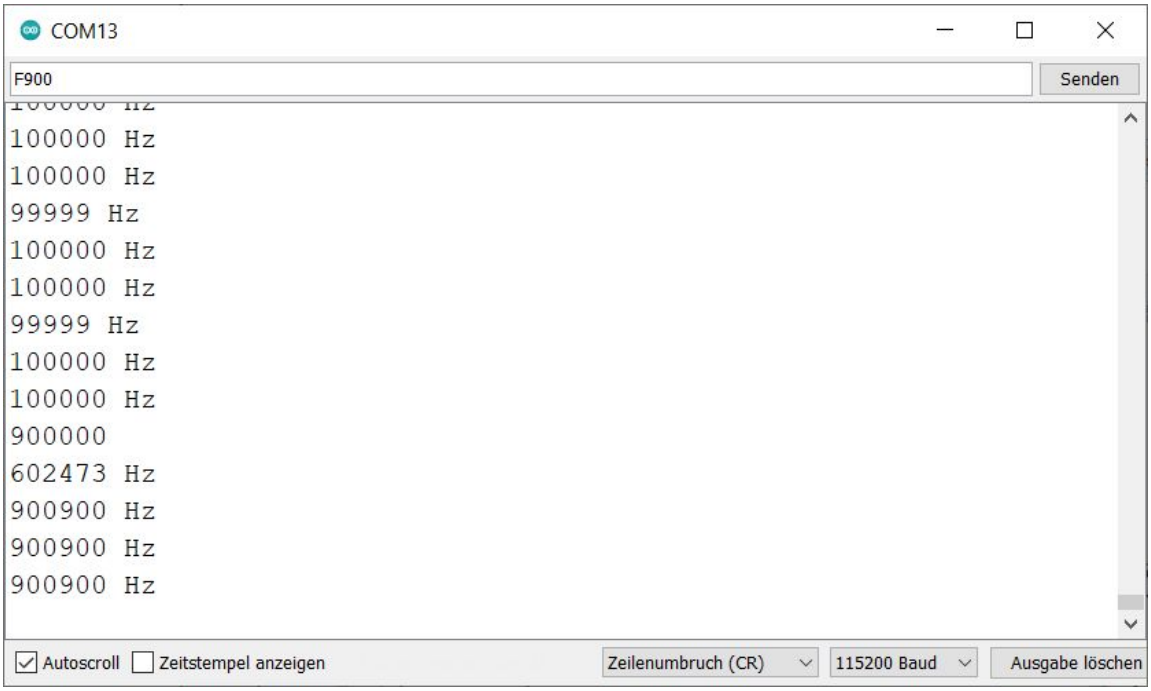
```
    t += 1000000;
    while (t>time_us_32());
    pwm_set_enabled(2, false);
    f= pwm_get_counter(2);
    f += f_hi<<16;
    Serial.print(f);
    Serial.println(" Hz ");
  }
}

void loop1() {}
```



В программу также входит частотомер высокого разрешения. При прямом соединении выхода сигнала и входа счетчика можно проверить заданный сигнал. При вводе типа f8 сначала отображается желаемая частота в Гц. После небольшой задержки отображается измеренная частота.



Частота, подобная 100 кГц, генерируется с высокой точностью, поскольку она обходится плавным делителем 1000. С другой стороны, если вы хотите сгенерировать 900 кГц, возникает ошибка округления 900 Гц с результатом 900,9 кГц. Самые высокие плавные частоты — 10 МГц, 20 МГц и 50 МГц.

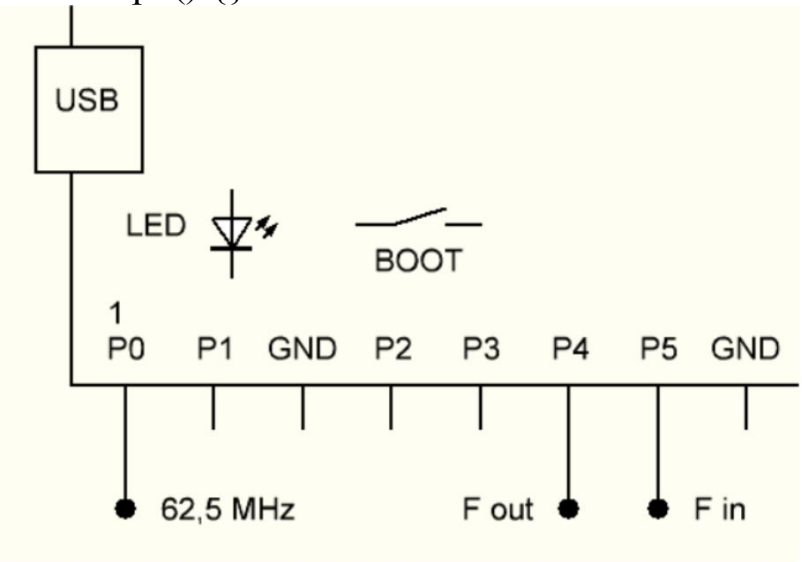


19 Делитель частоты

На этот раз блок ШИМ 2 инициализируется с внешним входом на P5 и выходом ШИМ на P4. Ширина счета устанавливается делителем = 100, а ширина импульса — 50%. Это дает вам универсальный делитель частоты со входом P5 и выходом P4. Коэффициент делителя можно установить в диапазоне от 2 до 32 768.

```
//Pico_PWM4

#include "pico/stdlib.h"
#include "hardware/pwm.h"
#include "pico/multicore.h"

void setup() {
  Serial.begin(115200);
}

void loop() {}

void setup1() {
  uint16_t divisor = 100;
  gpio_set_function(0, GPIO_FUNC_PWM);
  pwm_set_wrap(0, 1); //PWM 62.5 MHz
  pwm_set_gpio_level(0, 1);
  pwm_set_enabled(0, true);

  pwm_config cfg = pwm_get_default_config();
  pwm_config_set_clkdiv_mode(&cfg, PWM_DIV_B_RISING);
  pwm_init(2, &cfg, true);
  gpio_set_function(5, GPIO_FUNC_PWM);
  gpio_set_function(4, GPIO_FUNC_PWM);
  pwm_set_wrap(2, divider-1);
  pwm_set_gpio_level(4, divider/2);
  pwm_set_enabled(2, true);
}
```

void loop1() {}

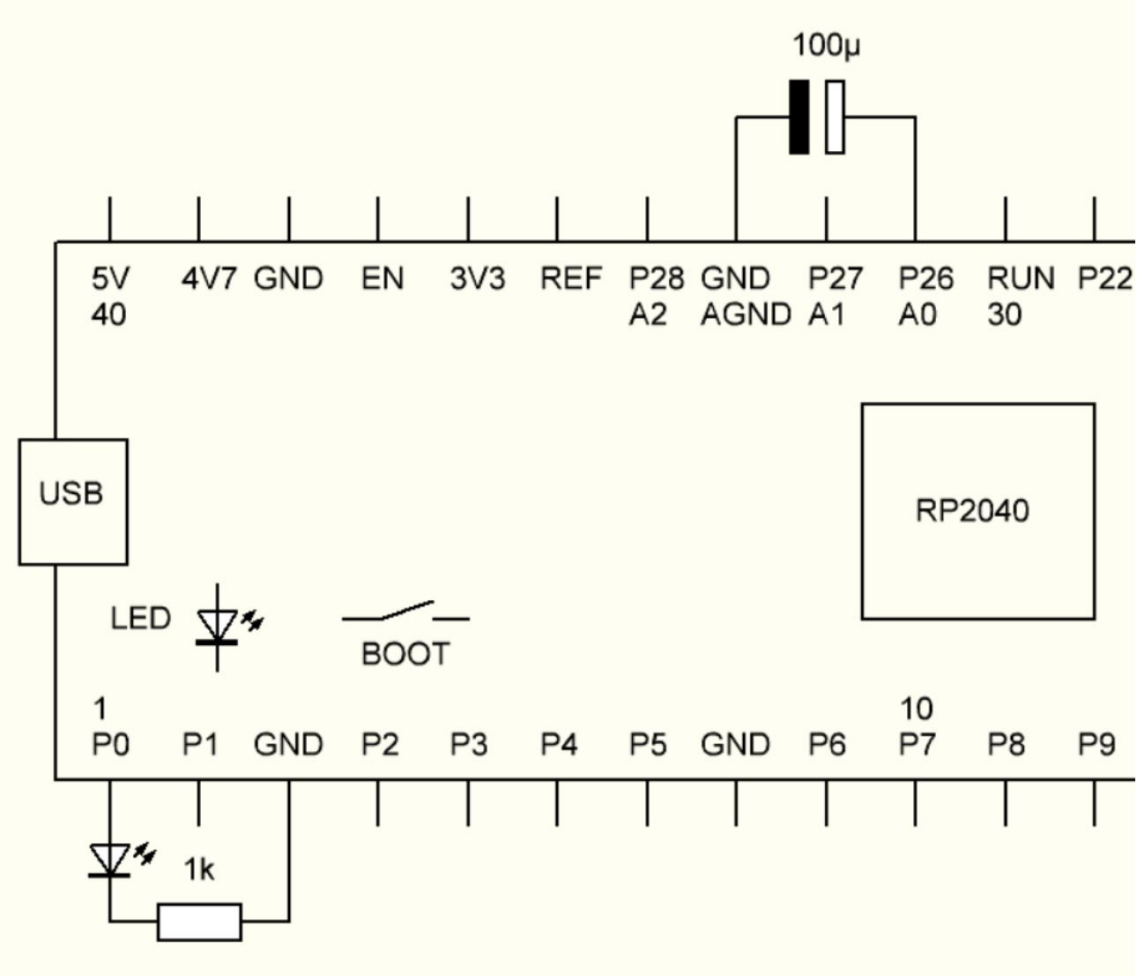


Программируемый делитель можно использовать в качестве прескалера для частотомера, который может измерять только более низкие частоты. Некоторые цифровые вольтметры содержат такой ограниченный счетчик. С помощью соответствующего прескалера они также могут измерять ВЧ-сигналы. В целях тестирования на порт P0 снова подается прямоугольный сигнал частотой 62,5 МГц. Если подключить его к P5, то на P4 появится сигнал 625 кГц.

20 RC-генераторы

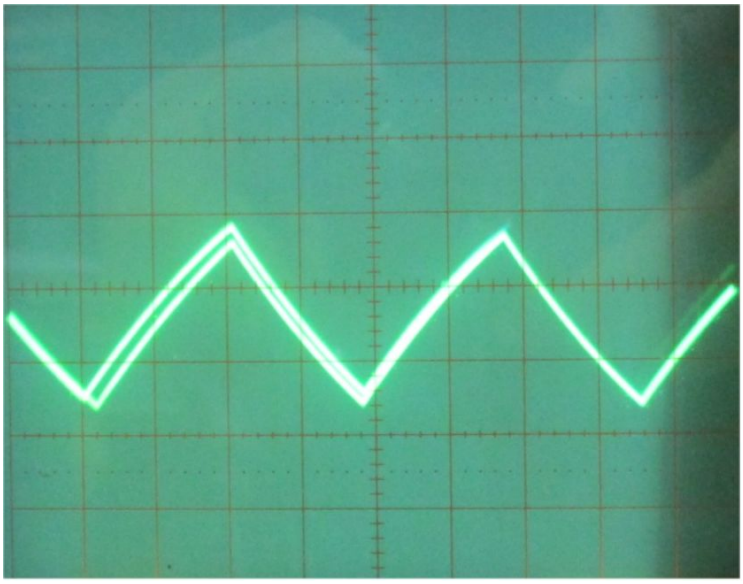
Программа Pico_RC1 реализует НЧ RC-генератор, который содержит электролитический конденсатор емкостью 100 мкФ в качестве единственного внешнего компонента. Необходимые резисторы являются внутренними и состоят из подтягивающих и понижающих резисторов порта P26.

Программа основана на функции модуля таймера NE555 и переключается между зарядкой и разрядкой при 1/3 и 2/3 рабочего напряжения. Для измерения напряжения конденсатора используется АЦП на выводе A0.



```
//Pico_RC1

#include "pico/stdlib.h"
#include "hardware/adc.h"

void setup() {
  adc_init();
  adc_gpio_init(26);
  gpio_pull_up(26);
  adc_select_input(0);
  gpio_init(0);
  gpio_set_dir(0, true);
  while(true){
    if (adc_read() > 2730){
      gpio_pull_down(26);
      gpio_put(0, 0);
    }
    if (adc_read() < 1365){
       gpio_pull_up(26);
       gpio_put(0, 1);
    }
  }
}

void loop() {
}
```

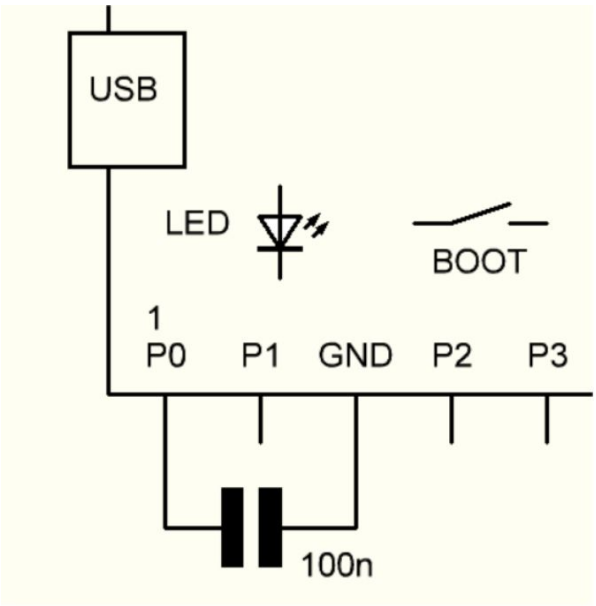
Программа одновременно управляет выходом D0, к которому подключен светодиод с последовательным резистором. Теперь он мигает с частотой около 0,3 Гц. Период колебаний пропорционален емкости. Следовательно, вы можете генерировать более высокие частоты с помощью конденсаторов меньшей величины. При конденсаторе емкостью всего 100 нФ достигается частота около 300 Гц. На осциллограмме представлена кривая напряжения на конденсаторе.



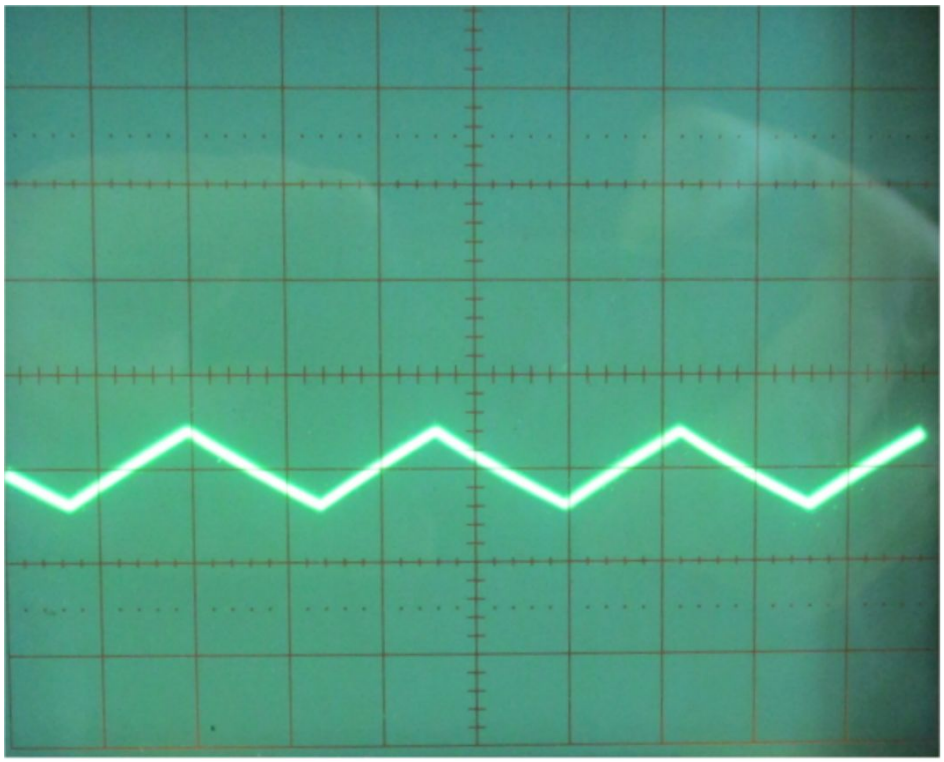
Даже без АЦП и всего с одним портом можно сформировать простой RC-генератор. Поскольку после сброса каждый порт работает как вход с гистерезисом, дальнейшая инициализация не требуется. Достаточно запросить вход и переключиться между повышением и понижением. Теперь вы можете подключить любой конденсатор между P0 и GND. Конденсатор емкостью 100 нФ обеспечивает частоту около 0,6 кГц из-за меньшего гистерезиса, составляющего около 0,4 В.

```
//Pico_RC2 RC generator P0
#include "pico/stdlib.h"

void setup() {}
void loop() {}

void setup1() {
  while (true) {
    if (f++; gpio_pull_down(0);
    else gpio_pull_up(0);
  }
}
```

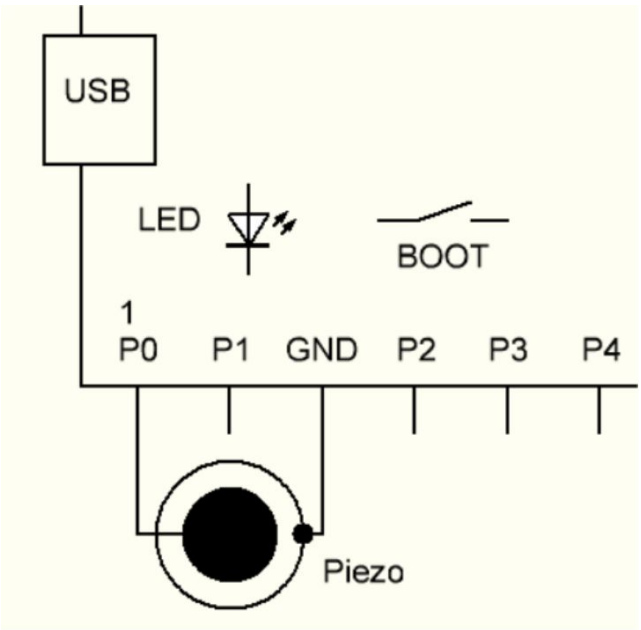
```
void loop1() {}
```

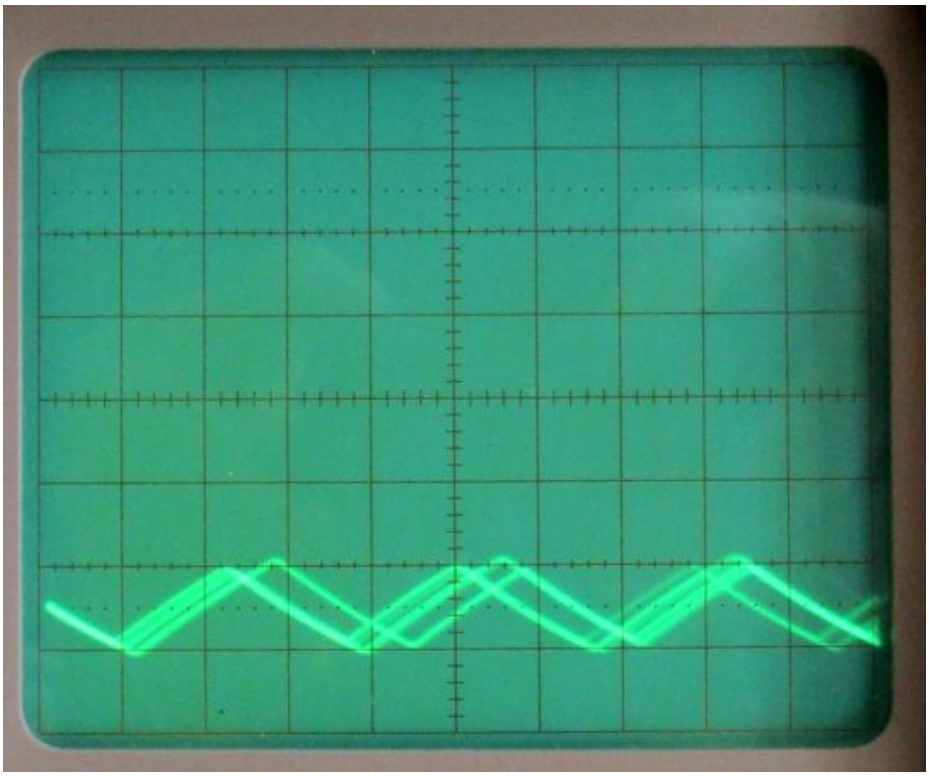


На осциллограмме показано напряжение сигнала на конденсаторе 100 нФ. Сигнал очень чистый и не содержит заметного дрожания фазы, поскольку контур управления работает на втором ядре и поэтому не подвергается воздействию других процессов. Температурную зависимость керамического конденсатора очень хорошо видно на осциллографе. Если прикоснуться к конденсатору, он нагревается и частота увеличивается.



Вместо конденсатора также можно использовать пьезопреобразователь, который тогда излучает тон около 1 кГц. Пьезодиск также является конденсатором и имеет емкость около 20 нФ. Свойства этого конденсатора изменяются при касании или отражении звука. Немного потренировавшись, вы сможете воспроизводить звуки разной частоты или даже простые мелодии.

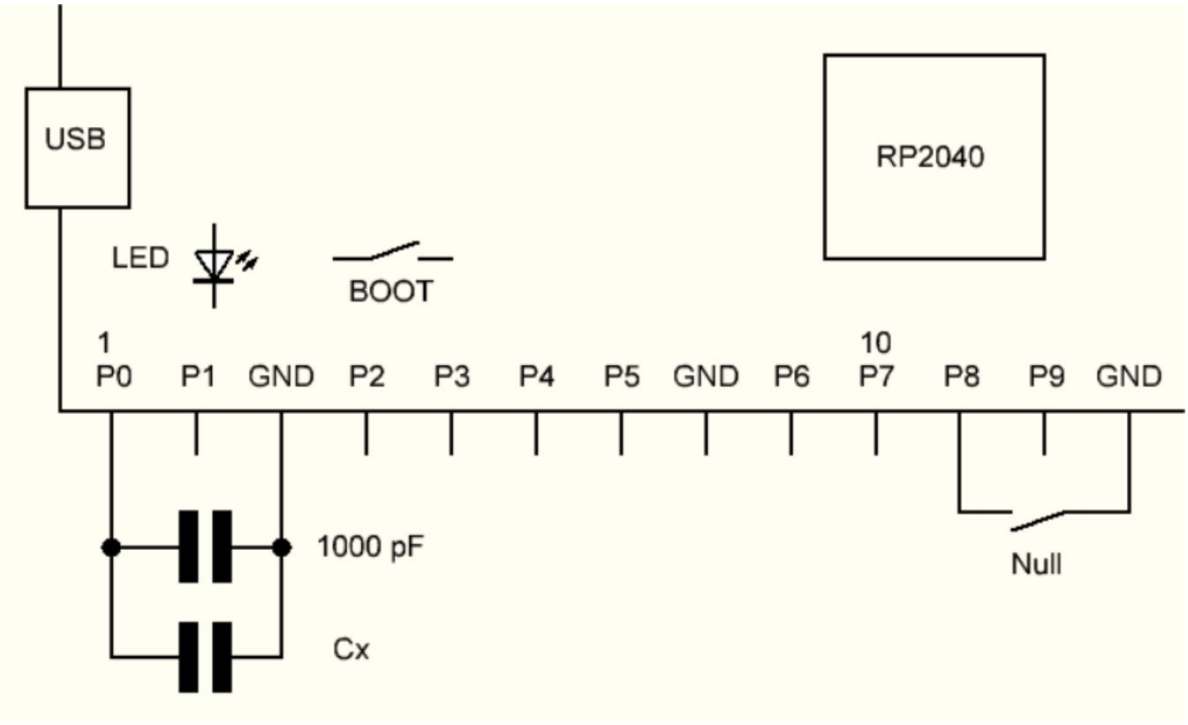

USB
LED
BOOT
1
P0
P1
GND
P2
P3
P4
Piezo



Данное измерение производилось полностью без конденсатора, т.е. только с учетом емкости осциллографа и порта. Вы можете увидеть треугольный сигнал прибл. 300 кГц. Небольшой джиттер вызван конечными во времени точками выборки где-то в диапазоне МГц. При большей емкости сигнал соответственно становится более точным.

21 Измерение емкости

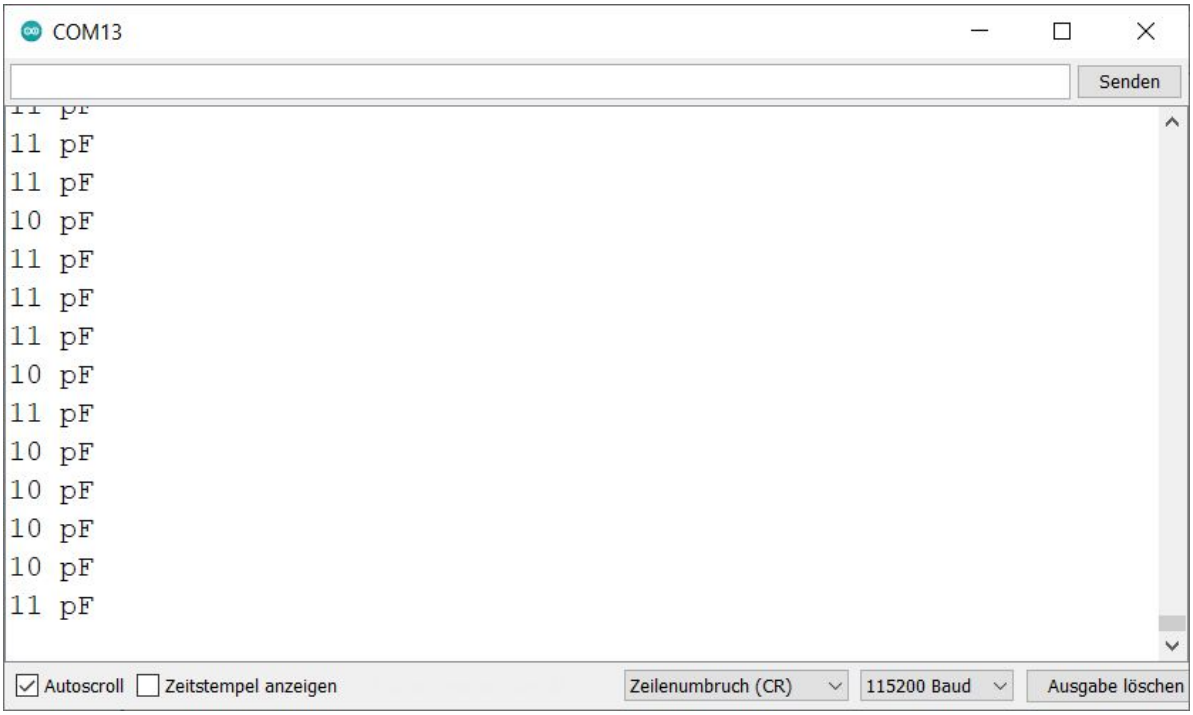
Это измерительное устройство основано на RC-генераторе с внутренними тянущими резисторами и внешним опорным конденсатором емкостью 1000 пФ. Измерение частоты происходит одновременно. Если дополнительно подключен внешний измерительный объект, емкость увеличивается, а частота снижается. При 1000 пФ измеряется частота около 60 кГц. Точная частота без объекта измерения вначале присваивается переменной f0. Емкость измеряемого объекта затем непрерывно рассчитывается по изменению частоты. При необходимости достаточно нажать кнопку P8, чтобы отрегулировать нулевую точку.



Измеритель можно использовать с хорошей точностью в диапазоне от 1 пФ до прибл. 100 нФ. Также можно проверять электролитические конденсаторы емкостью до 100 мкФ, хотя точность заметно снижается. Помимо конденсаторов, в обратном направлении можно измерять и диоды. На желтом светодиоде была измерена емкость перехода 7 пФ.

```
//Pico_RC4 C measurement
#include "pico/stdlib.h"

void setup() {
   Serial.begin(115200);
}

void loop() {}

void setup1() {
  int n=0;
  gpio_pull_up(8);
  uint32_t f0;
  while (true) {
    uint32_t f = 0;
    uint32_t t = time_us_32() + 1000000;
    while (t>time_us_32()){
      while (1==gpio_get(0));
      gpio_pull_up(0);
      f++;
      while (0==gpio_get(0));
      gpio_pull_down(0);
    }
    n++;
    if (n==2) f0=f;
    if (0==gpio_get(8)) n=0;
    if (n>2){
      n=3;
      uint32_t c= f0*1000/f-1000;
      if (c>1000000000) c=0;
      Serial.print(c);
      Serial.println(" pF ");
    }
  }
}
```

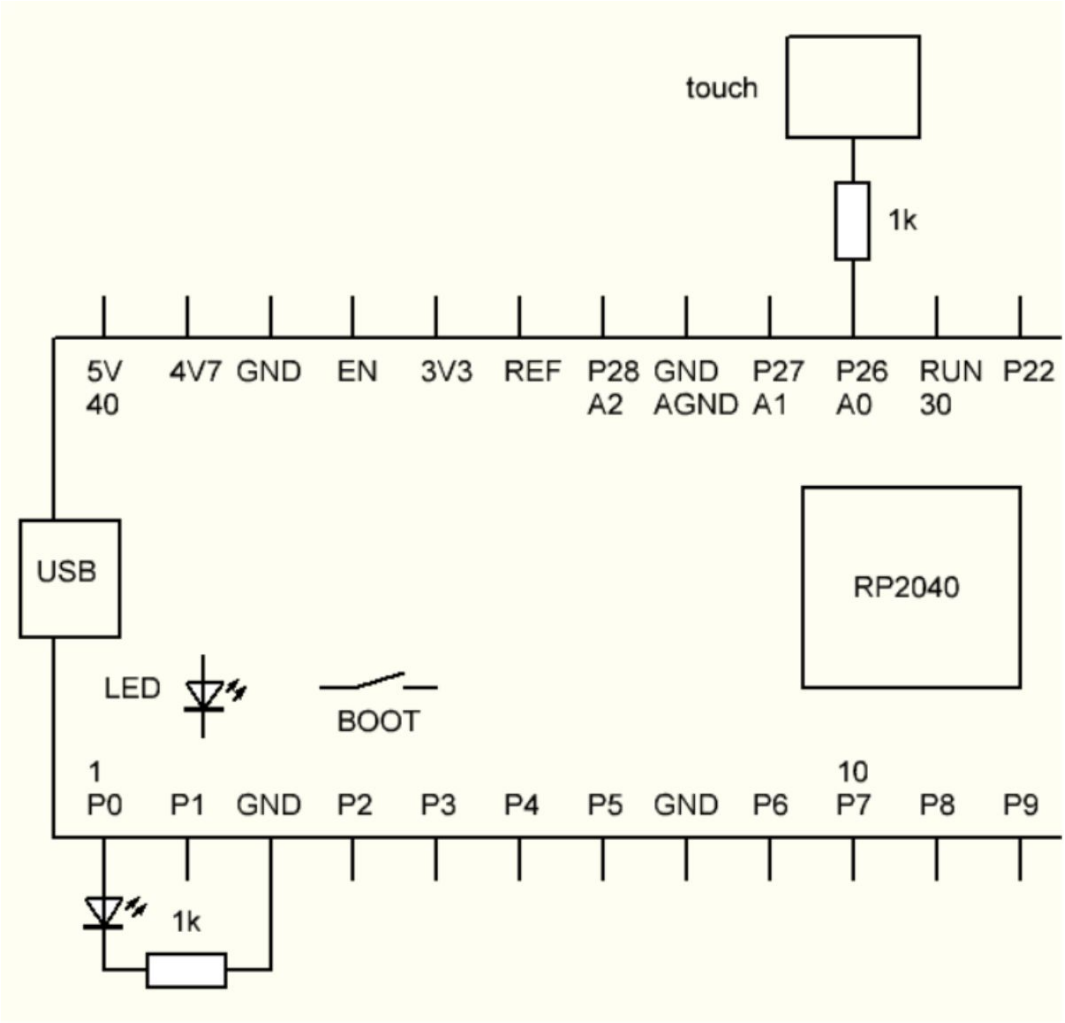
void loop1() {}



Другое возможное применение — емкостные сенсорные датчики. При прикосновении к изолированному проводу емкость увеличивается на 1 пФ примерно до 5 пФ. Прямое прикосновение к входу изолированного человека приводит к примерно 100 пФ.

22 Сенсорный датчик

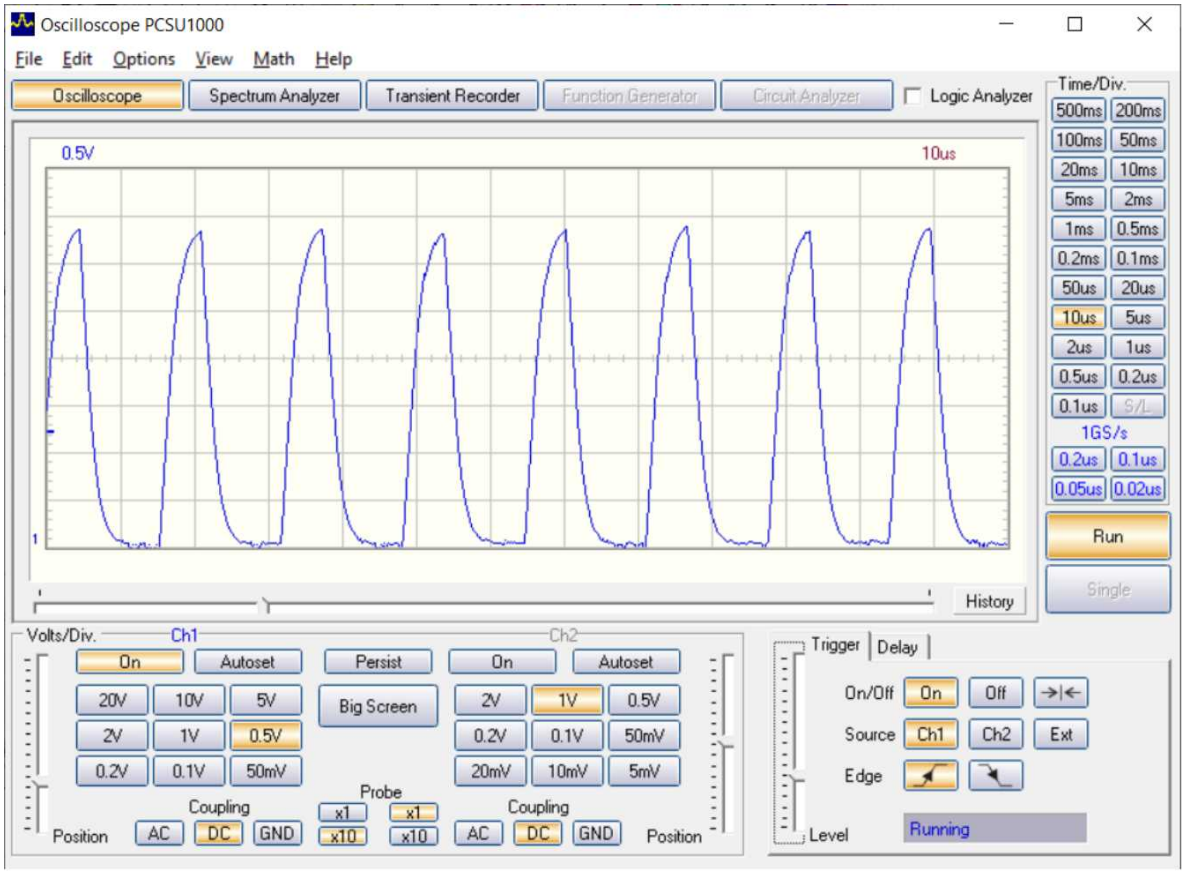
Датчик касания использует контакт P26 (A0) в качестве RC-генератора с резисторами и емкостью порта. АЦП постоянно измеряет напряжение и пытается поддерживать его в пределах от 1/3 до 2/3 рабочего напряжения. При прикосновении емкость увеличивается и изменяются характеристики генератора. Для защиты от ВЧ-излучения последовательно с контактом датчика включен резистор сопротивлением 1 кОм. Аналоговые измерения должны оцениваться таким образом, чтобы прикосновение было четко обнаружено. В этом случае включается выход P0.



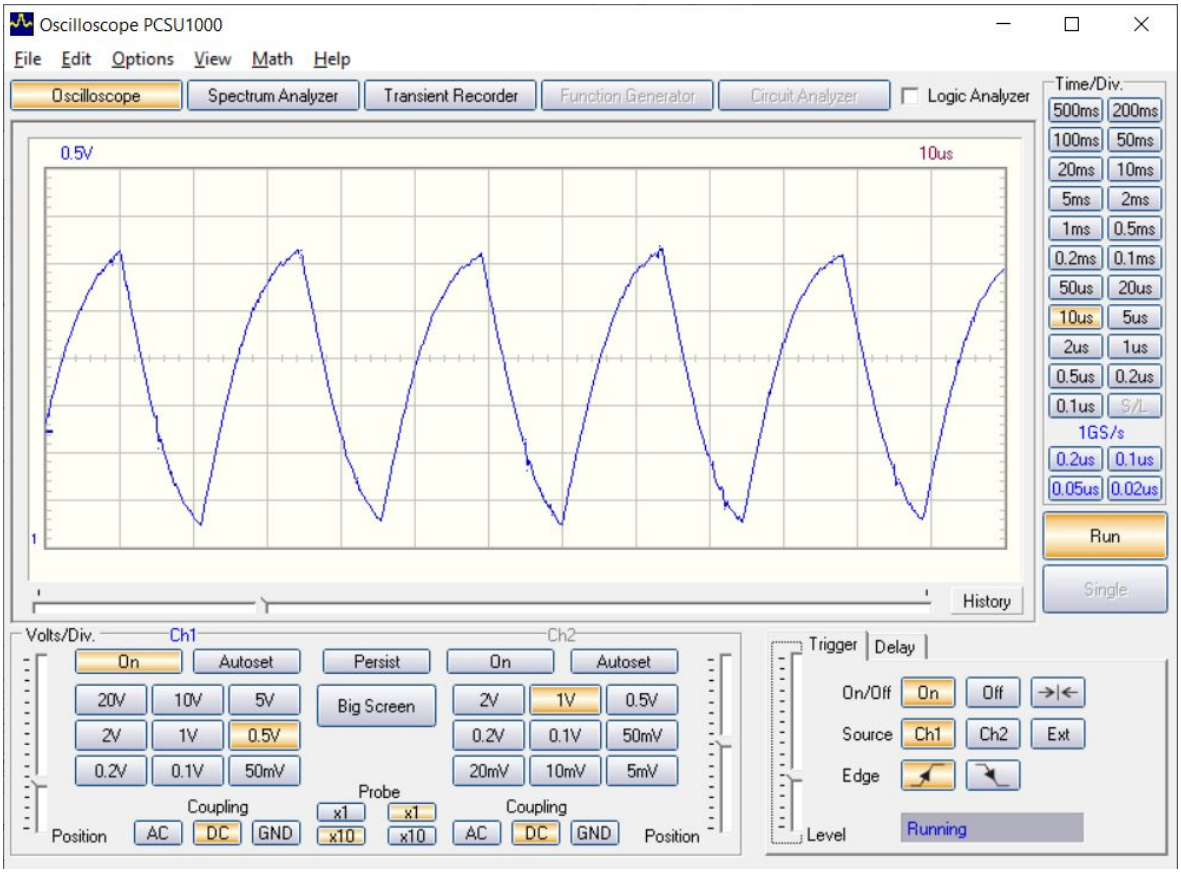
```
//Pico_RC3 Touch
#include "pico/stdlib.h"
#include "hardware/adc.h"

void setup() {
  Serial.begin(115200);
  adc_init();
  adc_gpio_init(26);
  gpio_pull_up(26);
  adc_select_input(0);
  gpio_init(0);
  gpio_set_dir(0, true);
  int u1, u2, u, u0;
  int n=0;
  while(true){
    u=0;
    n++;
    for (int n=0; n<100; n++){
      u1=adc_read();
      if (adc_read() > 2730){
        gpio_pull_down(26);
        u+=u1;
     }
     u2=adc_read();
     if (u2 < 1365){
        gpio_pull_up(26);
         u-=u2;
      }
    }
    u/=100;
    Serial.println(u);
    sleep_ms(50);
    if (n==2) u0=u;
    if (n>3) n=3;
```

```
    if (abs(u-u0) > 100) gpio_put(0,1); else gpio_put(0,0);
  }
}

void loop() {}
```

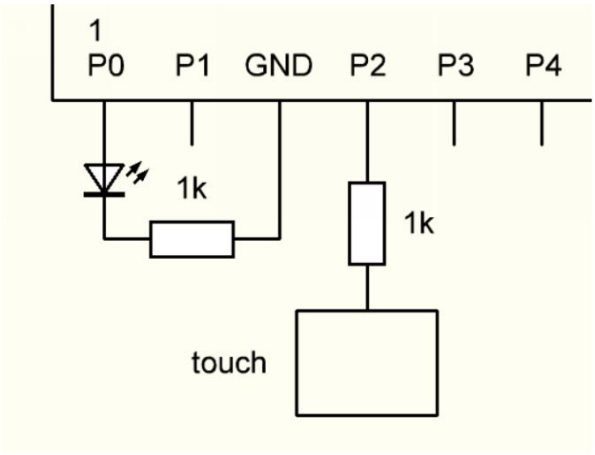
Анализ сигнала генератора показывает, что частота на холостом ходу составляет около 60 кГц. Программа работает слишком медленно, чтобы поддерживать напряжение в заданных пределах. Таким образом, амплитуда увеличивается примерно до 3,3 В. При оценке регистрируется разница напряжений между верхней и нижней точкой переключения как среднее значение для 100 запусков. Среднее значение, например, 500.



При прикосновении к контакту датчика емкость увеличивается, а частота падает, например, до 40 кГц. При изменении усредненной амплитуды напряжения более чем на 100 выполняется операция переключения.



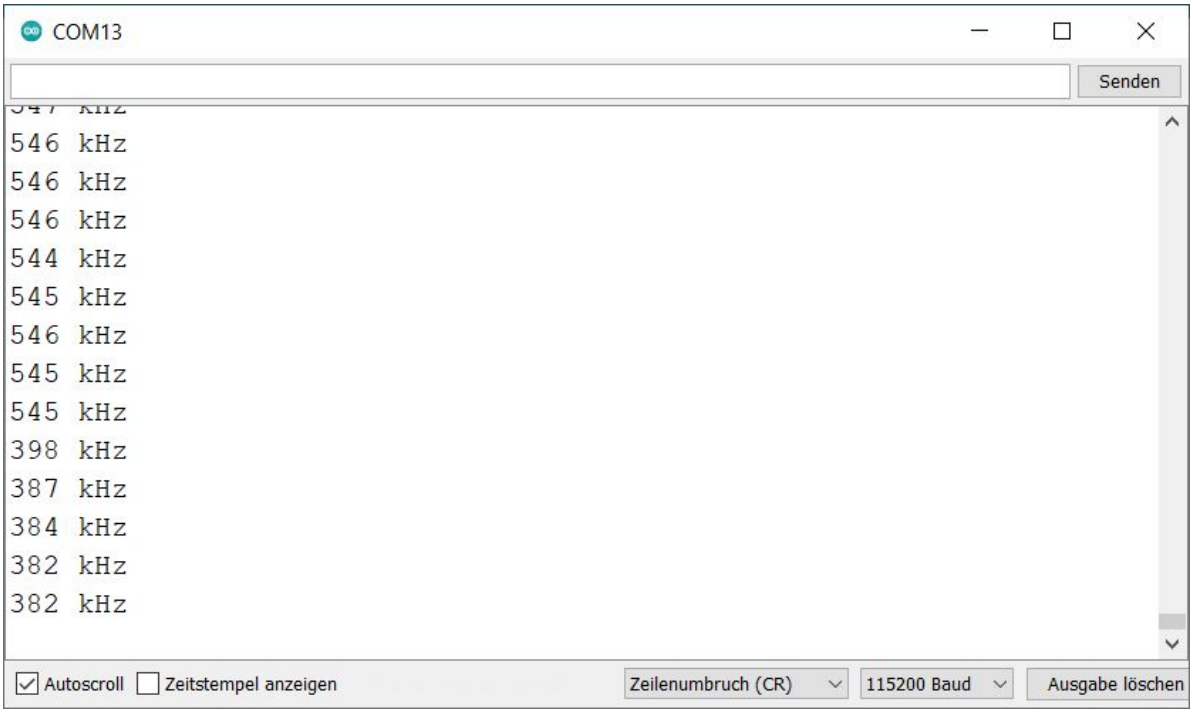
Второй вариант сенсорного датчика использует цифровой вход для настройки быстродействующего RC-генератора. При этом здесь происходит измерение частоты с разрешением 1 кГц. В зависимости от типа подключенного кабеля или сенсорного контакта измеряются частоты 600 кГц и более. В случае касания частота падает существенно ниже 500 кГц. Контакт датчика находится на P2, а выход на P0.




```
//Pico_RC5 Touch2
#include "pico/stdlib.h"

void setup() {}

void loop() {}

void setup1() {
  gpio_init(0);
  gpio_set_dir(0, true);
  gpio_init(2);
  while (true) {
    uint32_t f = 0;
    uint32_t t = time_us_32() + 1000;
    while (t>time_us_32()){
      while (gpio_get(2)==1);
        gpio_pull_up(2);
        f++;
      while (gpio_get(2)==0);
        gpio_pull_down(2);
    }
    if (f < 500) gpio_put(0,1);
    else gpio_put(0,0);
    Serial.print(f);
    Serial.println(" kHz ");
    sleep_ms(100);
  }
```

```
}

void loop1() {}
```

Для целей управления выводится измеренная частота. В примере в качестве контакта датчика была подключена алюминиевая фольга. В случае прямого контакта частота падала ниже 400 кГц. В зависимости от поверхности датчика и установленного порога частоты программа также подходит для емкостного датчика приближения без прямого контакта. Тогда достаточно приблизить руку примерно на 2 см. Также можно использовать изолирующую поверхность между датчиком и рукой.

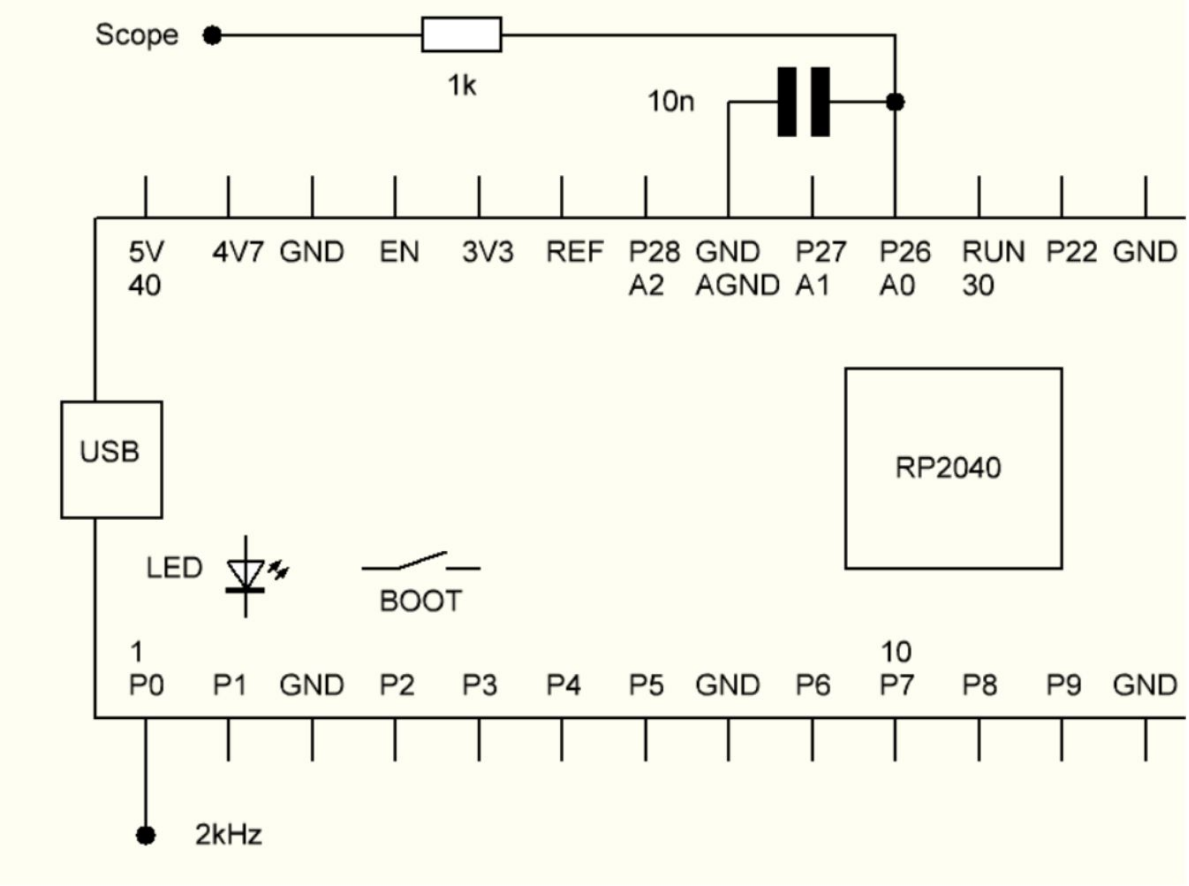


23 Быстродействующий осциллограф

АЦП Rpi Pico достигает частоты дискретизации до 500 кГц. Если вы хотите в полной мере воспользоваться этой скоростью, вам придется использовать прямой доступ к памяти (DMA), что относительно сложно. Очень простыми средствами можно достичь частоты 200 кГц. Контур с 500 точками измерения работает в Core1 и тактируется на частоте 200 кГц с помощью микросекундного таймера. Таким образом, промежуток времени между двумя точками измерения составляет 5 мкс. После записи измеренных значений они отправляются на ПК. В последовательном плоттере осциллограммы получаются с осью времени 500 мкс на деление шкалы. Самая высокая измеряемая частота составляет 100 кГц.

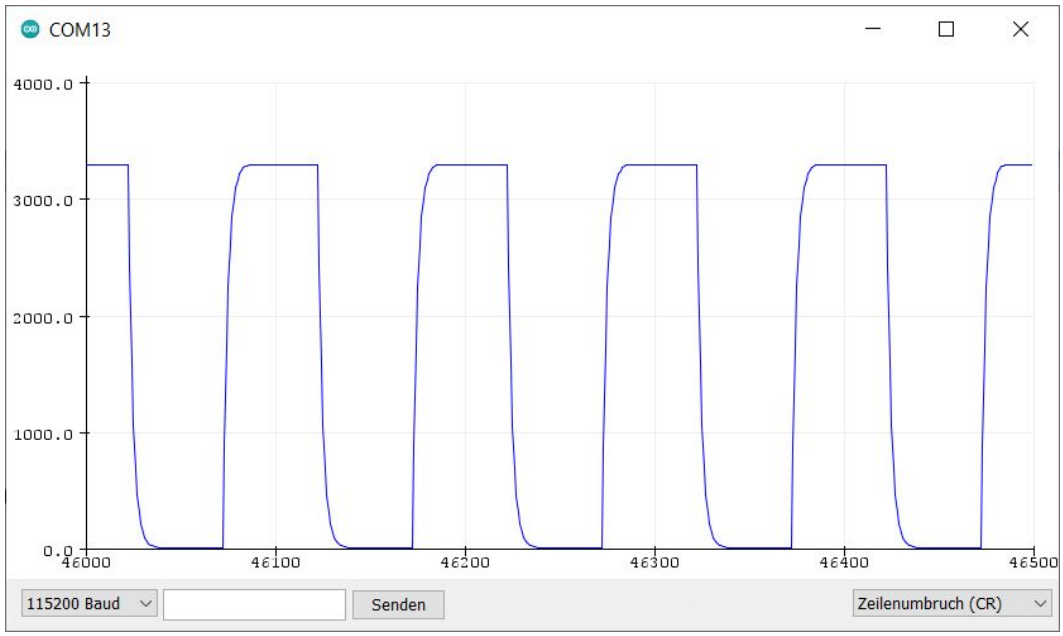
```
//Pico_Scope2
#include "pico/stdlib.h"
#include "hardware/adc.h"
#include "hardware/pwm.h"
#include "pico/multicore.h"

void setup() {
  Serial.begin(115200);
}

void loop() {
}

void setup1() {
  unsigned int scope[1000];
  gpio_set_function(0, GPIO_FUNC_PWM);
  pwm_set_wrap(0, 62499); //PWM 2 kHz, 50%
  pwm_set_gpio_level(0, 31250);
  pwm_set_enabled(0, true);
  adc_init();
  adc_gpio_init(26);
  adc_select_input(0);
  while (true) {
    uint32_t t = time_us_32();
```

```
    for (int n = 0; n < 500; n++){
      scope[n]=adc_read();
      t+=5; while (t>time_us_32()); //200 kHz
    }
    for (int n = 0; n < 500; n++){
      Serial.println ((int) scope[n]*3300/4095);
    }
    sleep_ms(1000);
  }
}

void loop1() {
}
```

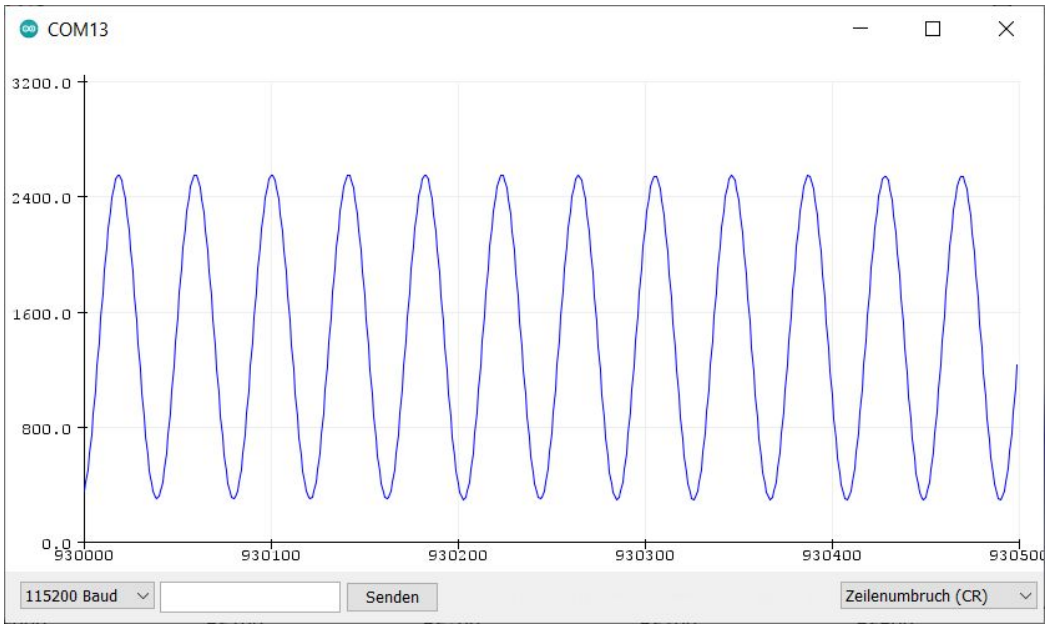


Осциллограф имеет тестовый выход с прямоугольным сигналом частотой 2 кГц. Аналоговый вход должен быть защищен резистором сопротивлением 1 кОм.

Вместе с параллельным конденсатором емкостью 10 нФ получается фильтр нижних частот с частотой среза 16 кГц, что особенно полезно для исследований в диапазоне ЗЧ. Для более высоких частот сигнала конденсатор следует исключить.

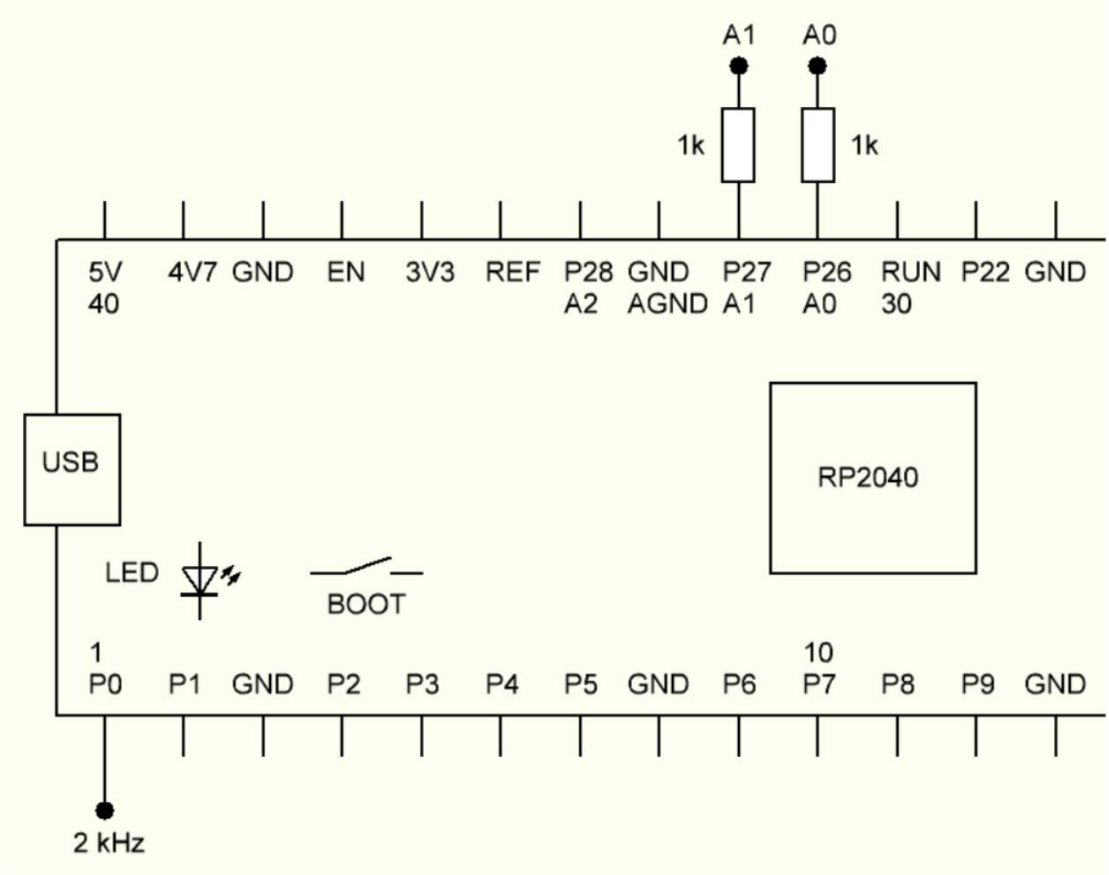


В примере показан тестовый сигнал частотой 2 кГц на выходе RC-фильтра нижних частот. Одно колебание точно умещается на участке шкалы со 100 точками измерения. Это подтверждает точную частоту дискретизации 200 кГц. Вы можете увидеть типичное закругление краев из-за фильтра нижних частот. Второй пример показывает синусоидальный сигнал частотой около 5 кГц.


COM13
3200.0
2400.0
1600.0
800.0
0.0
930000
930100
930200
930300
930400
930500
115200 Baud
Senden
Zeilenumbruch (CR)

24 Двухканальный осциллограф

Двухканальный осциллограф заполняет массив из 1000 значений поочередно результатами обоих каналов. Период дискретизации dt вначале устанавливается равным 10 мкс, так что частота дискретизации составляет 100 кГц, а время отклонения — 1 мс/дел. Скорость можно изменить с помощью последовательной команды. И снова предоставляется тестовый сигнал частотой 2 кГц.

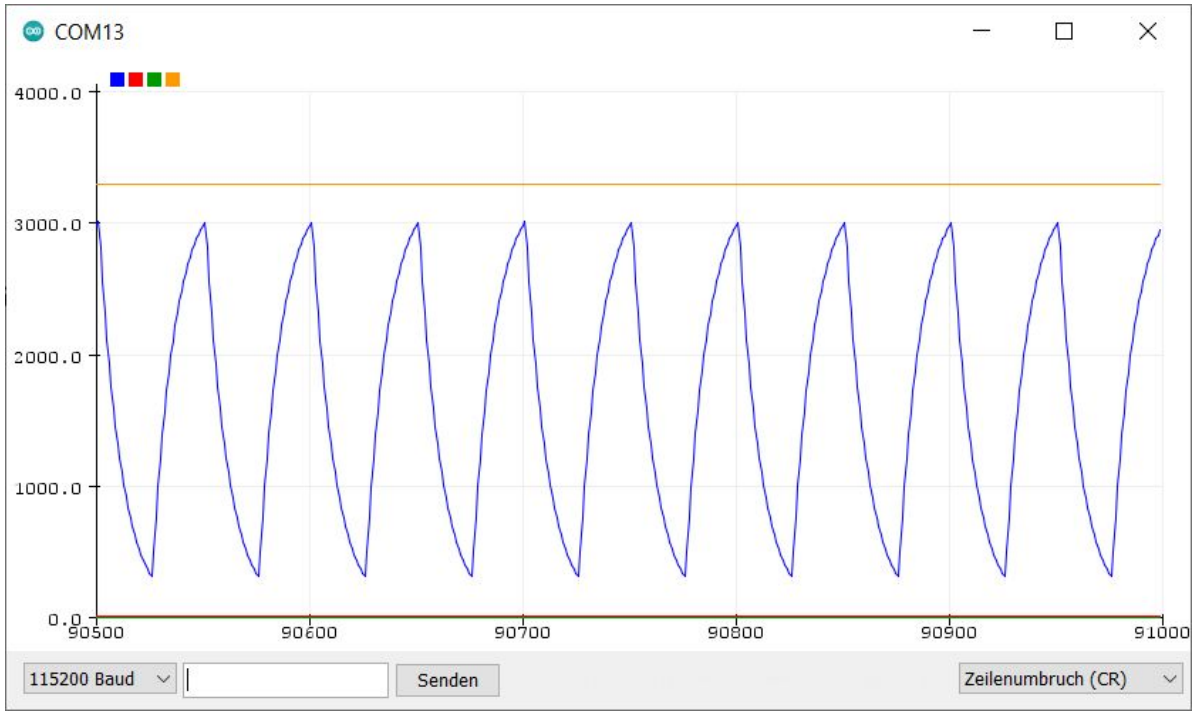



```
//Pico_Scope2 Dual Channel
#include "pico/stdlib.h"
#include "hardware/adc.h"
#include "hardware/pwm.h"
```

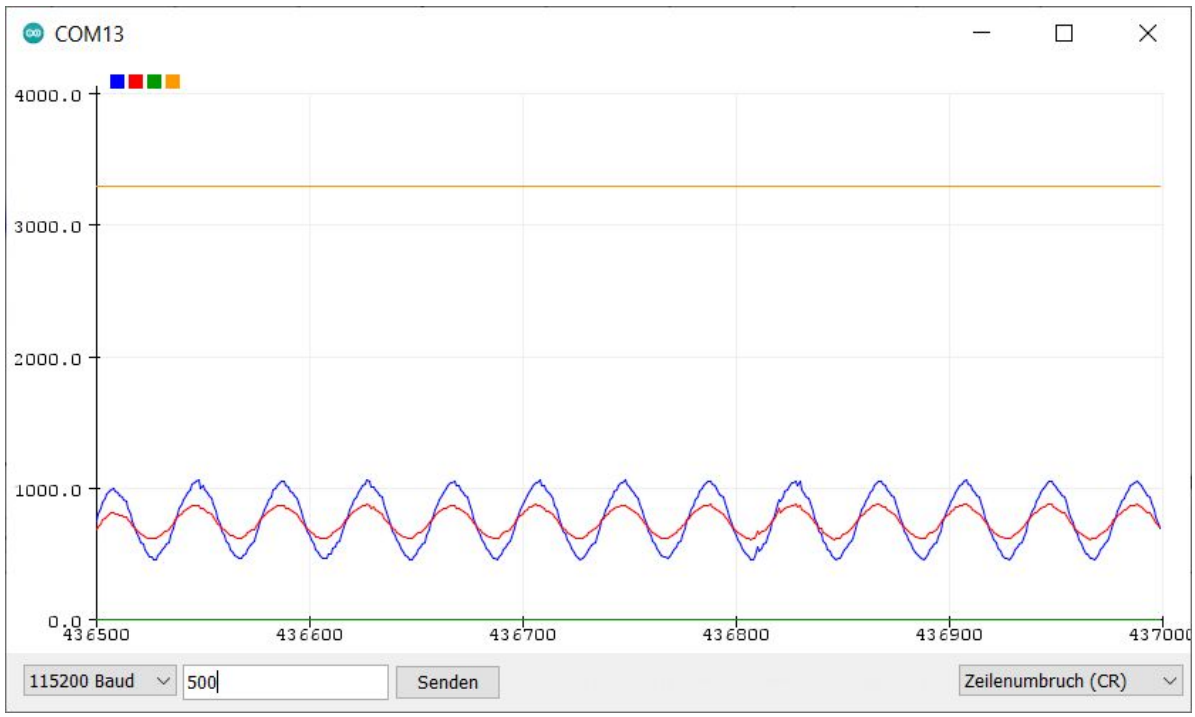
```
#include "pico/multicore.h"

void setup() {
  Serial.begin(115200);
}

void loop() {}

void setup1() {
  unsigned int scope[1000];
  unsigned int dt, dt2;
  dt=10;
  gpio_set_function(0, GPIO_FUNC_PWM);
  pwm_set_wrap(0, 62499); //PWM 2 kHz, 50%
  pwm_set_gpio_level(0, 31250);
  pwm_set_enabled(0, true);
  adc_init();
  adc_gpio_init(26);
  adc_gpio_init(27);
  adc_select_input(0);
  while (true) {
    if (Serial.available()){
      dt2 = Serial.parseInt();
      if( dt2>0) dt=dt2;
    }
    uint32_t t = time_us_32();
    for (int n = 0; n < 500; n++){
      adc_select_input(0);
      scope[2*n]=adc_read();
      adc_select_input(1);
      scope[2*n+1]=adc_read();
      t+=dt;
      while (t>time_us_32());
    }
    for (int n = 0; n < 500; n++){
      Serial.print ((int) scope[2*n]*3300/4095);
      Serial.print (" ");
```

```
      Serial.print ((int) scope[2*n+1]*3300/4095);
      Serial.print (" ");
      Serial.print (0);
      Serial.print (" ");
      Serial.println (3300);
    }
    sleep_ms(1000);
  }
}

void loop1() {
}
```

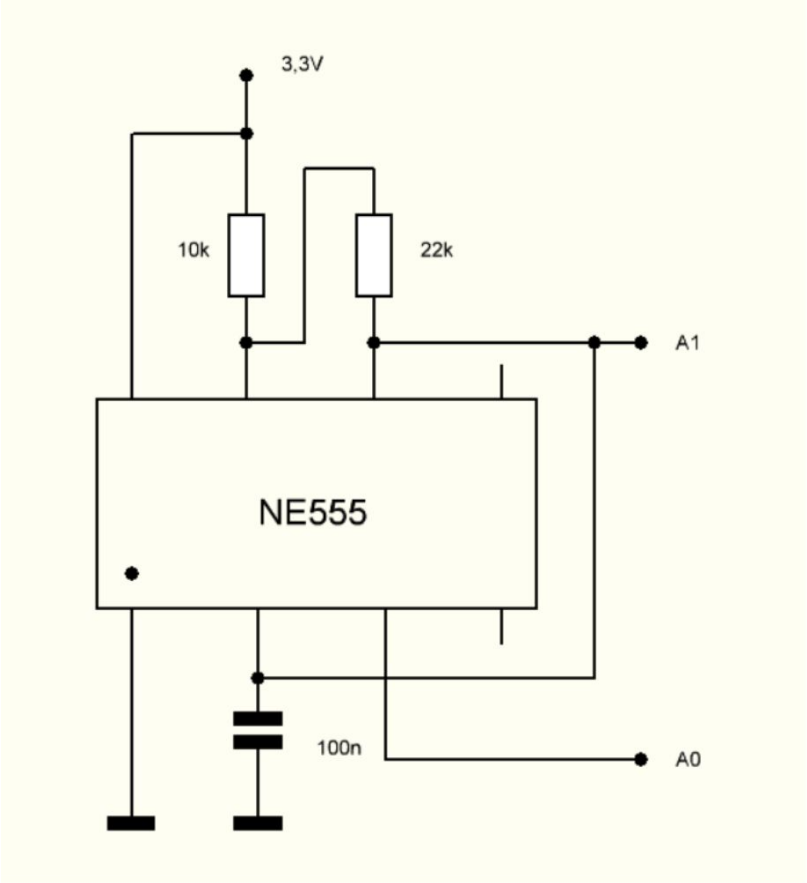
В первом примере измерения показан сигнал частотой 2 кГц после прохождения фильтра нижних частот с сопротивлением 10 кОм и 10 нФ на первом канале. Второй канал был подключен к GND. На осциллограммах также всегда присутствуют две вспомогательные линии при 0 и при 3300 мВ. Это обходит автоматическую регулировку диапазона последовательного плоттера, который в противном случае всегда настраивает шкалу Y для разных сигналов.

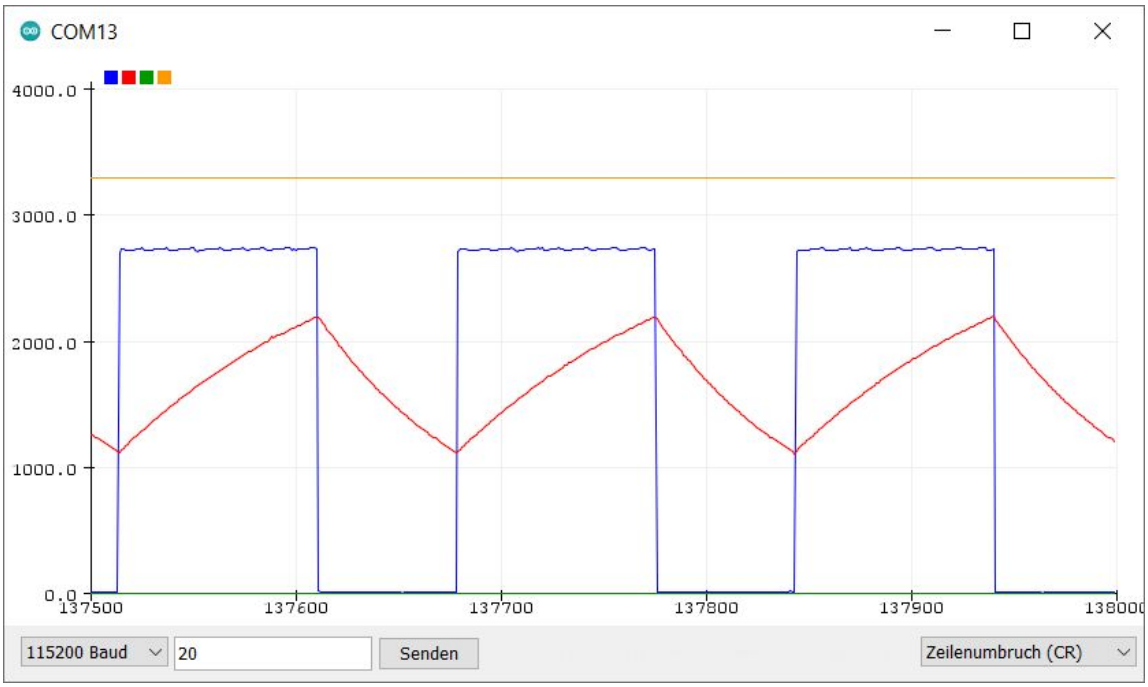


Установка 500 мкс приводит к временному коэффициенту 50 мс/дел. Измерение показывает два открытых входа с подключенными измерительными кабелями. В обе стороны рассеивается интерференционный сигнал частотой 50 Гц с разной амплитудой. При периоде 20 мс в каждой части шкалы наблюдается 2,5 колебания.



Двухканальный осциллограф может быть очень полезен при исследовании электронных схем. Здесь собрана типичная схема генератора с прецизионным таймером NE555. Осциллограф используется для измерения напряжения на зарядном конденсаторе и выходного сигнала NE555. В этом случае подходящим оказалось время отклонения 2 мс/дел, которое было установлено при времени выборки 20 мкс.

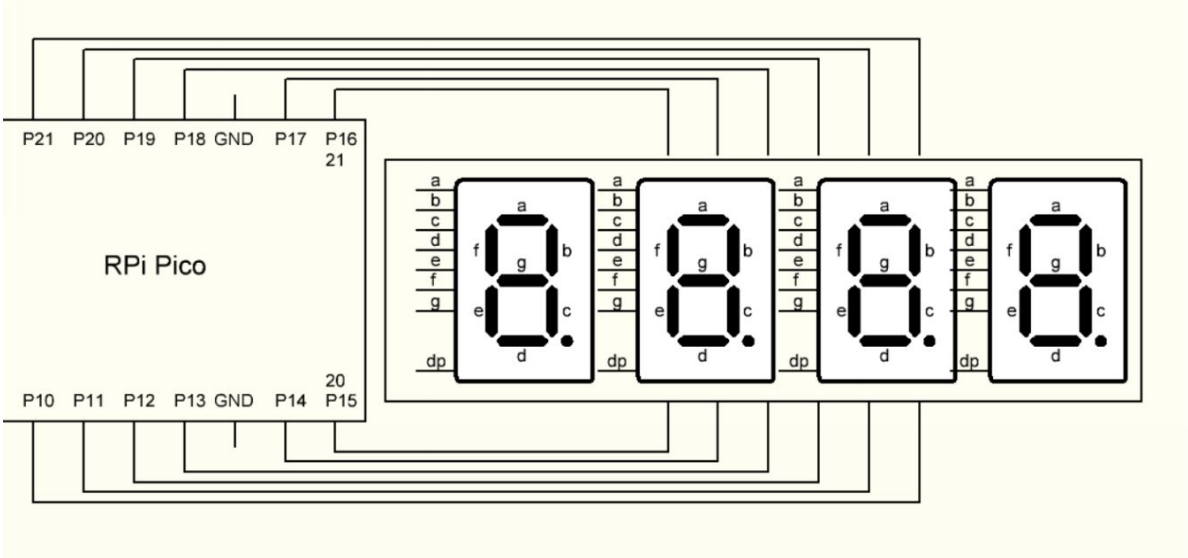

3,3V
10k
22k
A1
NE555
100n
A0


COM13
4000.0
3000.0
2000.0
1000.0
0.0
137500
137600
137700
137800
137900
138000
115200 Baud
20
Senden
Zeilenumbruch (CR)

25 Семисегментный дисплей

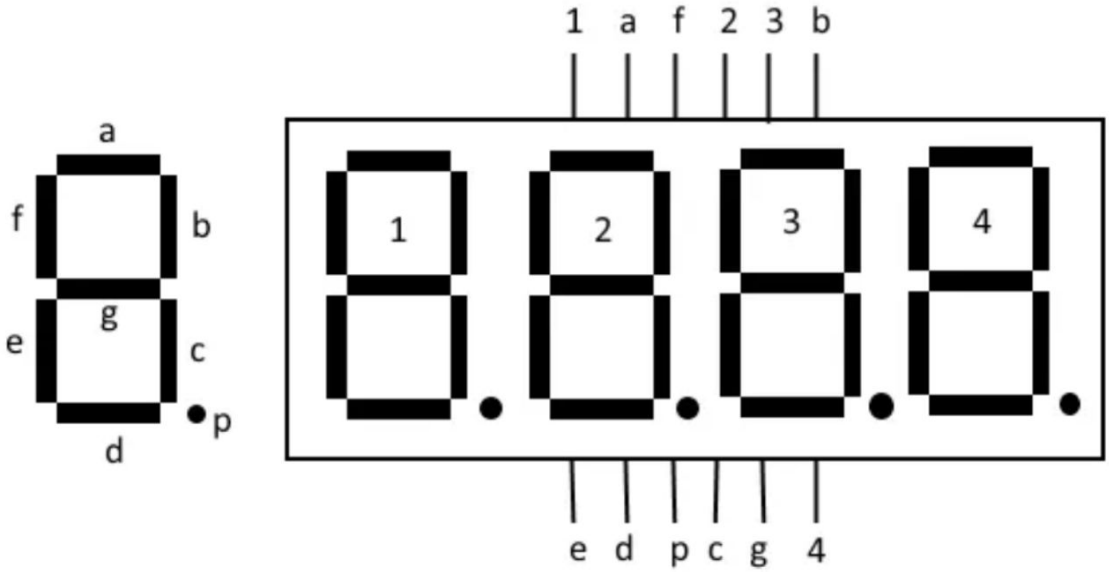
Среди функций SDK Rpi Pico есть функции маски, которые можно использовать для объединения нескольких GPIO в один более крупный порт. Таким образом, можно сформировать 8-битный порт, 16-битный порт или даже целый порт для всех входов/выходов. Здесь предстоит управлять четырехзначным семисегментным дисплеем, имеющим в общей сложности 12 портов. Четыре из них подключены к общим катодам отдельных цифр от 1 до 4. Остальные восемь контактов относятся к анодам сегментов от a до f и десятичной запятой.

Для управления использовались 12 GPIO на правом краю Pico, то есть порты от GP10 до GP21. Порты подключил так, чтобы самый короткий провод оказался посередине, который потом лежит под остальными проводами. Это меняет порядок портов с GP10 на GP15.



В исходном коде шаблоны от d[0] до d[9] были определены для цифр от 0 до 9. В этом случае единицы вставляются в позиции четырех общих катодов, так что сначала все четыре цифры отключаются. На втором этапе желаемое катодное соединение отключается.

В результате получается следующее назначение портов:



GP21GP20GP19GP18GNDGP17GP16

b 3 2 f a 1

4 g c p d e

GP10GP11GP12GP13GNDGP14GP15

Учитывая все 32 возможных порта, получается следующее назначение. Используются средние 12 портов.

0000 0000 004g cpde 1af2 3b00 0000 0000

0000 0000 0011 1111 1111 1100 0000 0000 = 0x003FFC00

В результате создается маска 0x003FFC00, которая указывает, какие порты должны управляться как общий порт. gpio_init_mask (маска) инициализирует эту группу. С помощью gpio_set_dir_masked (маска, маска) все переключаются как выходы. С помощью gpio_put_masked (маска, значение) вы можете переключать все 12 бит вместе в соответствии с битами в значении. Или вы можете полностью переключить их с помощью gpio_set_mask (маска).

Еще одна полезная функция переключает ток портов. Существует четыре уровня: 2 мА, 4 мА, 8 мА и 12 мА. В этом случае все сегментные линии устанавливаются на 2 мА с помощью gpio_set_drive_strength, а все катодные цифры — на 12 мА. Это означает, что вам не нужны последовательные резисторы для светодиодов дисплея.

```
//Pico_SevenSegment
#include "pico/stdlib.h"

void setup() {
  uint32_t mask = 0x003FFC00;
  uint32_t value = 0x003FFC00;
  uint32_t d[14];
  d[0]= 0b111011110101<<10;
  d[1]= 0b111001110111<<10;
  d[0]= 0b1111110101<<10;
  d[1]= 0b111000101<<10;
  d[2]= 0b111011110011<<10;
  d[3]= 0b111011010111<<10;
  d[4]= 0b111101000111<<10;
  d[5]= 0b011111010111<<10;
  d[6]= 0b0111110111<<10;
  d[7]= 0b111011000101<<10;
  d[8]= 0b1111110111<<10;
  d[9]= 0b1111010111<<10;

  gpio_init_mask (mask);
  gpio_set_dir_masked (mask, value);
  gpio_set_drive_strength(10,GPIO_DRIVE_STRENGTH_12MA);
  gpio_set_drive_strength(11,GPIO_DRIVE_STRENGTH_2MA);
  gpio_set_drive_strength(12,GPIO_DRIVE_STRENGTH_2MA);
  gpio_set_drive_strength(13,GPIO_DRIVE_STRENGTH_2MA);
  gpio_set_drive_strength(14,GPIO_DRIVE_STRENGTH_2MA);
  gpio_set_drive_strength(15,GPIO_DRIVE_STRENGTH_2MA);
  gpio_set_drive_strength(16,GPIO_DRIVE_STRENGTH_12MA);
  gpio_set_drive_strength(17,GPIO_DRIVE_STRENGTH_2MA);
  gpio_set_drive_strength(18,GPIO_DRIVE_STRENGTH_2MA);
  gpio_set_drive_strength(19,GPIO_DRIVE_STRENGTH_12MA);
  gpio_set_drive_strength(20,GPIO_DRIVE_STRENGTH_12MA);
```

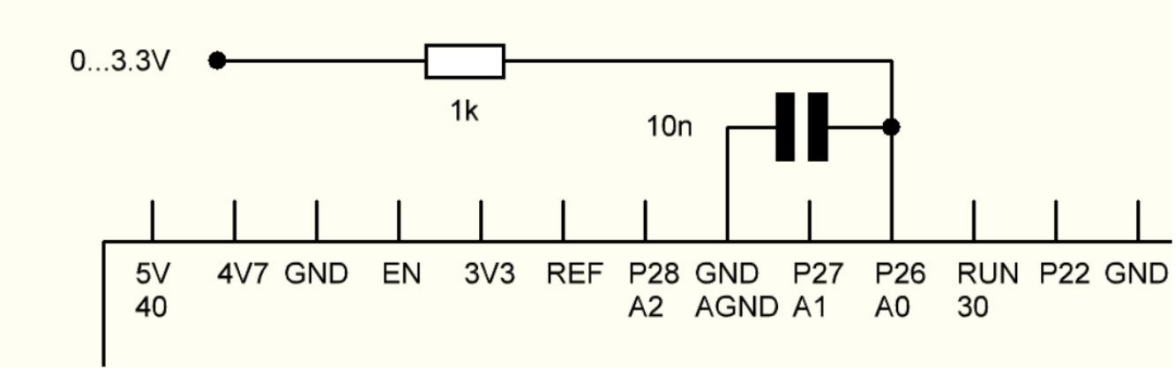
```
  gpio_set_drive_strength(21,GPIO_DRIVE_STRENGTH_2MA);

  int n=0;
  while(true){
    d[14]=d[n%10];
    int nn= n/10;
    d[13]=d[nn%10];
    nn= nn/10;
    d[12]=d[nn%10];
    nn= nn/10;
    d[11]=d[nn%10];
    nn= n/10;
    n++;
    for (int i=0; i<25; i++){
      gpio_put_masked (mask, d[11]);
      gpio_put(13,1); //DP
      gpio_put(16,0);
      sleep_ms(1);
      gpio_put_masked (mask, d[12]);
      gpio_put(19,0);
      sleep_ms(1);
      gpio_put_masked (mask, d[13]);
      gpio_put(20,0);
      sleep_ms(1);
      gpio_put_masked (mask, d[14]);
      gpio_put(10,0);
      sleep_ms(1);
    }
  }
}

void loop() {}
```

В цикле for все четыре цифры управляются одна за другой в течение одной миллисекунды каждая в мультиплексе. Десятичная точка должна быть установлена индивидуально с помощью gpio_put(13,1), что было сделано здесь для первой цифры. Всего программа подсчитывает переменную n за цикл 0,1 с.

Затем отдельные цифры разделяются, и их шаблоны сохраняются в регистрах от d[11] до d[14] для вывода в контуре мультиплексирования. Это обеспечивает стабильное изображение без мерцания.

26 Цифровой вольтметр

Четырехзначный семисегментный дисплей можно использовать в качестве индикатора напряжения, поскольку он четкий и его можно считывать с большого расстояния. Программа снова использует БИХ-фильтр для сглаживания измеренных значений. На дисплее используется десятичная точка в первой цифре, поэтому может отображаться напряжение от 0,000 В до 3,300 В.




```
//Pico_SevenSegmentAD

#include "pico/stdlib.h"
#include "hardware/adc.h"

void setup() {
  uint32_t ad=0;
  uint32_t iir=0;
  Serial.begin(115200);
  adc_init();
  adc_gpio_init(26);
  //gpio_pull_up(26);
  adc_select_input(0);
  uint32_t mask = 0x003FFC00;
  uint32_t value = 0x003FFC00;
  uint32_t d[14];
  d[0]= 0b111011110101<<10;
```

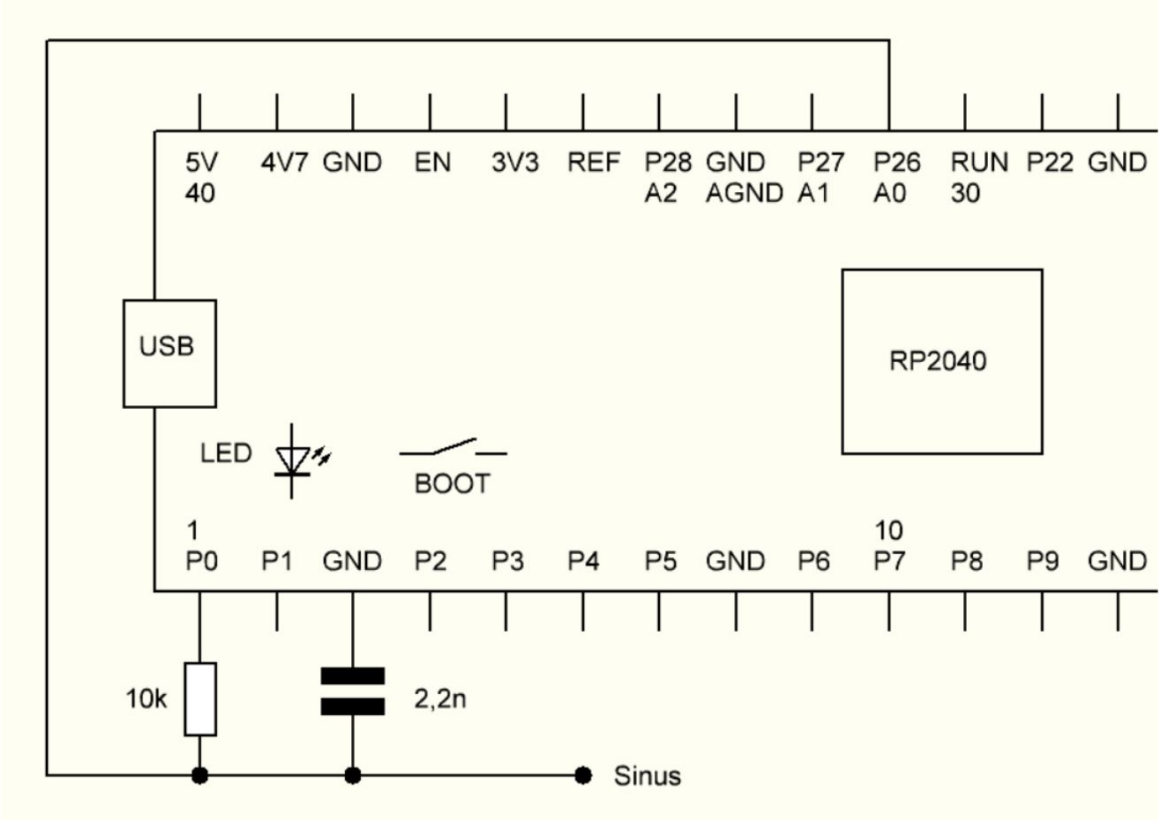
```
d[1]= 0b111001110111<<10;
d[0]= 0b1111110101<<10;
d[1]= 0b111000101<<10;
d[2]= 0b111011110011<<10;
d[3]= 0b111011010111<<10;
d[4]= 0b111101000111<<10;
d[5]= 0b011111010111<<10;
d[6]= 0b0111110111<<10;
d[7]= 0b111011000101<<10;
d[8]= 0b1111110111<<10;
d[9]= 0b1111010111<<10;

gpio_init_mask (mask);
gpio_set_dir_masked (mask, value);
gpio_set_drive_strength(10,GPIO_DRIVE_STRENGTH_12MA);
gpio_set_drive_strength(11,GPIO_DRIVE_STRENGTH_2MA);
gpio_set_drive_strength(12,GPIO_DRIVE_STRENGTH_2MA);
gpio_set_drive_strength(13,GPIO_DRIVE_STRENGTH_2MA);
gpio_set_drive_strength(14,GPIO_DRIVE_STRENGTH_2MA);
gpio_set_drive_strength(15,GPIO_DRIVE_STRENGTH_2MA);
gpio_set_drive_strength(16,GPIO_DRIVE_STRENGTH_12MA);
gpio_set_drive_strength(17,GPIO_DRIVE_STRENGTH_2MA);
gpio_set_drive_strength(18,GPIO_DRIVE_STRENGTH_2MA);
gpio_set_drive_strength(19,GPIO_DRIVE_STRENGTH_12MA);
gpio_set_drive_strength(20,GPIO_DRIVE_STRENGTH_12MA);
gpio_set_drive_strength(21,GPIO_DRIVE_STRENGTH_2MA);

while(true){
  ad = adc_read()*3300/4095;
  iir = (iir*3+ad)/4; //IIR filter
  Serial.println(iir);
  d[14]=d[iir%10];
  int nn= iir/10;
  d[13]=d[nn%10];
  nn= nn/10;
  d[12]=d[nn%10];
  nn= nn/10;
```

```
    d[11]=d[nn%10];
    for (int i=0; i<125; i++){
      gpio_put_masked (mask, d[11]);
      gpio_put(13,1); //DP
      gpio_put(16,0);
      sleep_ms(1);
      gpio_put_masked (mask, d[12]);
      gpio_put(19,0);
      sleep_ms(1);
      gpio_put_masked (mask, d[13]);
      gpio_put(20,0);
      sleep_ms(1);
      gpio_put_masked (mask, d[14]);
      gpio_put(10,0);
      sleep_ms(1);
    }
  }
}
void loop() {
}
```

27 Синусоидальных генератора DDS

Прямой цифровой синтез (DDS) — это метод генерации синусоидальных сигналов и других сигналов с высоким частотным разрешением. Программа Pico_ScopeDDS сочетает в себе DDS-генератор и быстродействующий осциллограф с частотой дискретизации 200 кГц.

Генератор DDS использует рассчитанную таблицу синусоид dds(1024) с амплитудным разрешением 600 точек. Значения берутся отсюда через равные промежутки времени и выводятся как ширина импульса ШИМ. Частота ШИМ составляет 200 кГц. С той же периодичностью вызывается функция прерывания, при которой выводятся новые табличные значения и принимаются измеренные значения АЦП. Таким образом, частота ШИМ, обновление выходных значений ШИМ и точки выборки осциллографа синхронны.

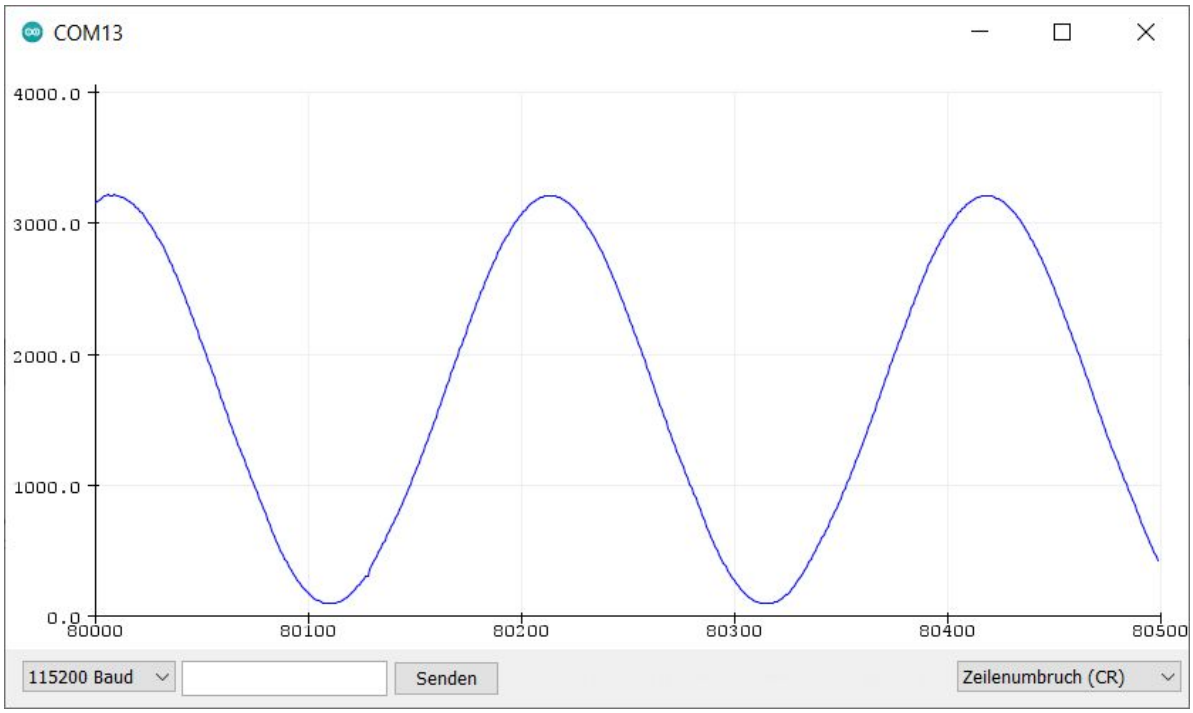


Сигнал ШИМ сглаживается ФНЧ сопротивлением 10 кОм и 2,2 нФ и одновременно подается на АЦП на вход A0.

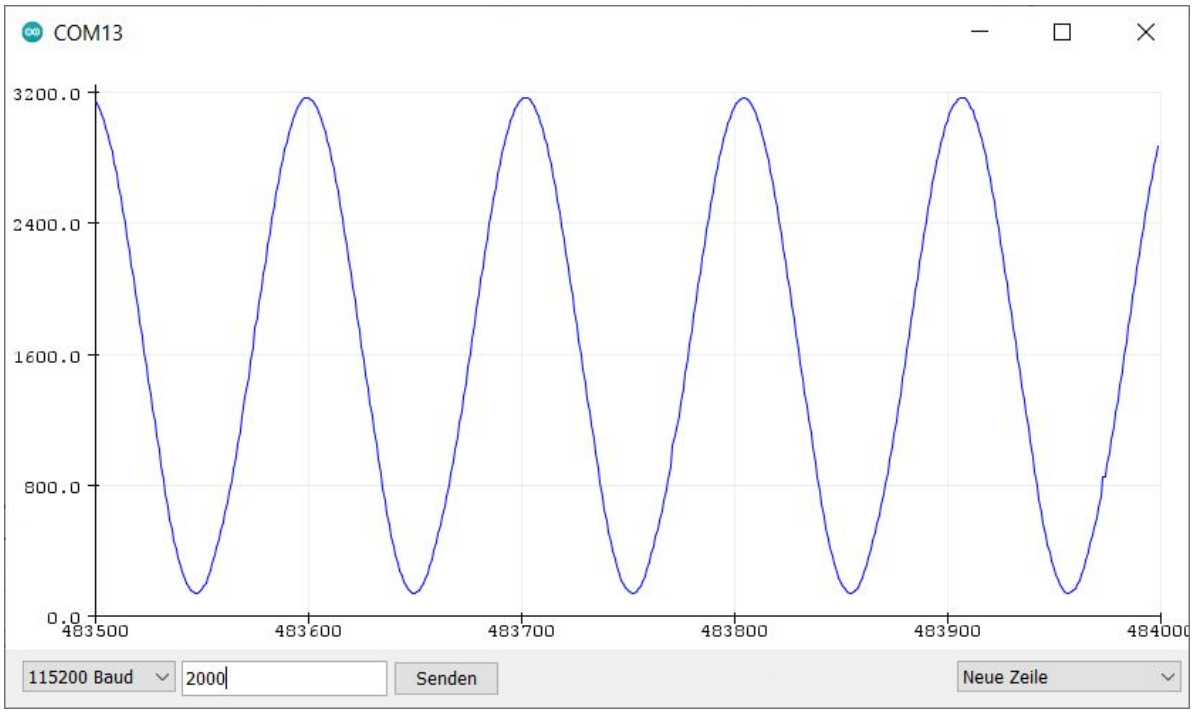
```
//Pico_ScopeDDS
#include "pico/stdlib.h"
#include "hardware/adc.h"
#include "hardware/pwm.h"
#include "pico/multicore.h"

unsigned int scope[1000];
unsigned int dds[1024];
uint16_t accumulator, i;
unsigned int f, dp;

void setup() {
  f=1000;
  dp=f*64/200;
  Serial.begin(115200);
  for (int j = 0; j < 1024; j++){
    dds[j]=312+300* sin(2*PI*j/1024.0);
  }
  while(true){
    if (Serial.available()){
      f = Serial.parseInt();
      if( f>0) dp=f*64/200;
    }
    i=0;
    sleep_ms(1000);
      for (int j = 0; j < 500; j++){
      Serial.println ((int) scope[j]*3300/4095);
      }
  }
}

void loop() {}

void setup1() {
  adc_init();
```

```
  adc_gpio_init(26);
  adc_select_input(0);

  gpio_set_function(0, GPIO_FUNC_PWM);
  pwm_set_wrap(0, 624); //PWM 200 kHz
  pwm_set_gpio_level(0, 312);
  pwm_set_enabled(0, true);
  pwm_set_irq_enabled(0, true);
  irq_set_exclusive_handler(PWM_IRQ_WRAP, pwm_int);
  irq_set_enabled(PWM_IRQ_WRAP, true);
  while (true);
}

void pwm_int() {
  pwm_clear_irq(0);
  pwm_set_gpio_level(0, dds[akku>>6]);
  accu+=dp;
  if (i<500){
    scope[i]=adc_read();
    i++;
  }
}

void loop1() {}
```

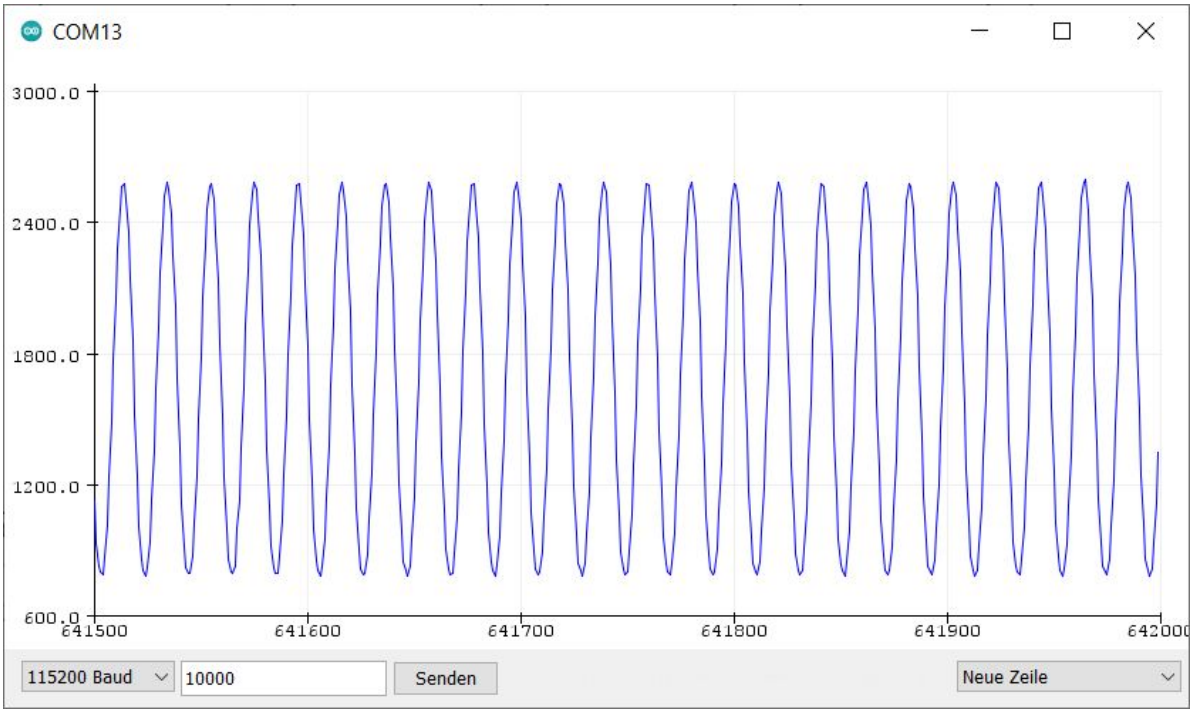
После запуска программы генерируется частота 1000 Гц. При преобразовании dp=f*64/200 на основе этого вычисляется разность фаз dp, которая добавляется к 16-битному фазовому аккумулятору во времени с выходным сигналом. Старшие 10 бит аккумулятора служат указателем адреса текущей позиции в таблице синусоид, dds [аккумулятор>>6]. Это дает разрешение по частоте около 4 Гц и строгую пропорциональность между dp и выходной частотой. На осциллограмме показан измеренный синусоидальный сигнал частотой 1 кГц на последовательном плоттере.



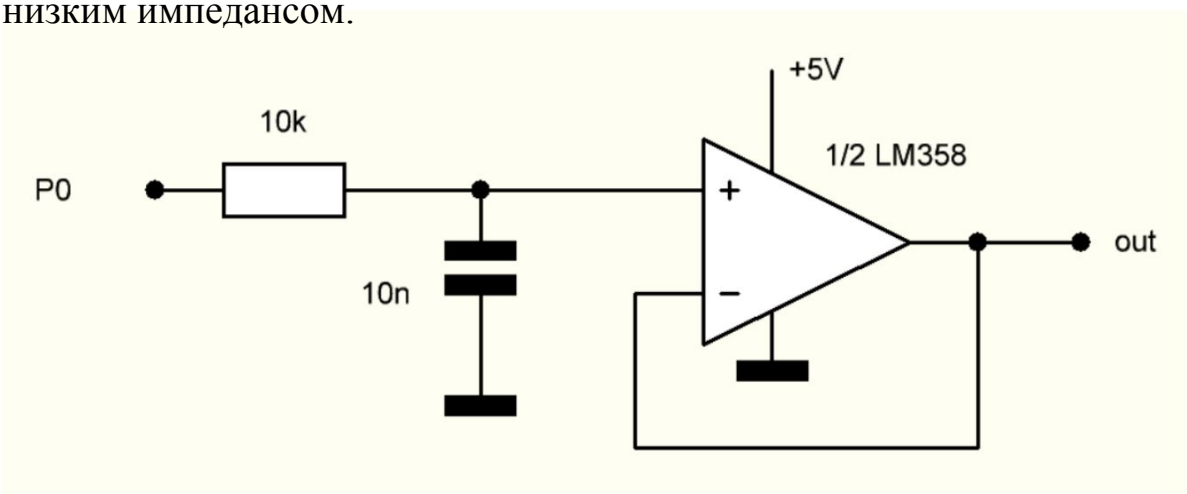
Программа позволяет устанавливать любые частоты посредством последовательных команд. В окне отправки введите, например. 2000 и подтвердите ввод клавишей возврата. Это устанавливает частоту генератора DDS на 2000 Гц.



Даже 10 кГц и более по-прежнему работают без заметных искажений. Однако амплитуда значительно уменьшается, поскольку частота сигнала уже превышает частоту среза фильтра нижних частот 7,2 кГц.



Выходной сигнал имеет относительно высокое сопротивление (10 кОм), поэтому амплитуда сигнала может резко падать при подключении внешних нагрузок. Буферный усилитель с операционным усилителем LM358 обеспечивает выходной сигнал с низким импедансом.

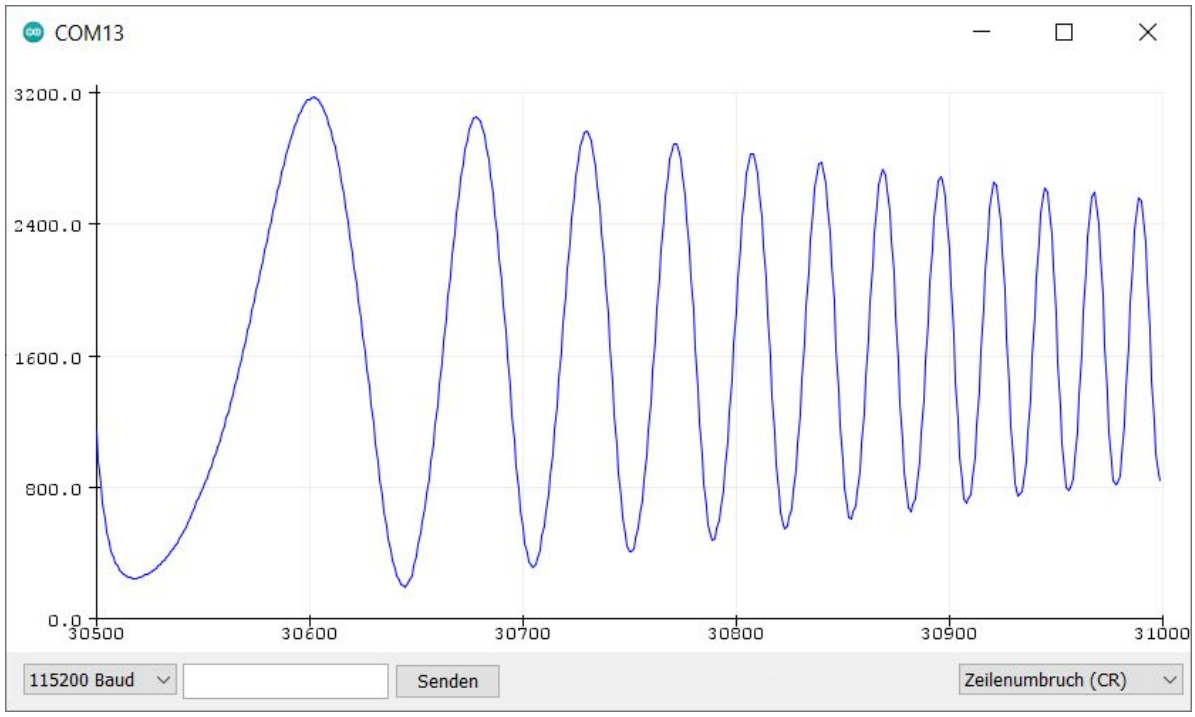


28 Свип-генератор

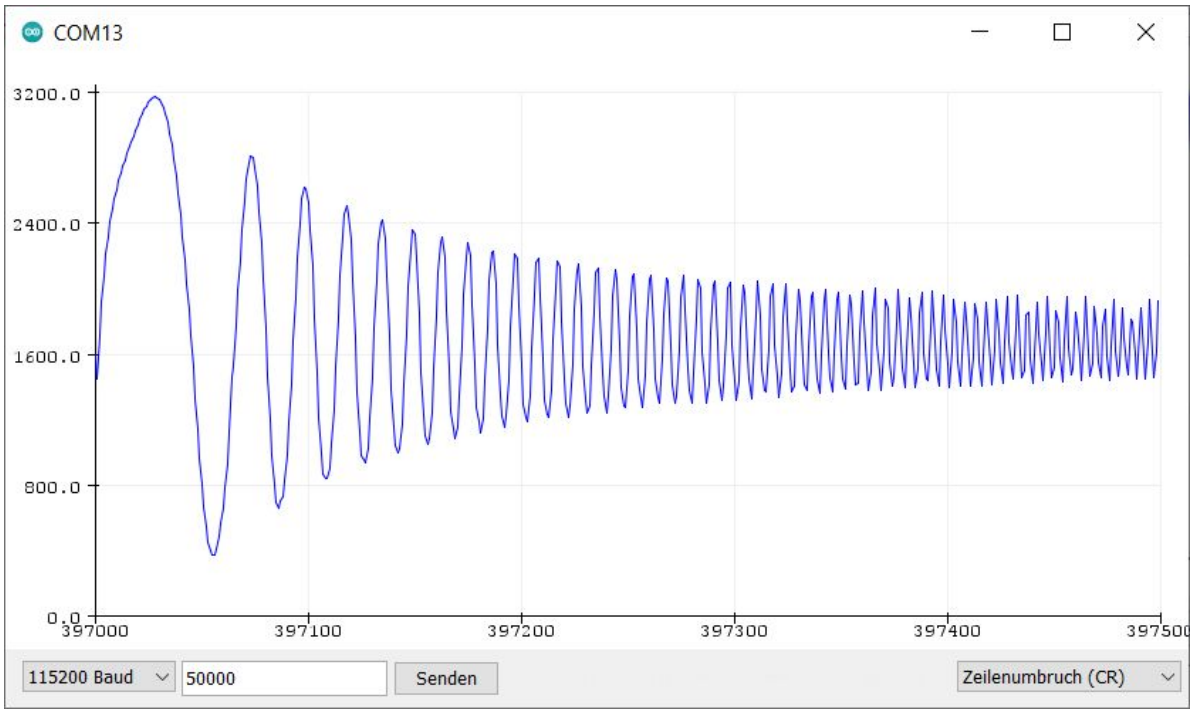
Свип-генератор, который линейно увеличивает частоту до определенного конечного значения, подходит для записи частотных характеристик. Эту задачу можно решить небольшими изменениями в генераторе синусоидального сигнала DDS. Введенная частота f теперь является конечной частотой и вначале установлена на 10 кГц. Решающее изменение заключается в функции прерывания. При каждом запуске измерения текущая частота пересчитывается.

```
//Pico_SweepDDS
#include "pico/stdlib.h"
#include "hardware/adc.h"
#include "hardware/pwm.h"
#include "pico/multicore.h"

unsigned int scope[1000];
unsigned int dds[1024];
uint16_t accumulator, i;
unsigned int f, dp;

void setup() {
  f=10000;
  dp=f*64/200;

...
...

void pwm_int() {
  pwm_clear_irq(0);
  pwm_set_gpio_level(0, dds[akku>>6]);
  if (i>499) akku+=dp;
  if (i<500){
    akku+=dp*i/500;
    scope[i]=adc_read();
    i++;
  }
  }
```

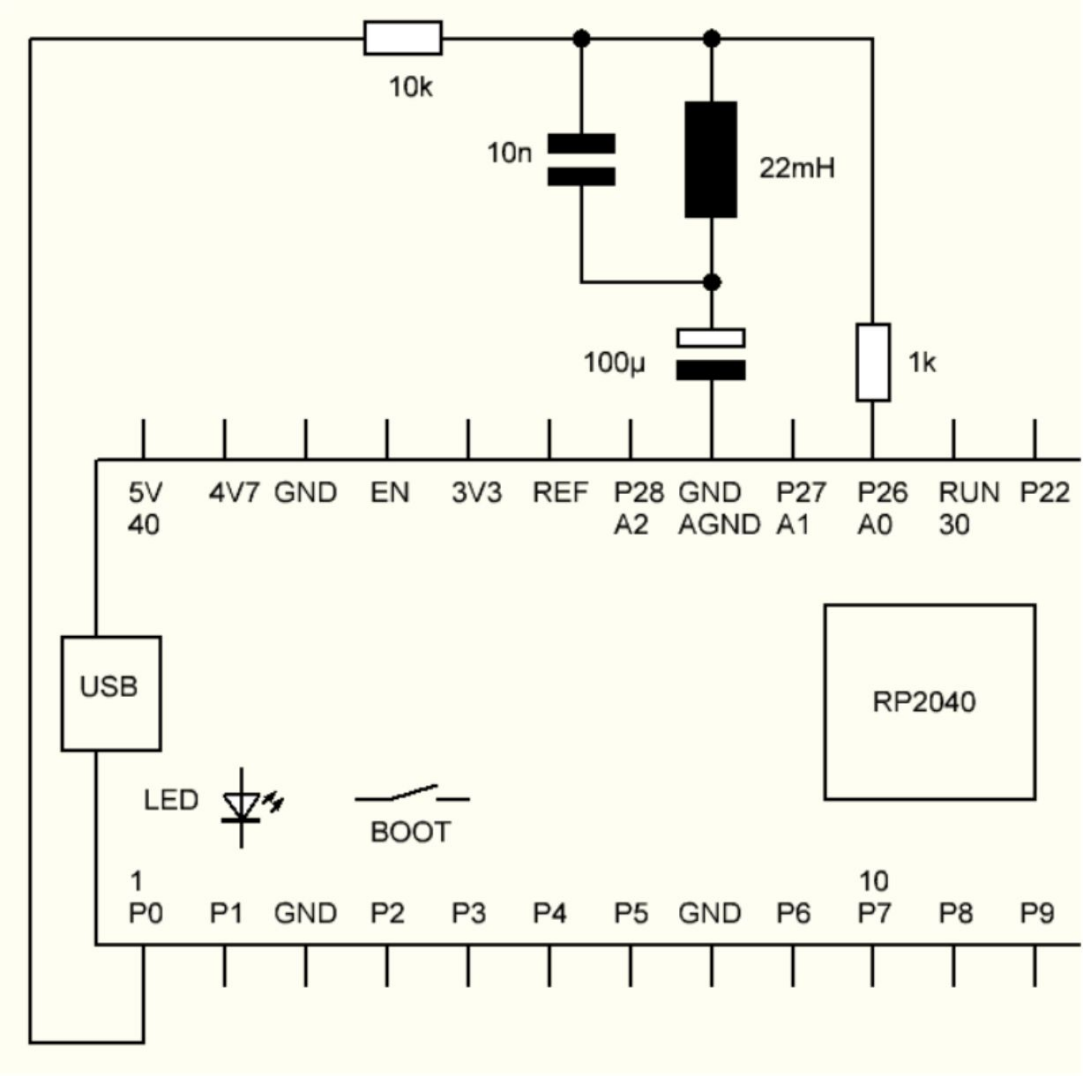
На осциллограмме показан отклик ФНЧ сопротивлением 10 кОм и 2,2 нФ в диапазоне от 0 до 10 кГц. Части шкалы обозначают частоты 2 кГц, 4 кГц, 6 кГц, 8 кГц и 10 кГц. На частоте среза около 7 кГц амплитуда сигнала должна была упасть на 3 дБ, т.е. до коэффициента 0,71. Результат подтверждает теорию в пределах точности считывания.



При измерении до 50 кГц влияние фильтра нижних частот становится еще более очевидным. В то же время в диапазоне выше 30 кГц видны определенные недостатки представления, поскольку частота сигнала слишком близка к частоте дискретизации. Если колебание состоит из слишком малого числа точек выборки, возникают заметные нарушения.

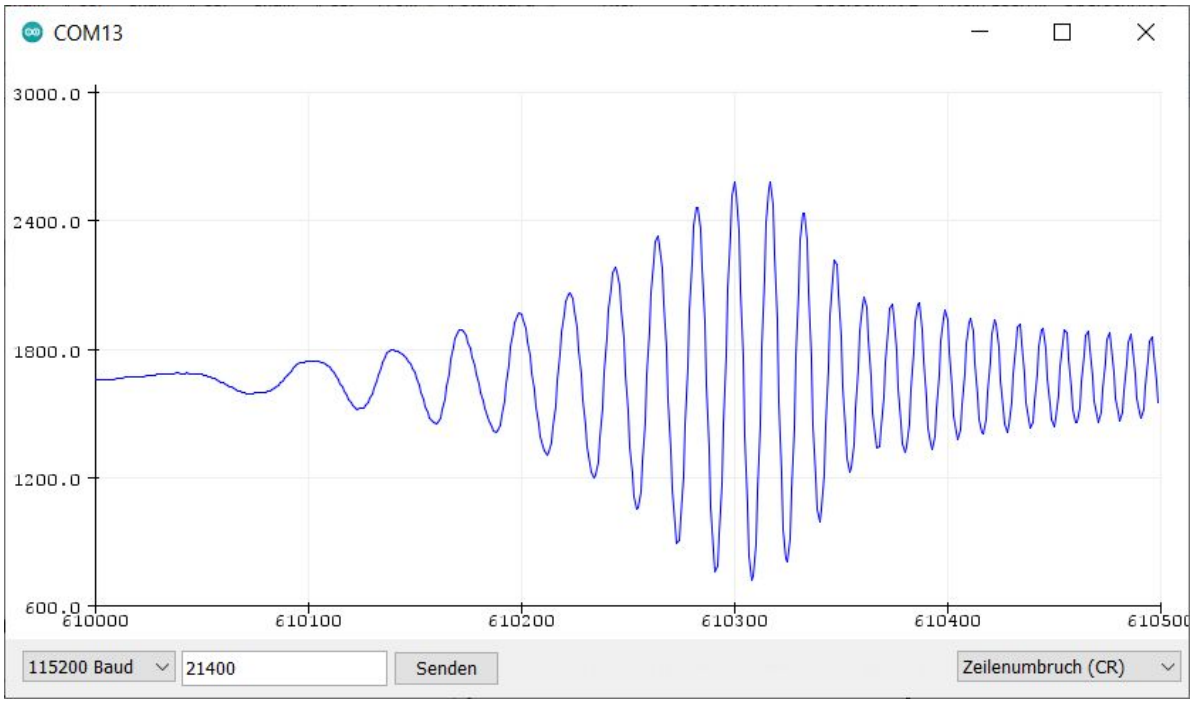


Свип-генератор также подходит для исследования полосовых фильтров. Здесь рассматривается LC-фильтр. Катушка имеет емкость 22 мГн и конденсатор 10 нФ. Это приводит к резонансной частоте 10,7 кГц.

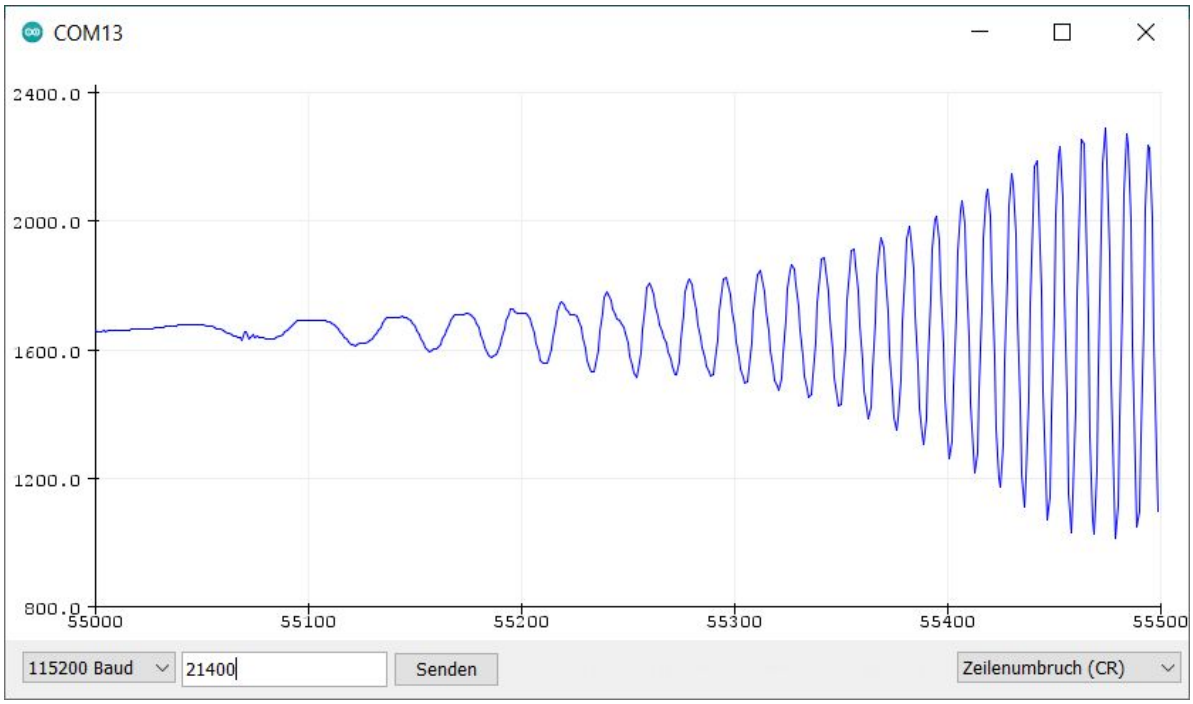


Генератор был настроен на конечную частоту 21,4 кГц. Это должно поместить резонанс в середину временной оси. Однако на самом деле она значительно выше, что, видимо, говорит о высоких допусках компонентов.

Испытание с помощью синусоидального генератора DDS показало, что амплитуда в резонансном контуре становится максимальной на частоте 12 кГц. С генератором развертки резонанс кажется несколько выше. Однако следует иметь в виду, что резонансному контуру всегда требуется определенное время для полного стабилизации. В результате максимальная амплитуда оказывается несколько задержанной. Для измерения было бы лучше увеличивать частоту медленнее. Однако против этого говорит ограниченная ширина выходного окна, поскольку тогда не каждое колебание можно было бы представить достаточно точно.



Поведение резонансного контура существенно меняется, когда небольшой магнит приближается к ферритовому сердечнику. Индуктивность уменьшается, поэтому резонансная частота увеличивается. Кроме того, становятся видны искажения в диапазоне малых частот, которые указывают на магнитное насыщение сердечника.

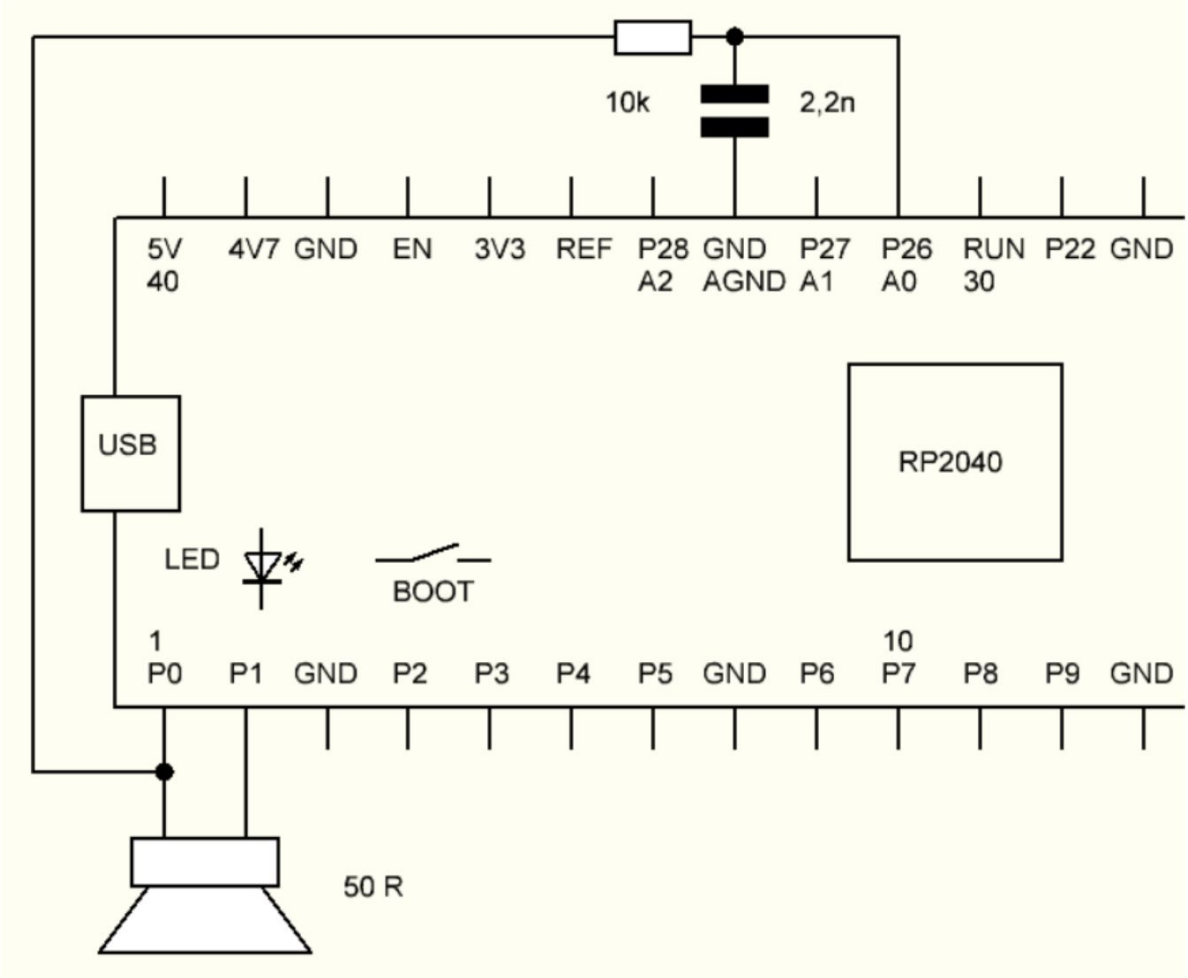

COM13
2400.0
2000.0
1600.0
1200.0
800.0
55000
55100
55200
55300
55400
55500
115200 Baud
21400
Senden
Zeilenumbruch (CR)

29 Двухтональный генератор

Большинство звуков содержат не только одну частоту, но и по крайней мере кратную основной частоте с уменьшением амплитуды по мере увеличения порядкового номера. Каждая форма волны может состоять из синусоидальных колебаний соответствующей частоты, фазы и амплитуды. В некоторых случаях двух колебаний достаточно для получения определенных звуков. Еще одним применением двухтонального генератора является измерительная техника. Например, можно управлять коротковолновым передатчиком с двумя тонами, а затем измерять очень слабые побочные излучения, которые могут возникнуть в результате ошибок линейности.

На этот раз частота дискретизации была снижена до 50 кГц и, таким образом, было достигнуто лучшее разрешение по частоте — один герц. Здесь смешиваются два тона с частотой 440 Гц и 523 Гц и равными амплитудами, в результате чего получается типичный двухтональный звук. Для этого можно использовать два выхода ШИМ, сигналы которых затем объединяются с резисторами. Однако можно обойтись и одним выходом, если складывать сигналы в числовом виде, а затем выводить их. Любые новые частоты можно ввести через последовательный вход.

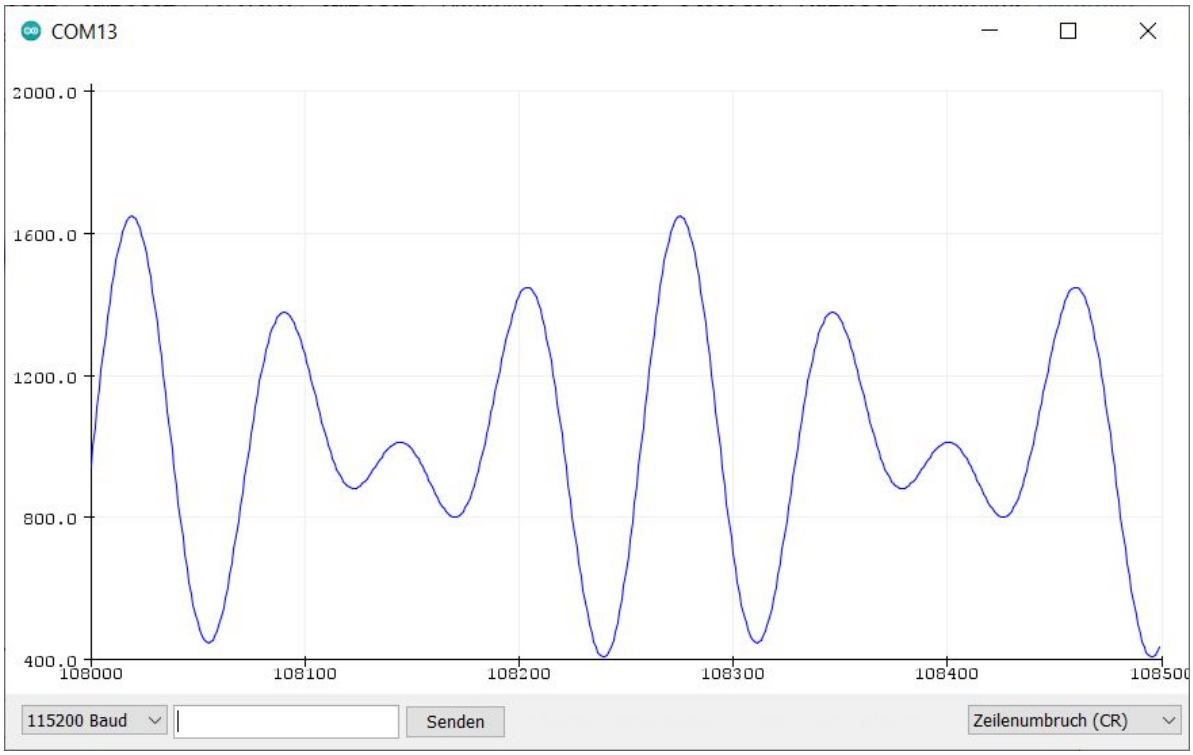
Кроме того, выход расширен до усилителя класса D, что позволяет подключить громкоговоритель с высоким сопротивлением непосредственно между D0 и D1. Сигнал ШИМ на D1 находится в противофазе с сигналом на D0, поэтому напряжение сигнала на громкоговорителе удваивается. Сигнал, отображаемый на осциллографе, представляет собой либо сигнал, рассчитанный для выхода, либо сигнал, измеренный через A0.



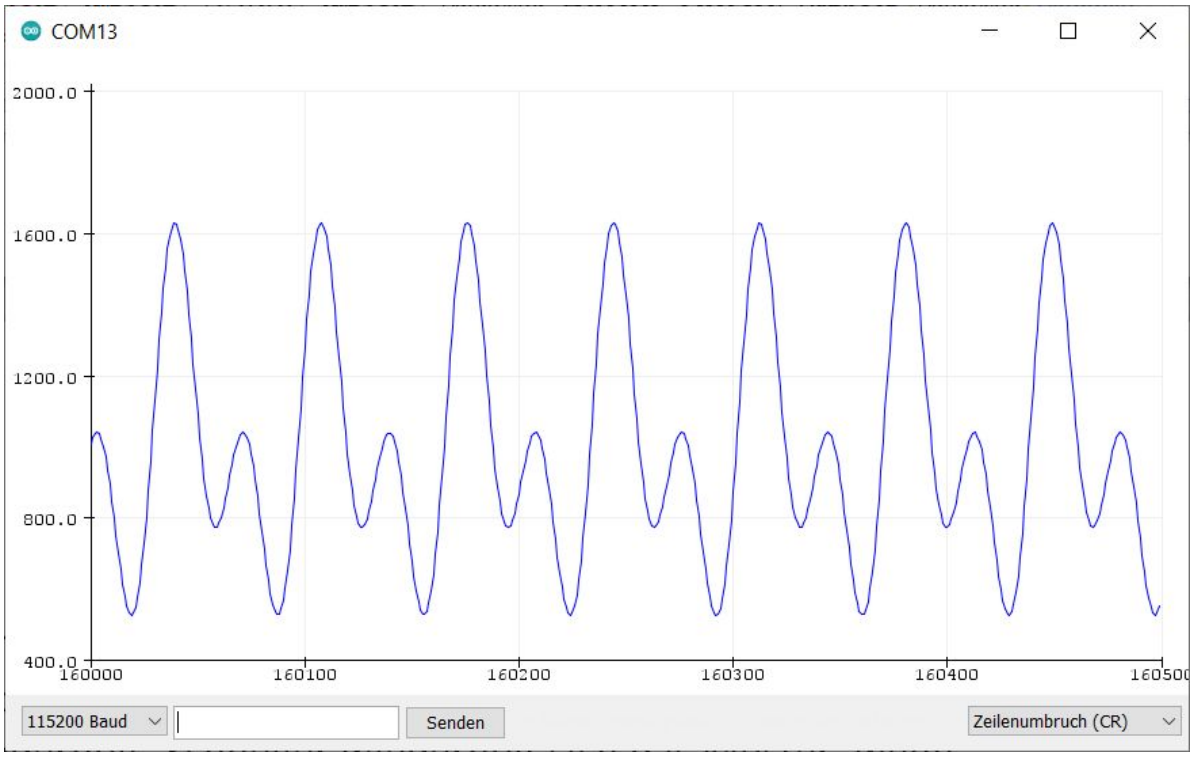

```
//Pico_ScopeDDS2 Two-tone
#include "pico/stdlib.h"
#include "hardware/adc.h"
#include "hardware/pwm.h"
#include "pico/multicore.h"

unsigned int scope[1000];
unsigned int dds[1024];
uint16_t akku1, akku2, i;
unsigned int f, f1, f2, dp1, dp2, t;

void setup() {
  f1=440;
  dp1=f1*64/50;
```

```
  f2=523;
  dp2=f2*64/50;
  Serial.begin(115200);
  for (int j = 0; j < 1024; j++){
    dds[j]=625+400* sin(2*PI*j/1024.0);
  }
  while(true){
    if (Serial.available()){
      f = Serial.parseInt();
      if( f>0){
        t++;
        if (t>2)t=1;
        if (t==1) dp1=f*64/100;
        if (t==2) dp2=f*64/100;
      }
    }
    i=0;
    sleep_ms(1000);
      for (int j = 0; j < 500; j++){
      Serial.println ((int) scope[j]*3300/4095);
      }
  }
}

void loop() {}

void setup1() {
  adc_init();
  adc_gpio_init(26);
  adc_select_input(0);

  gpio_set_function(0, GPIO_FUNC_PWM);
  gpio_set_function(1, GPIO_FUNC_PWM);
  pwm_set_wrap(0, 2499); //PWM 50 kHz
  pwm_set_gpio_level(0, 1250);
  pwm_set_gpio_level(1, 1250);
  pwm_set_enabled(0, true);
```

```
  pwm_set_irq_enabled(0, true);
  irq_set_exclusive_handler(PWM_IRQ_WRAP, pwm_int);
  irq_set_enabled(PWM_IRQ_WRAP, true);
  while (true);
}

void pwm_int() {
  pwm_clear_irq(0);
  unsigned int u=dds[akku1>>6]+ dds[akku2>>6];
  pwm_set_gpio_level(0,u);
  pwm_set_gpio_level(1,2499-u);
  akku1+=dp1;
  akku2+=dp2;
  if (i<500){
    scope[i]=u;
    //scope[i]=adc_read();
    i++;
  }
}

void loop1() {}
```



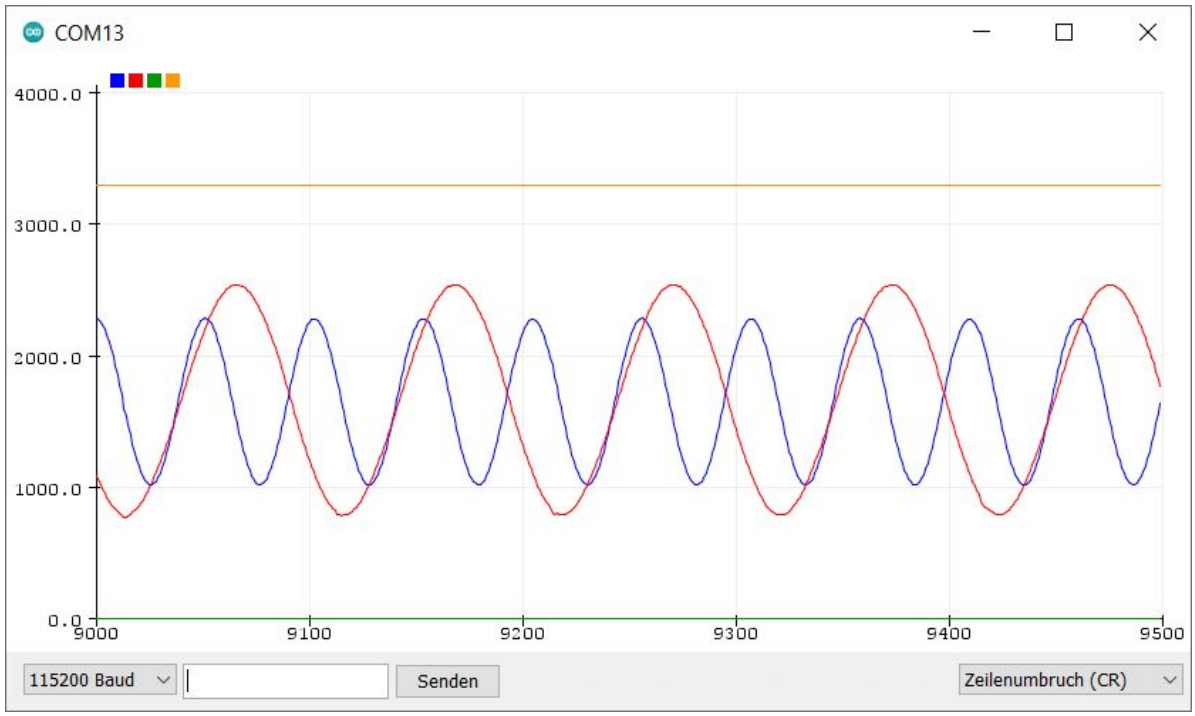
Сумма двух колебаний 1200 кГц и 1600 Гц снова приводит к периодической форме кривой, которая повторяется после нескольких колебаний. Паттерн повторяется быстрее, если добавляются точные кратные основной частоте. Вторая осциллограмма показывает смесь 1,5 кГц и 3 кГц.


COM13
2000.0
1600.0
1200.0
800.0
400.0
160000
160100
160200
160300
160400
160500
115200 Baud
Senden
Zeilenumbruch (CR)

30 Двухканальный DDS и осциллограф

Генератор DDS расширен до двух независимых каналов, а осциллограф также получает второй канал. Чтобы два измерения AD уместились во временном окне прерывания ШИМ, частоту ШИМ пришлось снизить до 100 кГц. Таким образом, измерения производятся с интервалом 10 мкс, так что участок шкалы со 100 точками измерения в окне плоттера занимает всего 1 мс.

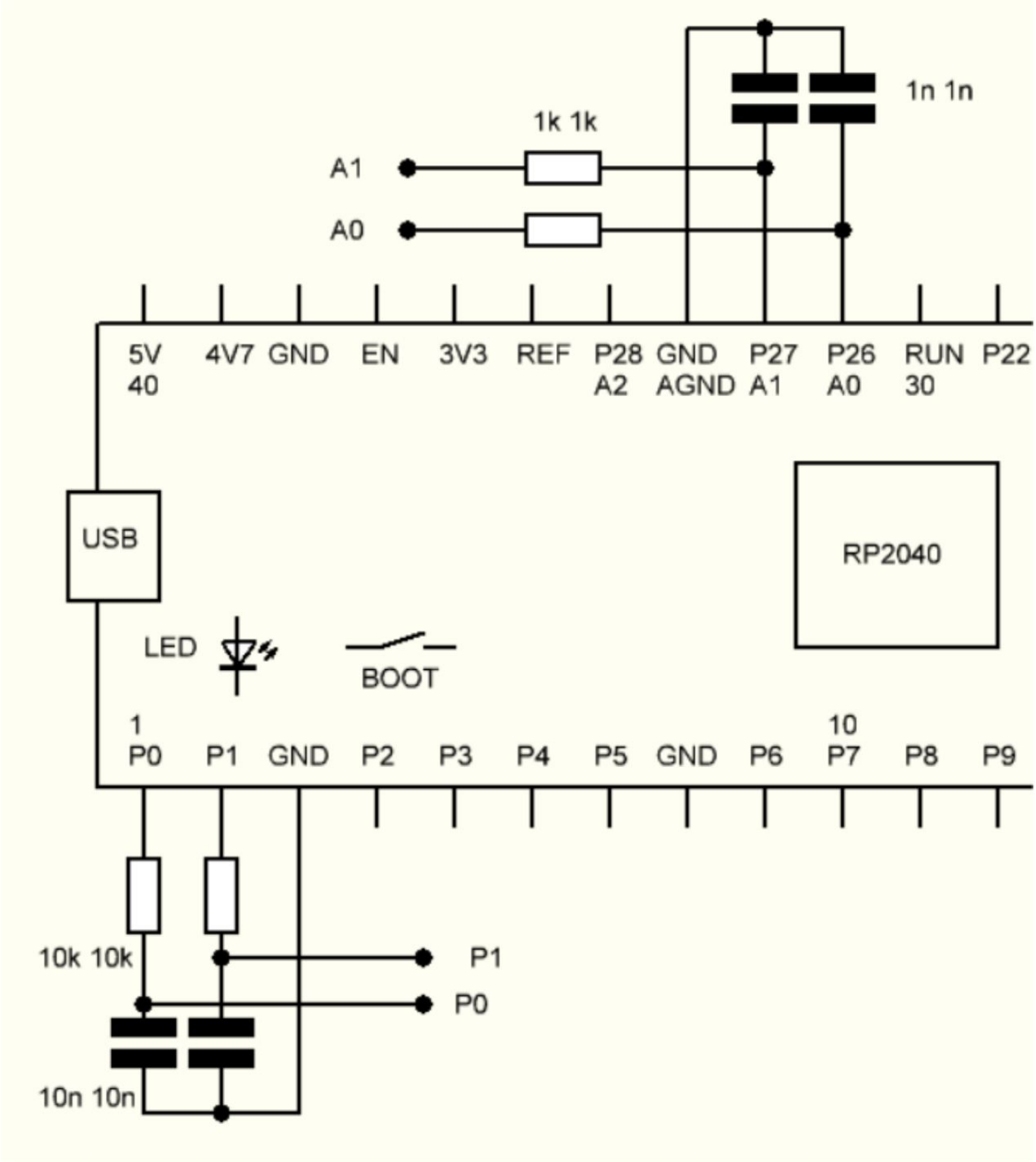
При запуске на выходах ШИМ 0 и 1 генерируются два синусоидальных сигнала с частотами 1 кГц и 2 кГц. Осциллограф работает с частотой 10 мкс, т. е. 1 мс/дел.



Однако три важных параметра можно изменить с помощью последовательных команд последовательного плоттера:

f500 устанавливает выход P0 на 500 Гц. Диапазон 2 Гц ...10 кГц
g5000 устанавливает выходной сигнал P1 на 5 кГц. Диапазон 2 Гц ...10 кГц

t10 устанавливает отклонение 10 мс/дел. Диапазон 1...100 мс/дел.



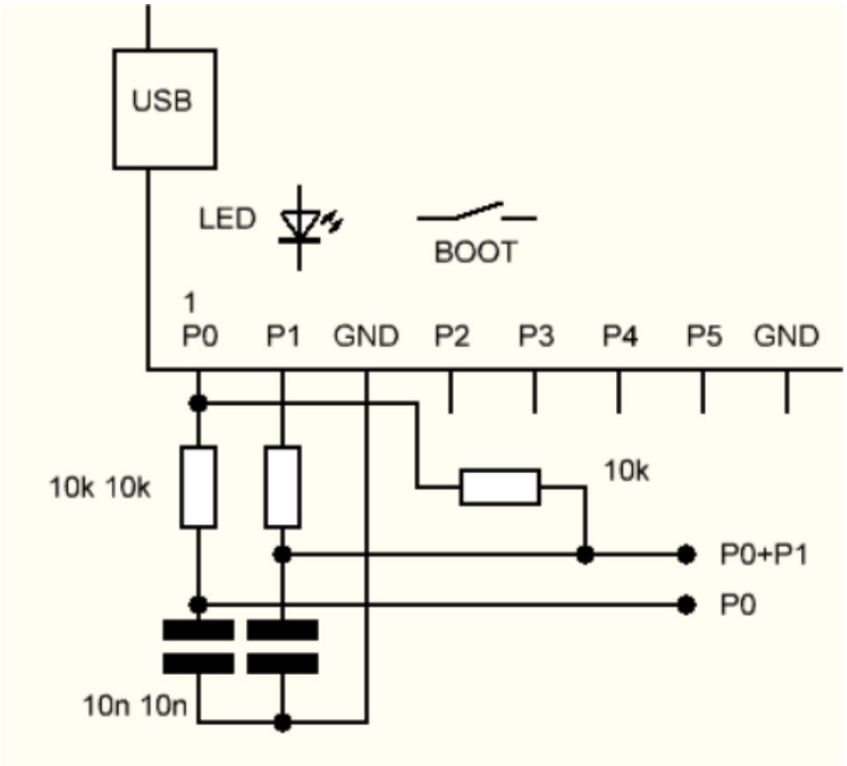

```
//Pico_ScopeDDS2 Dual channel//Commands
: f1000, g2000, t1: 1000Hz, 2000Hz, 1ms/div
#include "pico/stdlib.h"
#include "hardware/adc.h"
#include "hardware/pwm.h"
```

```
#include "pico/multicore.h"

unsigned int scope[1000];
unsigned int dds[1024];
uint16_t akku1, akku2, i;
unsigned int d, f1, f2, dp1, dp2, t, t0;

void setup() {
  f1=1000;
  dp1=f1*64/100;
  f2=2000;
  dp2=f2*64/100;
  Serial.begin(115200);
  for (int j = 0; j < 1024; j++){
    dds[j]=625+400* sin(2*PI*j/1024.0);
  }
  while(true){
    if (Serial.available()){
      char ch = Serial.read();
      uint32_t d = Serial.parseInt();
      if (ch==102) dp1=d*64/100; //f
      if (ch==103) dp2=d*64/100; //g
      if (ch==116) t0=d; //t
    }
    i=0;
    sleep_ms(1000);
    for (int j = 0; j < 500; j++){
      Serial.print ((int) scope[2*j]*3300/4095);
      Serial.print (" ");
      Serial.print ((int) scope[2*j+1]*3300/4095);
      Serial.print (" ");
      Serial.print (0);
      Serial.print (" ");
      Serial.println (3300);
    }
  }
}
```

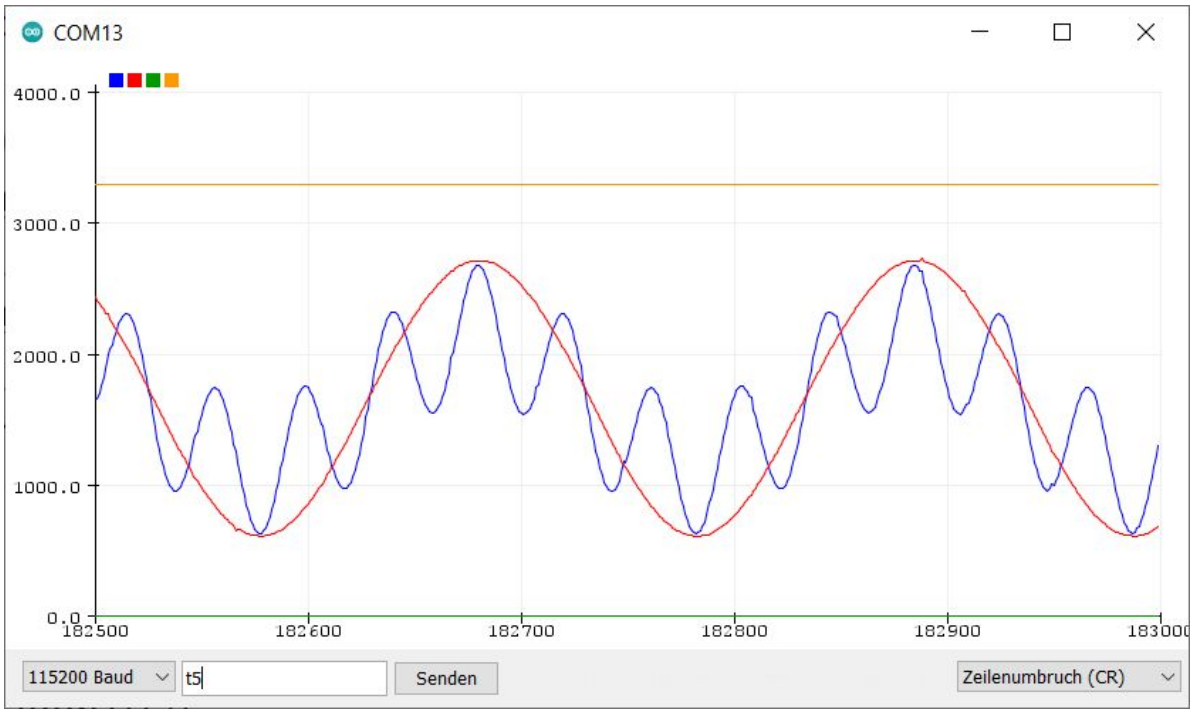
```
void loop() {}

void setup1() {
  adc_init();
  adc_gpio_init(26);
  adc_gpio_init(27);
  adc_select_input(0);
  gpio_set_function(0, GPIO_FUNC_PWM);
  gpio_set_function(1, GPIO_FUNC_PWM);
  pwm_set_wrap(0, 1249); //PWM 100 kHz
  pwm_set_gpio_level(0, 624);
  pwm_set_gpio_level(1, 624);
  pwm_set_enabled(0, true);
  pwm_set_irq_enabled(0, true);
  irq_set_exclusive_handler(PWM_IRQ_WRAP, pwm_int);
  irq_set_enabled(PWM_IRQ_WRAP, true);
  while (true);
}

void pwm_int() {
  pwm_clear_irq(0);
  pwm_set_gpio_level(0,dds[akku1>>6]);
  pwm_set_gpio_level(1,dds[akku2>>6]);
  akku1+=dp1;
  akku2+=dp2;
  t++;
  if(t>=t0){
    t=0;
    if (i<500){
      adc_select_input(0);
      scope[2*i]=adc_read();
      adc_select_input(1);
      scope[2*i+1]=adc_read();
      i++;
    }
  }
  }
```

void loop1() {}

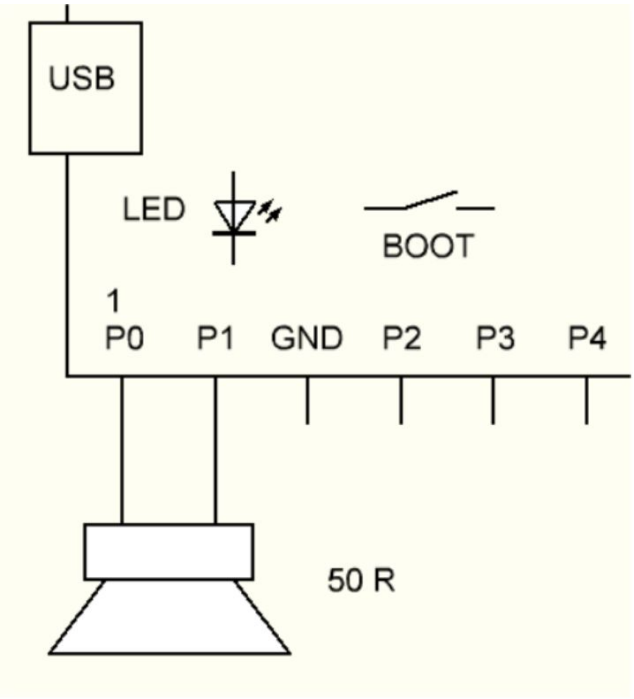
Два синусоидальных сигнала можно соединить вместе с помощью резисторов. Таким образом можно сложить две равные или разные амплитуды.



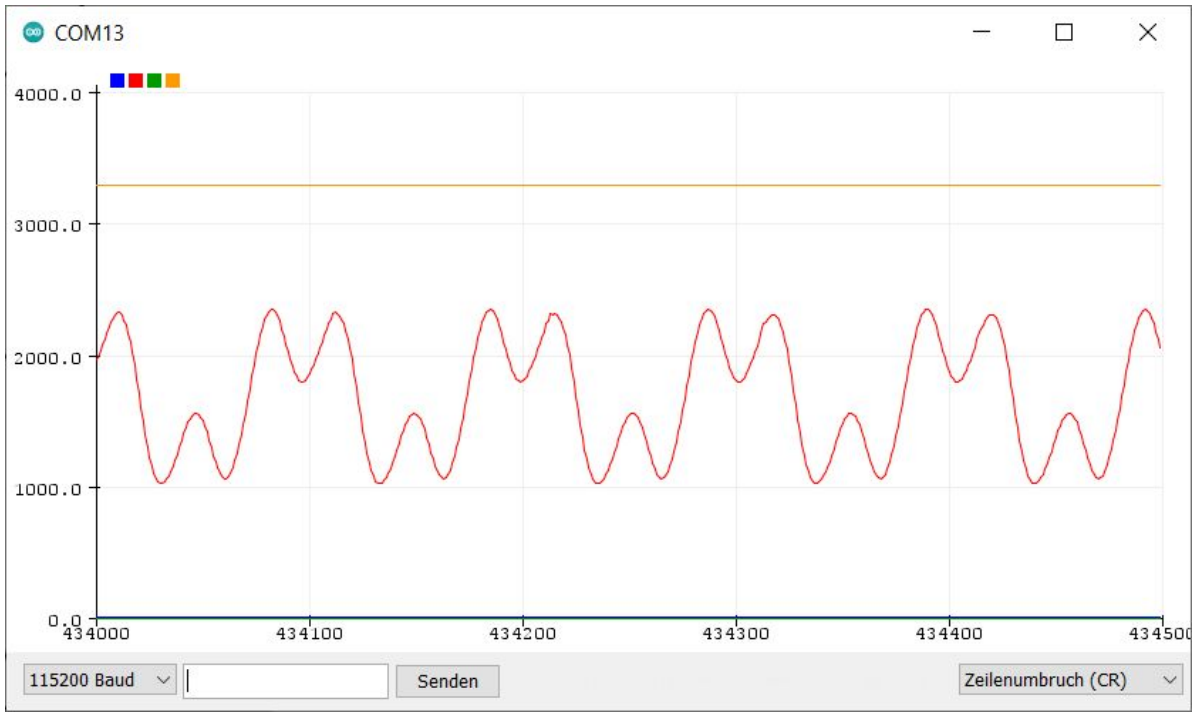
Пример измерения показывает базовую частоту 100 Гц и сумму второго сигнала 500 Гц. Время отклонения было установлено на 5мс/дел.



Как и в случае с усилителем класса D, громкоговоритель с высоким сопротивлением может управляться напрямую. Это позволяет создавать особые звуки.



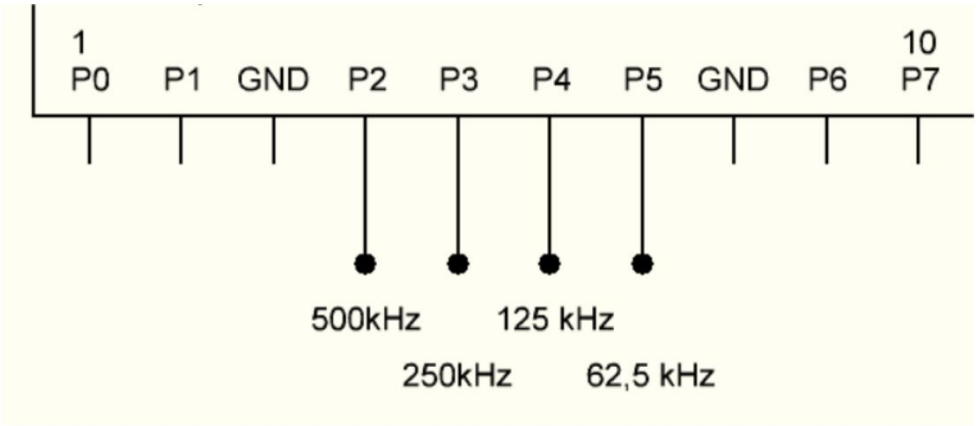
В принципе, любой сигнал может состоять из синусоидальных колебаний подходящей частоты и фазы. Следующий пример с двумя сигналами 1 кГц и 3 кГц показывает приближение к прямоугольному сигналу.



31 Делитель частоты PIO

В цифровой электронике известны многокаскадные двоичные счетчики типа CD4040. Каждая ступень делит частоту на два. После четырех ступеней входной сигнал делился на 16. Если каждая ступень имеет свой выход, также доступны прямоугольные сигналы с f/2, f/4 и f/8.

Здесь частоту 1 МГц следует разделить на 2, 4, 8 и 16. Благодаря блокам PIO (программируемого ввода-вывода) RPi Pico имеет подсистемы, с помощью которых можно решить эту задачу. Таким образом, у вас есть что-то вроде маленьких, очень быстрых и независимых микроконтроллеров с ограниченным набором команд, специально оптимизированных для прямого доступа к портам.



Требуется очень простая программа ассемблера PIO, которая одновременно управляет четырьмя выходами. Здесь в порты выводятся просто возрастающие числа. Каждый выход занимает только один такт, а тактовая частота может быть равна частоте системы или ее части. Цикл, или возврат в начало, не требует дополнительного времени, поэтому программа работает как цепочка из четырех двоичных делителей.

```
.program div16

.wrap_target

set pins, 0
set pins, 1
set pins, 2
set pins, 3
set pins, 4
set pins, 5
set pins, 6
set pins, 7
set pins, 8
set pins, 9
set pins, 10
set pins, 11
set pins, 12
set pins, 13
set pins, 14
set pins, 15

.wrap
```

Программу div16.pio необходимо скомпилировать с помощью ассемблера pioasm.exe, который включен в SDK как отдельный инструмент. Ассемблер вызывается через командную строку. При вызове pioasm div16.pio div16.h переведенная программа появляется в заголовочном файле div16.h. Программа, исполняемая модулем PIO, состоит из 16 команд в виде 32-битных чисел, каждая из которых требует только одного такта.

```
// ---------------------------------------------- //
// This file is autogenerated by pioasm; do not edit! //
// ---------------------------------------------- //

#pragma once

#if !PICO_NO_HARDWARE
#include "hardware/pio.h"
#endif

// ----- //
// div16 //
// ----- //

#define div16_wrap_target 0
#define div16_wrap 15

static const uint16_t div16_program_instructions[] = {
            // .wrap_target
    0xe000, // 0: set pins, 0
    0xe001, // 1: set pins, 1
    0xe002, // 2: set pins, 2
    0xe003, // 3: set pins, 3
    0xe004, // 4: set pins, 4
    0xe005, // 5: set pins, 5
    0xe006, // 6: set pins, 6
    0xe007, // 7: set pins, 7
    0xe008, // 8: set pins, 8
    0xe009, // 9: set pins, 9
    0xe00a, // 10: set pins, 10
    0xe00b, // 11: set pins, 11
    0xe00c, // 12: set pins, 12
    0xe00d, // 13: set pins, 13
    0xe00e, // 14: set pins, 14
    0xe00f, // 15: set pins, 15
            // .wrap
```

```
};

#if !PICO_NO_HARDWARE
static const struct pio_program div16_program = {
    .instructions = div16_program_instructions,
    .length = 16,
    .origin = -1,
};

static inline pio_sm_config div16_program_get_default_config(uint offset)
{
    pio_sm_config c = pio_get_default_sm_config();
    sm_config_set_wrap(&c, offset + div16_wrap_target, offset +
div16_wrap);
    return c;
}
#endif
```

The header file div16.h must be included in the C program. With pio_gpio_init, control over individual ports is transferred to the PIO. Other functions tell the PIO program itself which ports it should use.

```
//Pico_PIOdiv16 4 outputs
#include "pico/stdlib.h"<
#include "div16.h"

void setup() {
pio_gpio_init(pio0, 2);
pio_gpio_init(pio0, 3);
pio_gpio_init(pio0, 4);
pio_gpio_init(pio0, 5);
uint offset = pio_add_program(pio0, &div16_program);
pio_sm_set_consecutive_pindirs(pio0, 0, 2, 4, true);
pio_sm_config c = div16_program_get_default_config(offset);
sm_config_set_set_pins(&c, 2, 4);
sm_config_set_clkdiv(&c, 125);
```

```
pio_sm_init(pio0, 0, offset, &c);
pio_sm_set_enabled(pio0, 0, true);
while(true);
}

void loop() {
}
```

Решающая настройка частоты сигналов производится функцией sm_config_set_clkdiv(&c, 125). Таким образом, системная частота PIO делится на 125, так что тактовая частота составляет 1 МГц. С помощью pio_sm_set_enabled(pio0, 0, true) PIO наконец включается. Это создает нужные сигналы на выходах.

32 Логический анализатор

RPi Pico может запрашивать все порты одновременно с помощью функции SDK gpio_get_all(). Это позволяет построить быстродействующие логические анализаторы. Представленный здесь пример оценивает первые восемь портов от P0 до P7 и отображает изменяющиеся состояния последовательного плоттера. Программа PIO div16, которая генерирует четыре сигнала от P2 до P5, снова работает параллельно. На этот раз тактовая частота PIO установлена на 100 кГц. Таким образом, самая высокая частота на P2 — 50 кГц, самая низкая на P5 — 6,25 кГц.

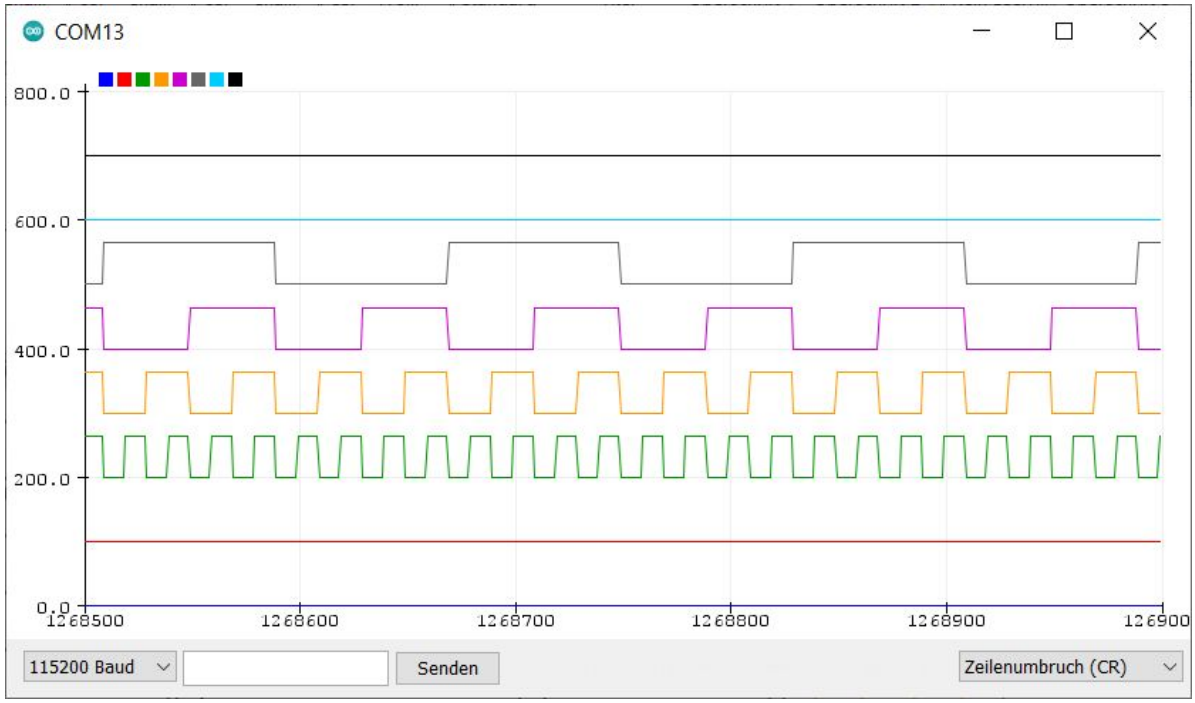
Хотя эти четыре порта работают как выходы, их состояния можно считывать параллельно. Программа PIO работает полностью независимо от обоих ядер процессора и не влияет на тайминги. С помощью этого метода можно также отслеживать другие программы и проверять сигналы на интерфейсах всех типов. Кстати, все подтягивающие резисторы на входах остаются включенными, так что каналы, которые не подключены, достоверно показывают ноль.

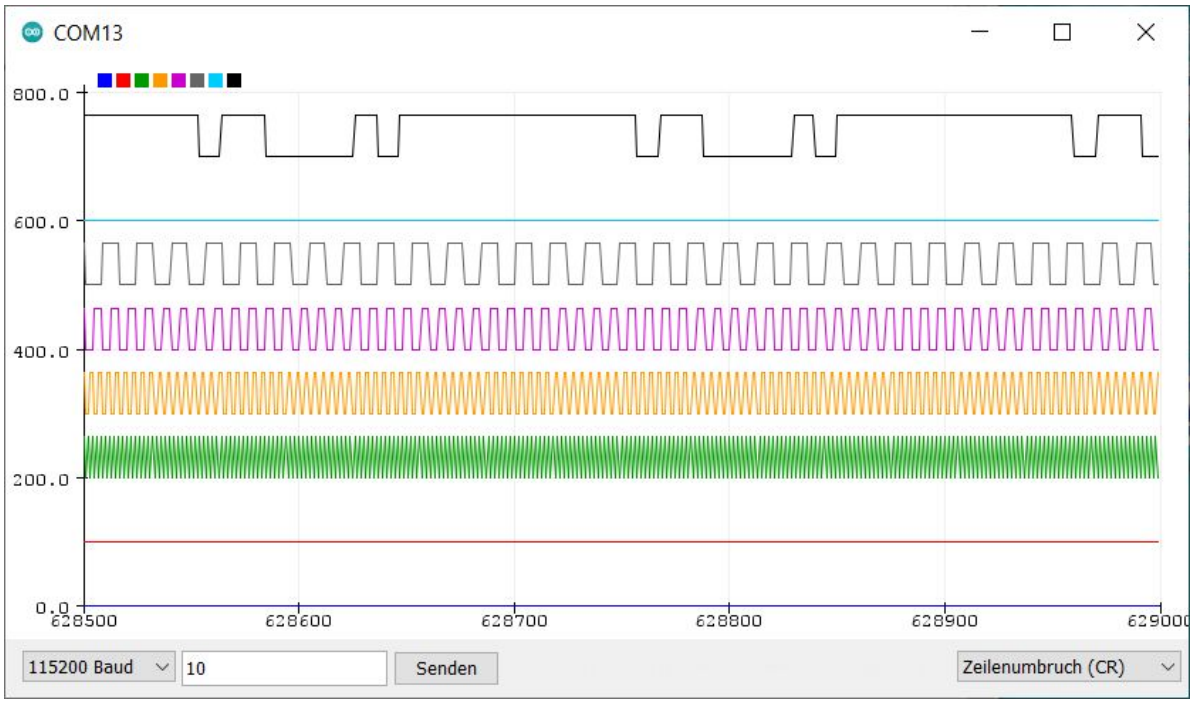
```
//Pico_Logik 8 inputs
#include "pico/stdlib.h"
#include "div16.h"

void setup() {
  Serial.begin(115200);
  pio_gpio_init(pio0, 2);
  pio_gpio_init(pio0, 3);
  pio_gpio_init(pio0, 4);
  pio_gpio_init(pio0, 5);
  uint offset = pio_add_program(pio0, &div16_program);
  pio_sm_set_consecutive_pindirs(pio0, 0, 2, 4, true);
  pio_sm_config c = div16_program_get_default_config(offset);
  sm_config_set_set_pins(&c, 2, 4);
  sm_config_set_clkdiv(&c, 1250);
  pio_sm_init(pio0, 0, offset, &c);
  pio_sm_set_enabled(pio0, 0, true);
  while(true);
}
```

```
void loop() {}

void setup1() {
  uint32_t ports[1000];
  int dt=1;
  while (true) {
    if (Serial.available()){
      int dt2 = Serial.parseInt();
      if( dt2>0) dt=dt2;
    }
    uint32_t t = time_us_32();
    for (int n = 0; n < 500; n++){
      ports[n]=gpio_get_all();
      t+=dt; while (t>time_us_32());
    }
    for (int n = 0; n < 500; n++){
      Serial.print (64*(ports[n]&1)) ;
      Serial.print (" ");
      Serial.print (100+32*(ports[n]&2)) ;
      Serial.print (" ");
      Serial.print (200+16*(ports[n]&4)) ;
      Serial.print (" ");
      Serial.print (300+8*(ports[n]&8)) ;
      Serial.print (" ");
      Serial.print (400+4*(ports[n]&16)) ;
      Serial.print (" ");
      Serial.print (500+2*(ports[n]&32)) ;
      Serial.print (" ");
      Serial.print (600+1*(ports[n]&64)) ;
      Serial.print (" ");
      Serial.println (700+(ports[n]&128)/2) ;
    }
    sleep_ms(1000);
  }
}
void loop1() {
}
```

Логический анализатор после запуска работает с периодом выборки 1 мкс. Однако его можно изменить с помощью последовательных команд последовательного плоттера. Фактически, анализатор все равно может работать намного быстрее, если вы уберете цикл задержки. Однако во многих случаях точная синхронизация с микросекундными интервалами выгодна.

Первое измерение показывает сигналы делителя частоты PIO. При периоде выборки 1 мкс ось X покрывает 50 мкс/дел. Самый быстрый сигнал показывает пять колебаний в одном делении шкалы. Таким образом, измерение подтверждает частоту ровно 100 кГц. Также ясно видно, что каждый спадающий фронт выходного сигнала вызывает изменение уровня на следующем выходе.





Второй результат измерения показывает сигнал последовательного интерфейса на скорости 9600 бод, который наблюдался через вход D7. Для измерения период выборки пришлось увеличить до 10 мкс. Тогда одно деление шкалы соответствует одной миллисекунде. Нетрудно заметить, что между двумя символами были вставлены паузы в 1 мс. Сигнал TXD инвертируется, поскольку обычно генерируется UART микроконтроллера. Поэтому уровень покоя высокий. Сами символы содержат стартовый бит в начале и восемь последующих битов данных. В данном случае был отправлен байт 67. Сигналы PIO на схеме можно оставить в виде отдельной временной сетки для сравнения. Если они мешают или нужны все доступные входы, их можно отключить, закомментировав строку pio_sm_set_enabled(pio0, 0, true).