



nwwind 2 января в 17:43

# МКА (машина конечных автоматов) для чайников на примере класса «кнопка» в arduino

Программирование микроконтроллеров, C++

## Зачем всё это нужно?

Когда чайник, уперевшись в необходимость отойти от простой последовательности действий, задаёт на хабре вопрос типа "как сделать вот ему с вероятностью 70% отвечают "погугли конечные автоматы" и 30% "используй finite state machine" в зависимости от страны работодателя профессионала. На следующий вопрос "а как?" отправляют в гугл. Идёт такой чайник, что только закончил мигать светодиодом и вытер пол лба, что учил в школе немецкий и всю жизнь работал бульдозеристом в этот гугл и видит там статьи типа [Википедия про конечные автомат](#) формулами и в которых понятны только предлоги.

Так как я тоже чайник, но до бульдозера работал программистом 30 лет назад, наступив на множество граблей по мере освоения программирования микроконтроллеров, решил написать эту статью простым языком для начинающих.

Итак, задача стоит, например, научить ардуину понимать нажатие кнопки типа клик, нажатие и удержание, то есть, короткий тык в кнопку, д. и очень длинный. Любый начинающий может сделать это в функции loop() и гордиться этим, но как быть, если кнопок несколько? А если пр надо не мешать выполнению другого кода? Я написал библиотеку SmartButton для Arduino, чтобы потом не возвращаться к теме кнопок. З. напишу, как она работает.

## Что нам мешает жить и как с этим бороться?

Нам мешает жить функция delay(), я её поборол при помощи МКА и написал свой класс SmartDelay. Я уже сделал новый класс (SmartDelay порождённый от SmartDelay, там всё то же самое, но в миллисекундах. В данной статье я не использую эту библиотеку, просто хвастаюсь.

Чтобы не утомлять читателя повторно рассказом о том, как плохо пользоваться функцией delay() и как жить без неё, я рекомендую сначала прочитать мою старую статейку [Замена delay\(\) для неблокирующих задержек в Arduino IDE](#). Я немножко повторю основные тезисы здесь е ниже по мере написания кода.

## Немного теории

Так или иначе, но у вас есть **объекты**. Вы можете называть их кнопками, дисплеями, светодиодными лентами, роботами и измерителем ур воды в бачке. Если ваш код выполняется не в одну ниточку последовательно, а по каким-то событиям, вы храните **состояние** объектов. Вы можете называть это как угодно, но это факт. Для кнопки есть, например, состояния "нажата" и "отпущена". Для робота, например: стоит, ед. прямо, поворачивает. Количество этих состояний конечно. Ну, в нашем случае, да. Добавим состояния "клик", "нажатие" и "удержание". Уже состояний, которые нам надо различать. Причём, интересны нам лишь последние три. Этими пятью состояниями живёт кнопка. В страшно внешнем мире происходят события, которые от кнопки в общем-то никак не зависят: тыкания в неё пальцем, отпускания, удержания на раз. время итп. Назовём их **событие**.

Итак, мы имеем **объект** "кнопка" у которого конечное количество **состояний** и на него действуют **события**. Это и есть **конечный автомат** теории КА список возможных событий называют словарём, а события словами. Я дико извиняюсь, я это проходил 30 лет назад и плохо пом терминологию. Вам же не терминология нужна, правда?

В данной статье мы напишем замечательный код, который будет переводить наш КА из состояния в состояние в зависимости от событий. ( является собственно **машиной конечных автоматов (МКА)**).

Не обязательно всё ваше устройство запихивать в один огромный КА, можно разбить на отдельные объекты, каждый из которых обслужив. своей МКА и даже находится в отдельном файле кода. Многие свои объекты вы сможете поместить на GitHub в виде готовых библиотек ар и использовать потом не вникая в их реализацию.

Если ваш код не требует переносимости и повторного использования, можно всё лепить в один большой файл, что и делают как раз начин.

## Практика

Основой проектирования МКА является таблица.

- По горизонтали пишем все-все возможные события.

- По вертикали все состояния.
- В клетках действия. Переход в новое состояние — это тоже действие.

Для кнопки это будет выглядеть вот так:

Состояние \ Событие	Нажата	Отпущена	Нажата 20мс	Нажата 250мс	Нажата 1с	Нажат
Не Нажата						
Кликнута						
Нажата						
Удержана						
Слишком долго удержана						

Зачем так сложно? Надо же описать все возможные состояния и учесть все возможные события.

События:

- Нажата — это когда мы обнаружили, что контакт замкнут.
- Отпущена — это когда выяснилось, что контакт разомкнут.
- Нажата 20мс — надо учестьдребезг контактов и игнорировать размыкание какое-то время, дальше уже считается, что клик.
- Нажата 250мс — для клика 250мс достаточно, всё что дольше — это уже нажатие.
- Нажата 1с — больше 1с это уже удержание.
- нажата 3с — удержание больше 3с посчитаем ошибкой.

Время вы можете поставить своё, как вам удобно. Чтобы не менять потом в разных местах цифры, лучше сразу заменить их буквами :)

```
#define SmartButton_debounce 20
#define SmartButton_hold 250
#define SmartButton_long 1000
#define SmartButton_idle 3000
```

Состояния:

- Не нажата — рапортуем, что никто кнопочку не трогал.
- Дребезг — Ждём конец дребезга контактов.
- Кликнута — был клик.
- Нажата — кнопку нажали.
- Удержана — кнопку нажали и подержали.
- Слишком долго удержана — что-то пошло не так, предлагаю отказаться от нажатия в этом случае.

Итак, переводим русский язык на язык C++

```
// Нога контроллера для кнопки
byte btPin;
// События
enum event {Press, Release, WaitDebounce, WaitHold, WaitLongHold, WaitIdle};
// Состояния
enum state {Idle, PreClick, Click, Hold, LongHold, ForcedIdle};
// Текущее состояния
```

```
enum state btState = Idle;
// Время от нажатия кнопки
unsigned long pressTimeStamp;
```

Обращу внимание, что и события и состояния указаны не как целая переменная типа byte или int, а как enum. Такая запись не столь любим профессионалами так как не даёт экономить биты столь дорогой в микроконтроллерах памяти, но, с другой стороны очень наглядна. При использовании enum нас не заботит каким числом закодирована каждая константа, мы пользуемся словами, а не числами.

#### Как расшифровать enum?

Давайте уже заполним таблицу. Знаком -> я обозначу переход в новое состояние.

Состояние \ Событие	Нажата	Отпущена	Нажата 20мс	Нажата 250мс	Нажата 1с	Нажата 3с
Не Нажата	->Дребезг	->Не Нажата				
Дребезг		->Не Нажата	->Кликнута			
Кликнута		->Не Нажата		->Нажата		
Нажата		->Не Нажата			->Удержана	
Удержана		->Не Нажата				->Слишком долго удерж
Слишком долго удержана		->Не Нажата				

Для реализации таблицы мы сделаем функцию void doEvent(enum event e). Эта функция получает событие и выполняет действие как описано в таблице.

## Откуда берутся события?

Настало время ненадолго покинуть нашу уютную кнопку и окунуться в ужасный внешний мир функции loop().

```
void loop() {
    // запоминаем текущий счётчик времени
    unsigned long mls = millis();
    // генерим события
    // нажатие кнопки
    if (digitalRead(btPin)) doEvent(Press)
    else doEvent(Release)
    // ожидание дребезга прошло
    if (mls - pressTimeStamp > SmartButton_debounce) doEvent(WaitDebounce);
    // ожидание нажатия прошло
    if (mls - pressTimeStamp > SmartButton_hold) doEvent(WaitHold);
    // ожидание удержания прошло
    if (mls - pressTimeStamp > SmartButton_long) doEvent(WaitLongHold);
    // совсем перебор по времени
    if (mls - pressTimeStamp > SmartButton_idle) doEvent(WaitIdle);
}
```

Итак, имея под рукой генератор событий, можно приступить к реализации МКА.

На самом деле, события могут генерироваться не только по подъёму уровня сигнала на ноге контроллера и по времени. События могут передаваться одним объектом другому. Например, вы хотите, чтобы нажатие одной кнопки автоматически "отпускало" другую. Для этого достаточно вызвать её doEvent() с параметром Release. МКА — это же не только кнопки. Превышение порога температуры является событием для МКА и вызывает включение вентилятора в теплице, к примеру. Датчик неисправности вентилятора меняет состояние теплицы на "авария" и так далее.

## Три подхода к реализации таблицы в код

План-A: if {} elseif {} else {}

Это первое, что приходит в голову начинающему. На самом деле, это неплохой вариант, если в таблице совсем мало заполненных клеток и действия сводятся только к смене состояний.

```
void doAction(enum event e) {
    if (e == Press && btState == Idle) { // нажата кнопка в состоянии кнопка отпущена
        btState=PreClick; // переходим в состояние ожидания дребезга
        pressTimeStamp=millis(); // запоминаем время нажатия кнопки
    }
    if (e == Release) { // отпущена кнопка
        btState=Idle; // Переходим в состояние кнопка отпущена
    }
    if (e == WaitDebounce && btState == PreClick) { // Прошло время дребезга
        btState=Click; // считаем, что это клик
    }
    // и так далее для всех сочетаний событий и состояний
}
```

Плюсы:

- Просто писать новичку, который впервые в руки взял ардуину.
- Первое, что приходит в голову.

Минусы:

- Сложно читать код.
- Сложно модифицировать код, если что-то поменялось в таблице.
- Ад, если большая часть клеток таблицы заполнена.

## План-Б: Таблица указателей на функции

Другая крайность — это таблица переходов или функций. Я про такой подход писал в статье [Обработка нажатий кнопок для Arduino](#). Скрес ООП и МКА.. В двух словах, вы создаёте для каждой **разной** клетки таблицы отдельную функцию и делаете таблицу из указателей на них.

```
// определяем тип, так проще
typedef void (*MKA) (enum event e);
// делаем таблицу
MKA action[6][6]={
    {&toDebounce,&toIdle,NULL,NULL,NULL,NULL},
    {NULL,&toIdle,&toClick,NULL,NULL,NULL},
    {NULL,&toIdle,NULL,&toHold,NULL,NULL},
    {NULL,&toIdle,NULL,NULL,&toLongHold,NULL},
    {NULL,&toIdle,NULL,NULL,NULL,&toVeryLongHold},
    {NULL,&toIdle,NULL,NULL,NULL,NULL}
};

// функция doEvent получается совсем простой
void doEvent(enum event e) {
    if (action[btState][e] == NULL) return;
    (*(action[btState][e]))(e);
}

// Примеры функций из таблицы

// В состояние "не нажата"
void toIdle(enum event e) {
    btState=Idle;
}

// В состояние ожидания дребезга
void toDebounce(enum event e) {
    btState=PreClick;
    pressTimeStamp=millis();
}

// и так далее
```

### Что за странные слова и значки здесь использованы?

Плюсы:

- Очень читаемый код. Не смейтесь. Достаточно один раз вкурить про массивы с указателями и вы получите простой красивый код. У вас табличка как табличка, ячейки отдельно расписаны — красота.
- Легко писать.
- Очень легко модифицировать логику.

Минусы:

- Жрёт память. Каждый указатель — это от двух до четырёх байтов.  $6 \times 6 = 36$  клеток занимают от 72 до 144 байтов памяти и это на каждую кнопку, а если их, скажем, четыре? Напомню, у вас всего 2048 байтов памяти для популярных чипов ардуины.

Минус оказался таким жирным, что я после написания Обработка нажатий кнопок для Arduino. Скрестить ООП и МКА. и первого же примера кода в деле выпилил это всё нафиг. В итоге я пришёл к золотой середине "switch { switch {}".

### План-В: Золотая середина или switch { switch {} }

Самый распространённый, средний вариант — это использование вложенных операторов switch. На каждое событие как case оператора switch надо писать switch с текущим состоянием. Получается длинно, но более-менее понятно.

```
void doEvent(enum event e) {
    switch (e) {
        case Press:
            switch (btState) {
                case Idle:
                    btState=PreClick;
                    pressTimeStamp=millis();
                    break;
            }
            break;
        case Release:
            btState=Idle;
            break;
        // ... так далее ...
    }
}
```

Умный компилятор всё-равно построит таблицу переходов, как описано в плане Б выше, но эта таблица будет в флеше, в коде и не будет занимать столь дорогую нам память переменных.

Плюсы:

- Быстрый код.
- Экономится память.
- Логика понятна. Можно читать.
- Легко менять логику.

Минусы:

- Портянка в несколько экранов при большом количестве состояний и событий, которую сложно распилить на отдельные файлы.

На самом деле, можно использовать план-В и план-А, то есть, switch по событиям, но if внутри по состояниям. На мой вкус, switch и на то и понятнее и удобнее, если потребуется что-то потом поменять.

### Куда вставлять мой код, который зависит от нажатий кнопки?

В табличке мы не учитывали наличие других МКА сосредоточившись на нашем обработчике событий кнопки. На практике же требуется не радоваться тому, что кнопка работает, но и выполнять другие действия.

У нас есть два важных места для вставки хорошего кода:

```
case Release:
    switch(btState) {
        case Click:
            // Вот здесь был клик и отпустили кнопку. Это точно был клик.
            break;
```

```
case WaitDebounce:
    switch (st) {
        case PreClick:
            btState=Click;
            // Вот здесь случился клик. Кнопка может быть потом ещё нажата и будет ещё и "нажатие" итп.
            break;
```

Лучше всего, чтобы не портить красоту, написать свои функции типа `offClick()` и `onClick()`, в которых будет обрабатываться уже эти события возможно, что они передадут его другому МКА :D

Я обернул всю логику работы МКА кнопки в класс C++. Это удобно, так как не отвлекает от написания основного кода, все кнопки работают самостоятельно, мне не надо изобретать имена кучи переменных. Только это совсем другая история и я могу написать, как оформлять ваш в классы и делать из них библиотеки для Arduino. Пишите в комментариях пожелания.

## Что в итоге?

В итоге, в теории, после прочтения этой статьи должно появиться желание заменить в любимом ардуиновом скетче свой нечитаемый глюковкод на красивый структурированный и с использованием МКА.

Обратите внимание, что в функции `loop()`, которая в нашем случае лишь генерит события, нет ни одной задержки. Таким образом, после приведённого кода можно писать что-то ещё: генерить события для других МКА, выполнять ещё какой-то свой код, не содержащий `delay()`;

Я очень надеюсь, что эта статья помогла вам понять, что такое МКА и как их реализовать в вашем скетче.

Полезна ли статья для освоения ардуины?

- ☐ Да, однозначно.
- ☐ Понял то, что не долго смог осилить сам.
- ☐ Подписался на гитхабе заодно.
- ☐ Фигня, таких статей полно.
- ☐ Мне не интересно.
- ☐ Что такое «микроконтроллер»?
- ☐ Что такое «программирование»?

Голосовать

Воздержаться

Проголосовали 78 пользователей. Воздержались 37 пользователей.

**Метки:** arduino, fsm, мка, конечный автомат, c++, программирование микроконтроллеров, чайник, примеры, обучение

↑ +12 ↓ 148 22,7k 28



↑ 22,0 ↓

Карма

0,0

Рейтинг

31

Подписчики

✉ Написать

✍ Подписат

Сергей Келер @nwwind

Сисадмин

Поделиться публикацией

ПОХОЖИЕ ПУБЛИКАЦИИ

8 октября 2012 в 15:35

Пишем модуль на C++ для nodejs на примере работы с MySQL

↑ +32

👁 23,6k

📖 168

💬 8

5 июня 2012 в 22:58

Программирование микроконтроллеров семейства Cypress

↑ +6

👁 8,5k

📖 23

💬 10

29 января 2011 в 20:30

USB Toolstick, или программирование микроконтроллеров «для самых маленьких»

↑ +77

👁 20,8k

📖 145

💬 55

ВОПРОСЫ И ОТВЕТЫ

Топ

🔍 Arduino 🟢 Простой

1 с

Как распознать слова написанные ручкой с датчиком?

🔍 Arduino 🟢 Простой

1 с

Что почитать про разработку под Arduino для написания оптимального кода на C?

🔍 Arduino 🟢 Простой

2 от

Что обозначают надписи у пинов? Как они расшифровываются?

🔍 Arduino 🟢 Простой

1 с

Как подключить много/несколько датчиков к Raspberry Pi/Arduino?

🔍 Arduino 🟢 Простой

4 от

Во сколько примерно обойдётся стоимость самостоятельной разработки мини робота на arduino?

Все вопросы

Задать вопрос

Комментарии 28

Отслеживать новые в ☐ почте ☐

AMDDev 02.01.18 в 20:42

🔗 📖

↑

Спасибо за статью! Позволю себе пару замечаний. Если статья для чайников, то я бы добавил еще диаграмму состояний для большей наглядности статье вы вводите свои определения "... Назовём их событие..." Во избежании путаницы, думаю было бы здорово упомянуть, что из статьи является теории автоматов. Описать входной алфавит, множество конечных состояний,... в терминах из теории.

Ответить

nwwind 02.01.18 в 21:07

🔗 📖 📄 ↻

↑

Думаю да, чтоит картинок добавить.

Я хотел мультфильм, но это уж для совсем-совсем чайников получится.

Теорию как раз не хотел привлекать тк всё-равное ЦА её читать не будет. Старался понятным непосвящённому языком писать.

Ответить

https://habr.com/post/345960/

7/12

 **Goron\_Dekar** 02.01.18 в 21:07   


А ещё таблицу указателей можно засунуть во флеш микроконтроллера (прогмет в терминах ATMEGA, как у ардуины не знаю)

Ответить

 **nwwind** 02.01.18 в 21:09   

Можно засунуть в PROGMEM, но оттуда доставать адрес надо каждый раз вычисляя смещение.  
Красота же метода как раз в вызове из двумерного массива.

Ответить

 **Goron\_Dekar**  03.01.18 в 00:17   

Вместо обращения к элементу массива по индексу надо написать вызов специальной функции, вот и всё.  
action[btState][e] заменить на getAction(btState,e), которая возвращает ссылку на функцию.

Ответить

 **nwwind** 03.01.18 в 13:54   

Можно так, да. Хммм...  
Надо поэкспериментировать будет.

Ответить

 **voidptr0** 02.01.18 в 21:09   

В большинстве случаев все чайники сразу хотят реализовать меню. Мне почему-то кажется, что реализация через автоматы будет не очень простой наглядной.


Ответить

 **nwwind** 02.01.18 в 21:17   

Почему?

У меня есть меню на МКА. Там наоборот просто: Кнопки вверх, вниз и пыщ. Длинный пыщ вызывает меню, он же закрывает.  
Имеем меню с состоянием «текущий пункт» и событиями старт, вверх, вниз, энтер, выход. События прилетают от кнопочек, как я описывал в стат вложенном меню будет «текущий путь».  
По желанию можно добавить события «длинный вверх» и такой же вниз как перемотку в начало и конец.  
Я к тому, что, если абстрагироваться от конкретных пунктов меню и реализации движения строчек на экране — то всё получится логично и красиво.  
Хитрое поведение кнопок можно решить через мой класс SmartButton переопределяя его абстрактные методы, но это уже другая статья будет. Думаю написать вскоре.

Ответить

 **NelSon29** 02.01.18 в 23:16   

Спасибо за статью, но хочу отметить, что ардуиновский компилятор совместим с C++11, так что смело можно использовать enum class вместо enum значительно повышает читабельность кода и помогает избежать ошибок. И для констант стоит использовать const, а не препроцессор

Ответить

 **nwwind** 03.01.18 в 13:58   

enum class — интересно. можно указать byte вместо int.  
работает! спасибо. хехе, век живи, век учись.

Ответить

 **nwwind** 03.01.18 в 21:43   

А есть ли какая-то синтаксическая конструкция модная типа сишного

```
struct {  
    byte flag : 1,  
    byte pin : 4  
}
```

Вот применительно к enum class бы? Туда же упаковать?

А то 11 байтов много.

- 4 время, никуда не деться, хотя да, можно в 13 бит записать тк нас только хвост интересует от времени.
- 1 пин.
- 2 словарь.
- 2 статус.

Итого 9. Хмм... Где ещё два байта?  
Переписал на enum class input: byte { };



получилось 9 байт.  
Хмм... а должно быть 7.

Запустил новый вариант с enum class, мне понравилось.

Ответить



gorbln 02.01.18 в 23:23



Позволю себе критику. Ибо являюсь той самой целевой аудиторией.

Библиотека сделана не «по-ардуиновски». Слишком сложно всё организовано. Примеров нормальных нет. На гитхабе тоже.

```
void toDispToggle(enum state st, enum input in) {
  Serial << «toDispToggle » << st << " " << in << endl;
  if (dMode == 0 || dMode == 1) {
    dMode ^= 1;
  }
```

Вот это — НЕ arduino way.

Не должно быть в коде такого. Все эти функции должны быть как-то разделены. Отдельно обработчик кнопки, отдельно обработчик события. Посмотрите, как сделан Serial. Ничего лишнего. Я не должен парить себе мозги, а где же и что вызывается, и в какое место кода мне надо сунуть обработчик по приходу пакета. Я не должен создавать функции.

Как мне видится удобный обработчик клика в соответствии с концепцией Arduino.

В начале создаём объект типа smartbutton. Как-то типа «SmartButton mybutt».

А в loop мы получаем его состояние, и делаем нужные дела, типа «if (mybutt.pressed) {...}».

Можно сколько угодно говорить, что «а привязать функцию мне удобнее». Но чайникам — неудобно писать кучу кода с неочевидными аргументами.

Кстати, невозможность использовать delay — это как бы тоже не плюс. Smartdelay-ем невозможно пользоваться.

Ответить



nwwind 03.01.18 в 14:16



В статье написано не как сделать библиотеку, а как сделать МКА. То есть, вообще, а не только кнопку.

Как упаковать её в класс и спрятать в отдельную папку — я напишу в следующей статье.

Я сделал ООПный интерфейс к кнопкам, можно наследовать базовый абстрактный класс и делать свои классы кнопок для меню, для управления двойного клика итп.

В асинхронном режиме же, писать в loop() что-то типа if (bt.pressed()) { код } не очень правильно. Лучше код вставлять в том месте, где собственно случается событие. Без классов если, я указал где, для примера. Событий много, какие нужны лично вам, я угадать не могу.

В модели ООП для МКА с кнопкой я делаю так:

```
class toggleSmartButton: public SmartButton {
private:
  byte myToggle=0;
public:
  toggleSmartButton(int p) : SmartButton(p) {} // Надо вызвать родительский конструктор, увы.
  virtual void onClick(); // Метод, который переопределяем. В базе он пустой.
};

// делаем объект. пин12.
toggleSmartButton bt(12);

// Этот метод переключает по клику
void modeSmartButton::onClick() {
  if (myToggle) {
    // Код для включения
    Serial.println("ON");
  } else {
    // Код для выключения
    Serial.println("OFF");
  }
  myToggle=!myToggle;
}

void loop() {
  bt.run(); // Надо положить сюда этот вызов.
}
```

Функция delay() имеет удобство только для чайников или чего-то однопоточного и неспешного. Ну, нет многозадачности в ардуине, нет. delay() блокирует выполнение, чип тупо стоит и ждёт.

Для SmartDelay есть интерфейс ООП, пожалуйста, управляйтесь событиями, делайте МКА для последовательности кусков кода с задержками. Э ардуинский путь как раз.

```
код1  
delay(d1);  
код2  
delay(d2);
```

Можно преобразовать к МКА в четыре состояния: код1, ждём d1, код2, ждём d2 и ваш этот код будет выполняться не мешая задержками другому

Если же надо просто выполнять некий код раз в N миллисекунд, то SmartDelay вообще отлично подходит.

Ответить



gorb1n 04.01.18 в 14:33



Как упаковать её в класс и спрятать в отдельную папку — я напишу в следующей статье.

Я сделал ООПный интерфейс к кнопкам, можно наследовать базовый абстрактный класс и делать свои классы кнопок для меню, для управления, для двойного клика итп.

После следующей статьи стало лучше, спасибо!

Правда, чтобы сделать как Serial, всё равно приходится наворачивать лишнее, но это терпимо, и как там говорилось, при возникновении индивидуальной непереносимости — может быть убрано в отдельный файл.

В асинхронном режиме же, писать в loop() что-то типа if (bt.pressed()) { код } не очень правильно. Лучше код вставлять в том месте, где собственно случается событие.

Лучше, кто же спорит. Только вот понимание как организовать структуру кода, чтобы всё было на своих местах — это черта, далеко не присуща начинающим.

Функция delay() имеет удобство только для чайников или чего-то однопоточного и неспешного. Ну, нет многозадачности в ардуине, нет. delay блокирует выполнение, чип тупо стоит и ждёт.

Именно. Собственно, если я юзаю делей в программе — я понимаю, что ~~мне~~ потеряю всё, что происходило во время делея (поэтому я обычно пользуюсь делеем на 20-50 мс, либо вообще не использую его).

Отдельное спасибо за пример для ~~дебилев чайников~~ обычных людей ~~типа меня~~ в следующей статье. =)

Ответить



nwwind 04.01.18 в 20:07



да, с той статье специально вам написал :)

дал, так сказать, рыбу и показал, как использовать удочку.

чтобы сделать как Serial

Это как? Так?

```
if (button.available()) {  
    bt=button.read();  
}
```

Не получится. То есть, можно, если речь идёт о кнопках для калькулятора. Для всего прочего это неудобно, жрёт память и тормозит.

Ответить



nwwind 03.01.18 в 14:23



Примеры для SmartButton да, надо написать поинтереснее и на весь интерфейс класса. Согласен.

Ответить



nwwind 03.01.18 в 14:33



Как мне видится удобный обработчик клика в соответствии с концепцией Arduino.

В начале создаём объект типа smartbutton. Как-то типа «SmartButton mybutt».

А в loop мы получаем его состояние, и делаем нужные дела, типа «if (mybutt.pressed) {.....}»

Это не лучший подход. Особенно для любителей `delay()`. Прикинь, ты уже три раза нажал на кнопку и отпустил, а выполнение только добралось до твоего `if (key.pressed())` через задержки.

Я предлагаю асинхронное выполнение. То есть, клик обрабатывается в том месте, где случился. В МКА в статье, я указал место, где вставлять се код. В `SmartButton` нет функции `pressed`, но есть метод `onClick()` пустой, виртуальный, и его надо определить унаследовав класс. Как вариант, мож сделать колбеки, функции, что будут вызываться по событиям, но они жрут память тоже тк событий много, на каждое надо указатель и не все нуж

Ответить



**Vanellope** 03.01.18 в 03:17



Самого главного нет: сколько ресурсов ушло на опрос одной кнопки. ROM, RAM, процессорное время.

Ответить



**nwwind** 03.01.18 в 14:17



Много.

Это не самый эффективный код, конечно же.

Но с чего-то начать надо. Статья не про кнопку, а про МКА.

Ответить



**nwwind** 04.01.18 в 12:10



6 байтов на базовый класс уходит.

я посчитал в следующей статье.

такты не считал, конечно :)

Ответить



**EvilBeaver** 03.01.18 в 10:20



А нас учили, что основой КА является граф переходов, а не табличка. Граф можно представить в виде таблички, это да, но это другой вопрос...

Ответить



**nwwind** 03.01.18 в 14:18



Граф лично мне показался менее наглядным.

Выше уже писали, что неплохо было бы и графом проиллюстрировать. Я, наверное, так и сделаю.

Ответить



**nwwind** 03.01.18 в 19:07



Продолжение: [habrahabr.ru/post/346006](http://habrahabr.ru/post/346006)

Ответить



**juray** 04.01.18 в 15:10



вопрошающих новичков надо отправлять не в википедию, а к трудам Шалыто (ключевая фраза для поиска «SWITCH-технология»)

Ответить



**nwwind** 04.01.18 в 19:59

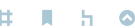


[ru.wikipedia.org/wiki/Switch-технология](http://ru.wikipedia.org/wiki/Switch-технология) — Оно?

Ответить



**juray** 15.01.18 в 13:20



она самая

Ответить



**TheSima** 04.01.18 в 22:08



Спасибо за статью!

Есть, всё таки Люди в этом мире, которые в гугл не отправляют.

Ещё раз спасибо, теперь за Вашу человечность)

Ответить



**nwwind** 04.01.18 в 22:09



Не за что. :)

Если что не понятно — спрашивайте.

Ответить

Написать комментарий

В /      \*

Предпросмотр

Отправить

☐ Markdown

САМОЕ ЧИТАЕМОЕ

- Сутки
- Неделя
- Месяц

Б — Брутальность. Официальный сайт Федерации настольного тенниса Республики Башкортостан (ФНТ РБ)

+42 14,9k 23 70

Зацените: сделал стол

+141 17,7k 217 261

Python для ребёнка: выбор самоучителя

+40 10,9k 242 44

Мессенджеры vs соцсети vs ... — анонс нового проекта

+7 4k 19 37

Минтруд: тестовое задание — это трудовые отношения

+23 11,6k 97 288

claygod	Разделы	Информация	Услуги	Приложения
Профиль	Публикации	Правила	Реклама	<div>Загрузите в App Store</div> <div>доступно Google</div>
Трекер	Хабы	Помощь	Тарифы	
Диалоги	Компании	Документация	Контент	
Настройки	Пользователи	Соглашение	Семинары	
ППА	Песочница	Конфиденциальность		