УДК 62-529 Научная статья

https://doi.org/10.32603/2071-8985-2022-15-7-66-72

Эффективный подход к разработке программного кода для управления электроприводом на примере макета лифтового подъемника

Н. А. Смирнов

Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина), Санкт-Петербург, Россия

smirnov-n1@mail.ru

Аннотация. Рассматривается разработка и использование программы управления электроприводом макета кабины лифтового подъемника. Цель работы состоит в том, чтобы показать, как в достаточно короткие сроки написать программу для использования ее на реальном макете и как изменить ее в случае необходимости при помощи микрокомпьютера и языка программирования Python версии 3, а также ряда компонентов для создания макета – двигателя, драйверов и пр. На данный момент, обладая базовыми знаниями программирования, можно в достаточно короткие сроки создать программу управления для макета или виртуальной модели. В данной статье поэтапно рассказано, как, не используя сложных концепций программирования, создать полноценно работающий макет лифтового подъемника. На каждом этапе объясняется, как работает та или иная функция или модуль. В конечном итоге получается рабочая программа управления и пояснения ее работы.

Ключевые слова: программа управления, электропривод, функциональное программирование, макет, моделирование

Для цитирования: Смирнов Н. А. Эффективный подход к разработке программного кода для управления электроприводом на примере макета лифтового подъемника // Изв. СПбГЭТУ «ЛЭТИ». 2022. Т. 15, № 7. С. 66–72. doi: 10.32603/2071-8985-2022-15-7-66-72.

Original article

An Effective Approach to the Development of a Program Code for Controlling an Electric Drive Using the Example of an Elevator Layout

N. A. Smirnov

Saint Petersburg Electrotechnical University, Saint Petersburg, Russia

smirnov-n1@mail.ru

Abstract. The development and use of the control program for the electric drive of the elevator cabin layout is considered. The purpose of the work is to show how to write a program in a fairly short time for using it on a real layout and the possibility of changing it if necessary. The use of a microcomputer and the Python version 3 programming language, as well as a number of components for creating a layout, such as an engine, drivers, and more. At the moment, having basic programming knowledge, you can create a control program for a layout or virtual model in a fairly short time. This paper describes step by step how, without using complex programming concepts, to create a fully working layout of an elevator hoist. At each stage, it explains how a particular function or module works. The end result is a working control program and explanations of its operation.

Keywords: control program, electric drive, functional programming, layout, modeling

For citation: Smirnov N. A. An Effective Approach to the Development of a Program Code for Controlling an Electric Drive Using the Example of an Elevator Layout // LETI Transactions on Electrical Engineering & Computer Science. 2022. Vol. 15, no. 7. P. 66–72. doi: 10.32603/2071-8985-2022-15-7-66-72.

Введение. Разработка моделей, для анализа поведения реальной конструкции широко используется на данный момент. Возможность в короткие сроки создать модель, которая давала

бы представление о поведении объекта, крайне полезна при изучении любого технического устройства. Цель данной статьи состоит в разработке модели кабины лифтового подъемника с использованием языка программирования Python. Объектом исследования служит программа на этом языке, демонстрирующая легкость написания программ для изучения устройств подобного рода. В статье рассказывается о библиотеке, которая использовалась в данной программе как средство, упрощающее работу с языком; описаны основные принципы, заложенные в программе, и в целом показана мощность данного языка программирования, реализация встроенных в него инструментов и их применение при создании программы для модели.

Для начала стоит сказать, что язык Python имеет очень простой синтаксис, не перегруженный большим объемом специального синтаксиса в сравнении с другими языками [1]. Зачастую в различных языках программирования требуется использовать специальный синтаксис – фигурные скобки, точки с запятой, объявления о том, публичная ли переменная или нет, а также специальные приставки к названиям переменных. Все это может отталкивать или раздражать программиста-разработчика моделей. В данной статье будет рассмотрена программа для создания модели, полезные свойства и преимущества языка, упрощающие работу с ним. Хочется отметить, что Python все еще очень сложен, так как это – полноценный язык программирования, однако он проще в освоении, чем большинство других языков.

Представлена программа для управления макетом кабины лифтового подъемника на языке Python. Показаны все этапы создания программы, начиная от инициализации библиотек и заканчивая рабочими циклами, расписаны ее преимущества перед аналогичной программой на упрощенном языке C. Целью исследования выступает разбор создания макета лифтового подъемника и показ того, что даже человек с низким уровнем знаний в программировании может писать программы для управления физическими макетами и, как следствие, их анализировать и использовать в научной деятельности. Показаны примеры комплектующих для этого макета и их характеристики. Данная модель показывает возможности микрокомпьютеров семейства Orange Pi и их совместимость с языком программирования Python.

Рассмотрим алгоритм программы. Он разбит на блоки, и каждый из них – это функция, которая входит в общий рабочий цикл. Каждая функция представляет собой фрагмент кода, управляющий той или иной частью алгоритма. Алгоритм состоит из четырех частей, каждая из которых изолирована от других и сообщается с ними только посредством передачи логических переменных там, где это требуется. Важно, что замена части программы в одной из функций фундаментально не повлияет на другие функции. Перед функциями располагаются инициализация (она вводит данные в систему) и тело цикла – то, что вызывает каждую функцию по мере надобности. Именно поэтому описание программы стоит начать с этих двух частей и уже затем перейти к логике программы.

Блок-схема работы программы (рис. 1). После запуска двигателя начинается открытие двери. Программа запускает функцию открытия, которая будет выполняться до момента, пока датчик открытия не сработает, затем дверь какое-то время будет открыта под воздействием функции ожидания. После этого начнет работу функция закрытия двери. Ей будет выделено определенное время на закрытие – если его не хватит, то это станет сигналом того, что дверь застряла и нужен реверс. Он запустится, и программа снова начнет открывать дверь, только уже через функцию реверса. Затем цикл вернется к стадии ожидания закрытия двери. Если же функция закрыла дверь за заданное время, то программа завершится.

Электрическая схема. Электрическая схема выполнена для того, чтобы показать, какие элементы используются в схеме – вплоть до модели резистора. В качестве платы для Orange Pi Lite использована база от Raspberry Pi 3, так как они взаимозаменяемы. На блок-схеме (рис. 2) показано, как соединены все элементы и какие части используются в настоящей электрической схеме. С ее помощью можно симулировать процесс, подобрав при перенесении в программу для симуляции по типу Proteus, в которой она изначально была создана, нужное сопротивление резисторов, затем – наиболее подходящие параметры для реальной схемы, чтобы не допустить неполадок на самом макете. Данная схема

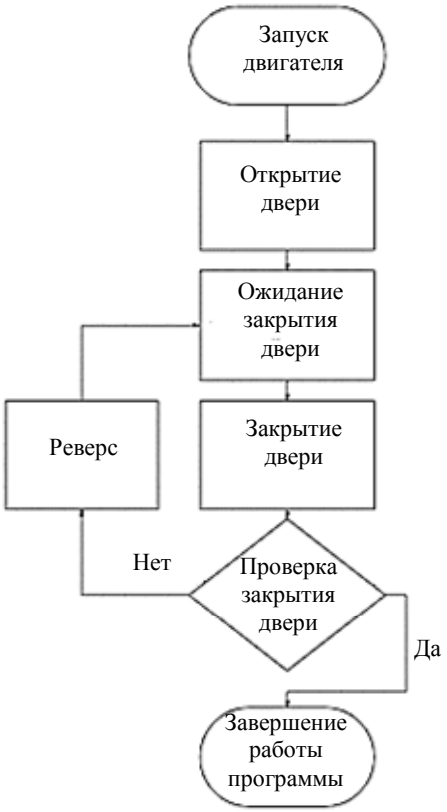

Рис. 1. Блок-схема
Fig 1. Block diagram

компактна и в то же время демонстрирует все устройство макета и его рабочие компоненты. Ограничивающие резисторы *R*1–*R*5 установлены на 220 Ом, это нужно для того, чтобы светодиоды не перегорели. Можно также изменять яркость, меняя резисторы в схеме, однако резисторы менее 100 Ом использовать не рекомендуется, так как диод может перегореть от большого тока. Кнопка и механические датчики также подсоединены к понижающим резисторам для ограничивания тока – на рисунке они показаны в левой части схемы. Два механических датчика работают от нажатия, поэтому на схеме они изображены в виде кнопок. В правой верхней части схемы можно видеть плату с четырьмя входами и выходами – драйвер тока. Какой именно использовать драйвер (в макете это drv8833), не принципиально – он нужен для подключения двигателя и ограничивания тока для него, так как если подключить двигатель напрямую к плате, он может просто сгореть. В качестве двигателя M лучше всего использовать бесколлекторный двигатель постоянного тока. В качестве индикаторов использованы обычные светодиоды красного и зеленого цвета с максимальным током в 20 мА. Выходы VSS и VS относятся к модулю вывода подачи питания моторов драйвера. GND – общий вывод питания драйвера. Обозначение GPIO на плате показывает порты, с помощью которых и осуществляется управление периферией – они принимают либо посылают логический сигнал.

Инициализация. Перед тем как объяснить, как происходит инициализация переменных и портов, следует сказать, что данная программа написана на микрокомпьютере Orange Pi Lite – о семействе микрокомпьютеров Orange Pi как альтернативе семейству Raspberry Pi было сказано ранее. На данном этапе нужно инициализировать

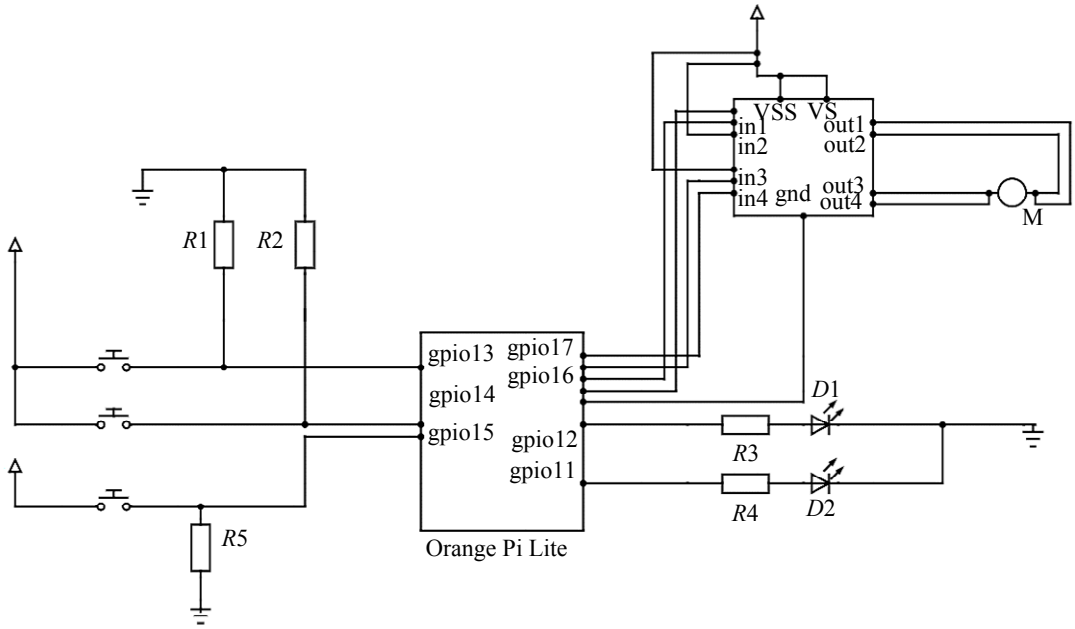

Рис. 2. Электрическая схема
Fig. 2. Wiring diagram

порты платы, чтобы подавать на них напряжения и тем самым управлять подключенными устройствами. Изначально импортируется библиотека OPi.GPIO. GPIO – это и есть название портов на которые будет подаваться напряжение, помимо этого еще импортируется библиотека time, которая пригодится позже. Затем активируются все порты, которые будут нужны в программе, напротив каждого идет комментарий, за исключением первых двух строк, нужных только для того, чтобы настроить библиотеку на работу именно с данной моделью платы. Видно, что потребуются порты для кнопки вызова лифта, зеленого и красного светодиодов, датчиков открытия и закрытия, а также портов, которые будут подавать ток на драйвер для управления электродвигателем макета. В самом низу идут логические переменные и временные константы, благодаря которым осуществляется управление циклом.

```
Import OPi.GPIO as GPIO
from time import time

GPIO.setboard(GPIO.LITE)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(15, GPIO.IN,
pull_up_down=GPIO.PUD_OFF) #кнопка вызова
GPIO.setup(11, GPIO.OUT) #зеленый светодиод
GPIO.setup(12, GPIO.OUT) #красный светодиод
GPIO.setup(13, GPIO.IN,
pull_up_down=GPIO.PUD_OFF) #датчик открытия
GPIO.setup(14, GPIO.IN,
pull_up_down=GPIO.PUD_OFF) #датчик закрытия
GPIO.setup(16, GPIO.OUT) #ток на первый
вход драйвера
GPIO.setup(17, GPIO.OUT) #ток на второй
вход драйвера

Reverse = False
Closing = False
Waiting_time = 5
Reverse-time = 10
Process_gone = True
```

Далее следует основной цикл программы, выполненный внутри конструкции try except. Данная конструкция в языке Python нужна для того, чтобы в случае ошибки выдать второй кусок кода после *except* – в данной ситуации это жизненно важно для платы, так как в случае некорректной работы или сбоя в подаче напряжения может произойти ошибка и плата микрокомпьютера просто сгорит [2]. Поэтому важно, чтобы в случае ошибки код сразу переходил к завершению. Внутри самой этой конструкции спрятан цикл *while*, который будет выполняться вплоть до тех пор, пока логическая переменная процесса будет равна *True* или пока не появится ошибка, которая прервет цикл. Сам же цикл в плане управления функциями представляет собой следующую конструкцию: запускается цикл открытия двери, после него цикл двери, находящейся в открытом состоянии и ожидающей некоторое время, сразу после этого в переменную передается значение логической переменной, говорящей о закрытии двери по истечении срока. Затем запускается цикл закрытия, и если все прошло своевременно, то дверь будет закрыта, а цикл завершен и очищен для нового запуска методом *cleanup*(). В противном случае произойдет реверс, и программа начнется с начала. Стоит сказать, что изначально цикл запускается нажатием на кнопку открытия – кнопку вызова лифта. Цикл очень короткий, и, как будет видно дальше, сильно отличается от аналогичной программы на другом языке. Одно из главных преимуществ заключается в том, что основная логика управления целой программой умещается в несколько строк, в этом цикле очень легко ориентироваться и в случае, если нужно развить программу, можно добавить в основной цикл функции, при этом не меняя его сути. Для начинающего разработчика или исследователя, которому требуется быстро написать прототип, это – несомненный плюс.

```
try:
    if GPIO.input(15):
     while process_gone == True:
            process_open_door()
            close_var = open_door(closing)
            if close_var == True:
                close()
except KeyboardInterrupt:
    GPIO.cleanup()
```

Теперь рассмотрим функции, которые несут в себе логику активации портов и управления аппаратной частью системы. Начинает этот список функция открытия двери – одна из самых коротких, тут мы снова видим цикл *while*. В данной функции проверяется, сработает ли датчик открытия двери на тринадцатом порте, и пока он не сработал, программа выполняет следующие 3 действия: одно из них включает красный диод, который сигнализирует об открытии двери; второе подает ток на первый вход двигателя; третье не подает ток на второй вход двига-

теля. Двигатель за счет этого начинает вращаться и открывает дверь макета, пока она, нажав на датчик открытия, не прервет выполнение цикла. Вся эта логика умещается в 4 строки и, как следствие, показывает простоту управления двигателем и другими устройствами. Библиотека GPIO позволяет за одну или две короткие строки программного кода выполнить то, что для других языков заняло бы значительно больший объем или представляло бы большую сложность.

```
def process_open_door():
    while GPIO.input(13) != 1:
     GPIO.output(12, 1) # красный диод включен
     GPIO.output(16, 1) # ток идет на первый вход двигателя
     GPIO.output(17, 0) # ток не идет на второй вход драйвера
```

Следующая функция – это функция ожидания двери, или же функция ожидания. В нее в качестве аргумента передается логическая переменная, которая позволяет прервать цикл внутри функции для того, чтобы выйти из функции в главный цикл. Модуль *time*, импортированный в самом начале программы, позволяет использовать метод *time*, который позволяет отсчитывать время в реальности и сравнивать с временной переменной. Реальное время отсчитывается в секундах, и переменная, которая в это время записывается, создается в самом начале функции. Далее цикл *while* продолжается до тех пор, пока переменная, отвечающая за начало функции закрытия, не активна. Внутри цикла с помощью активации портов выключается красный диод, включается зеленый, ток на оба двигателя не подается и двигатель останавливается. В конце каждого такого цикла функция проверяет, не прошло ли время закрытия двери: если временная переменная меньше, чем текущее время, то сразу после этого переменная, отвечающая за закрытие двери, становится положительной, после чего цикл разрывается и переменная передается в цикл.

```
def open_door(closing):
    open_time = time.time() # присвоение времени для сравнения с временем ожидания
    while closing == False:
     GPIO.output(12, 0) # выключение красного диода
     GPIO.output(11, 1) # включение зеленого диода
     GPIO.output(16, 0) # выключение двигателя
     GPIO.output(17, 0)
     if open_time > Waiting_time:
         closing = True
         return closing
```

Функция закрытия двери – самая большая по размеру и, в отличие от функции открытия двери, контролирует не только закрытие, но и возможный реверс. Опять используется метод *time* из модуля *time*, который позволяет отсчитывать время, для того чтобы контролировать момент, когда нужно будет включить функцию реверса. О реверсе будет рассказано позже, он нужен в случае, если дверь застряла или затормозилась и нужно снова ее открыть. В цикле *while* условие проверяется, пока не будет активирован датчик закрытия – до этого момента будет включен красный диод, выключен зеленый и ток на двигатель будет подаваться на один из входов, чтобы закрыть дверь. В конце каждого цикла проверяется, не превышено ли время закрытия. Если оно не превышено, переменная, отвечающая за весь цикл, меняет состояние и цикл заканчивается до следующего нажатия кнопки. В противном случае запускается функция реверса.

```
def close_door(process_gone):
    close_time = time.time() # время закрытия двери
    while GPIO.input(14) != 1 and close_time <= Reverse:
     GPIO.output(12, 1) # включение красного диода
     GPIO.output(11, 0) # выключение зеленого диода
     GPIO.output(17, 1) # подача тока на один вход двигателя
     GPIO.output(16, 0)
     if close_time > Reverse_time:
         do_reverse()
    process_gone = False
    return process_gone
```

Функция реверса, как было сказано ранее, нужна для того, чтобы если произойдет столкновение двери с препятствием, дверь снова открылась. В реальных лифтах для этого используется показание об изменении тока якоря, однако микрокомпьютер Orange Pi Lite ток якоря считывать не может, поэтому используется обычный таймер. В данном случае функция была запущена после

того, как время ожидания закрытия двери превысило переменную реверса по времени. После этого, до тех пор, пока датчик открытия двери снова не сработает, дверь будет вращать двигатель в обратном направлении, чтобы можно было устранить препятствие для нее и, если человека предположительно зажало дверью, он бы смог выбраться. После того как функция сработала, цикл начнется по новой с момента, когда дверь должна ждать закрытия.

```
def do_reverse():
    while GPIO.input(13) != 1:
     GPIO.output(12, 1) # все еще включен красный диод
     GPIO.output(17, 0) # двигатель начинает крутиться в другую сторону
     GPIO.output(16, 1)
```

Подводя итог, можно сказать, что данная программа содержит всего несколько логических конструкций, очень простых в освоении, и для человека, который только пришел в программирование, она подходит с самых ранних этапов знакомства. Основной объем этой программы занимает переключение портов для подачи на них напряжения с целью управления подключенными устройствами. Всего в программе содержится несколько циклов, логических сравнений и логических переменных. Данный язык идеально подходит для написания моделей подобного рода, а также позволяет людям, не использовавшим ранее языки программирования, начать этим заниматься [3]. Если же потребуется внести изменения, добавить дополнительную функцию, то ее нужно просто вписать в основный цикл, и это никак не изменит программу в корне. Модульный характер программы формирует ее архитектуру, которая позволяет не бояться внесения правок и изменений и делает ее невероятно гибкой.

Заключение. В данной статье было показано, что использование языка Python доступно для людей, которые не занимались программированием ранее и хотят создать свою модель. Разбор показал, что синтаксис языка не перегружен в сравнении с более распространенными аналогами. Можно использовать его как для полноценной разработки, так и для прототипирования. Разбор показывает, что для выполнения элементарных операций не требуются громоздкие конструкции на более ООП-ориентированных языках, которые делают эти операции менее читаемыми для начинающего.

Список литературы

1. Лентин Дж. Изучение робототехники с помощью Python: проектирование, моделирование, программирование и прототипирование интерактивного автономного мобильного робота с нуля с помощью Python, ROS, Open-CV. М.: ДМК Пресс, 2019. 249 с.

2. Мартелли А., Рейвенскрофт А., Холден С. Python: справочник: полное описание языка: [рассмотрены версии Python 2.7, 3.5 и новинки версии 3.6] / пер. с англ. А. Г. Гузикевича. М.: Диалектика, 2019. 3-е изд. 892 с.

3. Лутц М. Python: карманный справочник Python в вашем кармане / пер. с англ. М.: Вильямс, 2017. 5-е изд. 318 с.

Информация об авторе

Смирнов Никита Артемович – аспирант кафедры робототехники и автоматизации производственных систем СПбГЭТУ «ЛЭТИ».

E-mail: smirnov-n1@mail.ru

References

1. Lentin Dzh. Izuchenie robototekhniki s pomo-shch'yu Python: proektirovanie, modelirovanie, pro-grammirovanie i prototipirovanie interaktivno-go avtonomnogo mobil'nogo robota s nulya s pomoshch'yu Python, ROS, Open-CV. M.: DMK Press, 2019. 249 s. (In Russ.).

2. Martelli A., Rejvenskroft A., Holden S. Python : spravochnik: polnoe opisanie yazyka: [rassmotreny versii Python 2.7, 3.5 i novinki versii 3.6] / per. s angl. A. G. Guzikevicha. M.: Dialektika, 2019. 3-e izd. 892 s. (In Russ.).

3. Lutc M. Python: karmannyj spravochnik Python v vashem karmane / per. s angl. M.: Vil'yams, 2017. 5-e izd. 318 s. (In Russ.).

Information about the author

Nikita A. Smirnov – post-graduate student of the Department of Robotics and Automation of Production Systems, Saint Petersburg Electrotechnical University.
E-mail: smirnov-n1@mail.ru

Статья поступила в редакцию 26.05.2022; принята к публикации после рецензирования 02.06.2022; опубликована онлайн 13.09.2022.

Submitted 17.05.2022; accepted 23.05.2022; published online 13.09.2022.