Nagranie wideo z gry na BeetBoxie można obejrzeć pod adresem *https://goo.gl/7j8kix*.

BeetBox – perkusja z Raspberry PI i buraków

Raspberry PI to niedrogi i nieduży komputer, za pomocą którego można zrealizować różne przydatne projekty. Dostępność platformy i rozgłos jej towarzyszący sprawiły, że zainteresowali się nią także artyści. Mały komputer pozwala w łatwy i niedrogi sposób tworzyć „dzieła" interaktywne i daje zupełnie nowe możliwości ich twórcom – a szczególnie tym, którzy projektują eksponaty luźno rozumianej sztuki nowoczesnej. Jedną z takich osób jest Scott Garner z USA. Zaprojektował on niewielką, elektroniczną perkusję, której najbardziej zwracającymi na siebie uwagę elementami są… buraki.

Buraki nie są jednak kluczowe dla tego projektu. Wybór na nie padł najpewniej ze względu na ich angielską nazwę, która pozwala uzyskać pewną grę słów. Burak po angielsku to „beet", co wymawia się bardzo podobnie do słowa „beat", czyli muzycznego bitu. I ponieważ beatbox to rytmiczne wydawanie dźwięków przypominających perkusyjne, ale za pomocą ust, gardła i przepony, autor wybrał na swój projekt nazwę BeetBox, tworząc pudełko z burakami, których dotknięcie pozwala na generowanie dźwięków podobnych jak z perkusji. Projekt nie jest trudny i można go łatwo odtworzyć – przygotowując, na przykład, jako żart na zbliżający się Prima Aprilis.

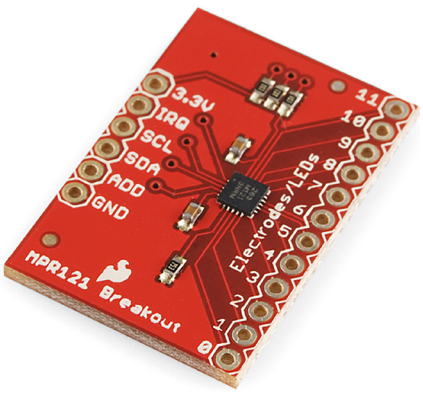
Wygląd

BeetBox został zaprojektowany tak, by w żadnym stopniu nie sugerował z góry, że jest to urządzenie elektroniczne. Obudowa składa się z drewna, a poza nią widoczne są jedynie buraki. Dzięki temu całość robi zupełnie niewinne wrażenie, trochę jakby była miała to być jakaś ekskluzywna skrzynka na warzywa. Zresztą konstrukcja obudowy nie jest tak ważna z punktu widzenia działania projektu – istotne jest tylko, by poszczególne buraki były od siebie odseparowane elektrycznie, co oznacza, że nie mogą się stykać, a stelaż, na którym się znajdują nie może

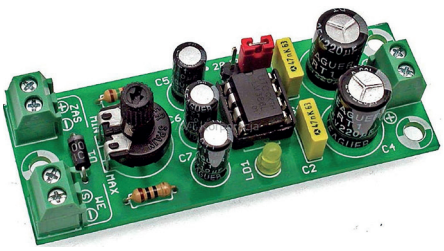
Potrzebne elementy:
- Raspberry PI.
- Moduł z czujnikiem MPR121.
- Moduł wzmacniacza audio.
- Głośnik.
- Przewody do połączenia ze sobą komponentów.
- 6 szpilek.
- Karta pamięci SD.
- Zasilacz USB 5 V.
- Deski do wykonania obudowy.
- 6 ładnych buraków.

przewodzić prądu. Z tego względu drewno wydaje się być dobrym wyborem szczególnie, że jest łatwe w obróbce, szerokodostępne oraz naturalne w naszym otoczeniu, dzięki czemu nie budzi podejrzeń, że może zawierać urządzenie elektroniczne.

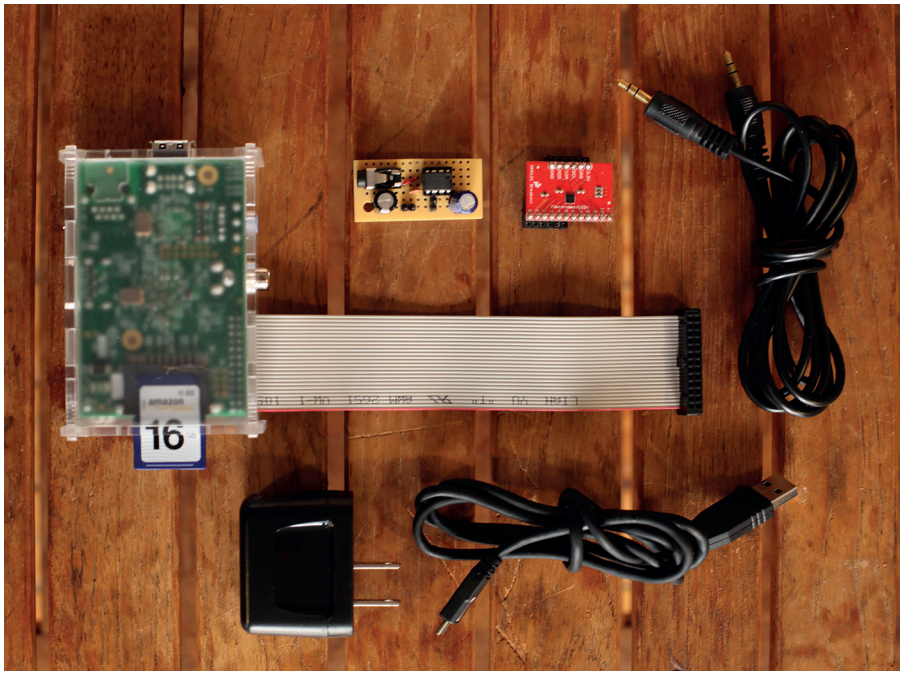
Autor na potrzeby obudowy wykorzystał deski o grubości ok. 12 mm, które dociął do potrzeb i wykończył za pomocą ręcznych i elektrycznych narzędzi. Konieczne było wykonanie nacięć, otworów i nawierceń. Całość można też pomalować i polakierować, by wyglądała estetyczniej i by zwiększyć trwałość powierzchni. Powycinane deski autor skleił klejem do drewna i zbił gwoździami, następnie uszczelniając wszystko pianką poliuretanową.



Fotografia 1. Płytka z układem MPR121



Fotografia 2. Moduł zasilacza z układem LM386



Fotografia 3. Komponenty elektroniczne użyte w projekcie

Budowa elektryczna

Z punktu widzenia elektroniki, projekt jest bardzo prosty. Serce „urządzenia" stanowi Raspberry PI – autor użył Modelu B+ z 512 MB pamięci RAM. Kluczowy jest też pojemnościowy sensor dotyku – tu autor skorzystał z gotowego modułu firmy Sparkfun, która zainstalowała układ Freescale MPR121 na PCB i wyprowadziła połączenia. To bardzo wygodne rozwiązanie (pozwalające uniknąć lutowania bardzo małych padów) szczególnie, że sam układ został wycofany z produkcji i wielu dostawców już nie ma go w swojej ofercie. Natomiast alternatywą dla kosztującej ok. 40 złotych płytki z MPR121 może być zamiennik produkowany przez inne firmy. Potrzebny jest też głośnik, a ponieważ poziom sygnału audio z Raspberry PI jest bardzo niski, przydatny będzie też wzmacniacz – np. regulowany AVT794, zbudowany na układzie LM386. Autor samodzielnie zbudował wzmacniacz właśnie w oparciu o LM386, ale nie będziemy opisywać tego fragmentu projektu, gdyż chcemy skoncentrować się na oprogramowaniu i sprzęcie w postaci modułów.

Oprócz dotąd wymienionych konieczne będą też: zasilacz USB, karta pamięci oraz przewody do połączenia Raspberry PI z modułem Sparkfuna, wyjścia audio z wzmacniaczem, wzmacniacza z głośnikiem a także – co ważne – modułu dotykowego z burakami. W tym ostatnim przypadku chodzi o dowolne przewody, których końcówki będzie można wbić w buraki. Można przykładowo użyć szpilek lub pinów lutowniczych, przymocowanych do przewodów. Autor zastosował szpilki, których połączenie z przewodami dodatkowo wzmocnił taśmą izolacyjną.

Zestaw komponentów elektrycznych został pokazany na **fotografii 3**, a schemat na **rysunku 4**.

Moduły programowe

Działanie urządzenia bazuje na zmianie pojemności buraka, po jego dotknięciu przez człowieka. Zdarzenie to jest wykrywane przez układ MPR121 i przekazywane interfejsem I²C do Raspberry PI. Komputer, w zależności od tego który burak został dotknięty, a więc na której elektrodzie zmieniła się wykrywana pojemność, odtwarza plik dźwiękowy, będący próbką jednego z instrumentów perkusji. Dźwięk po wzmocnieniu przez układ LM386 trafia do głośnika i wydobywa się z obudowy przez nawiercone otwory.

W związku z powyższym, do poprawnego działania urządzenia potrzebne są następujące elementy programowe:
- System operacyjny.
- Interfejs do komunikacji z MPR121 przez I²C.
- Narzędzie do odtwarzania plików dźwiękowych.
- Skrypt kontrolujący odczyty z MPR121 i decydujący, który plik dźwiękowy odtworzyć.
- Pliki dźwiękowe do odtwarzania.

Autor napisał skrypty w Pythonie i skorzystał z biblioteki **pygame**, która pozwala łatwo odtwarzać multimedia.

Za system operacyjny posłużył standardowy Raspbian, pobrany ze strony internetowej Raspberry PI. Następnie trzeba skonfigurować komputer, by korzystał z interfejsu I²C. Interfejs ten pozwala na podłączanie wielu komponentów za pomocą jednego zestawu przewodów i odwoływanie się do nich po adresach, w związku z czym wymaga konfiguracji. Co więcej, jest on wyprowadzony na pinach GPIO, które mogą być użyte w innym celu.

Konfiguracja

Aby móc swobodnie posługiwać się I²C z poziomu języka Python warto zainstalować dwa narzędzia: bibliotekę **python-smbus** i **i2c–tools**. Odbywa się to za pomocą poleceń:

sudo apt-get install python-smbus

sudo apt-get install i2c-tools

Należy też ustawić obsługę I²C jako domyślnie włączoną – najwygodniej jest to zrobić za pomocą narzędzia **raspi-config**, wpisując:

sudo raspi-config

a następnie wybierając pozycję Advanced Options i wskazując opcję I²C (**rysunek 5**). Na ekranie pojawią się jeszcze dwa okienka, w których trakcie konfigurator zapyta o potwierdzenie oraz o wprowadzenie tego ustawienia jako domyślne. W obu przypadkach należy zaznaczyć „YES”.

Po restarcie urządzenia poprawność konfiguracji można sprawdzić poprzez podgląd treści pliku **/etc/modules**, w którym na końcu powinny znaleźć się linijki:

i2c-bcm2708

i2c-dev

Wypada też sprawdzić, czy w systemie istnieje plik **/etc/modprobe.c/raspi-blacklist.conf**, a jeśli tak, to czy zawiera on linijki:

blacklist spi-bcm2708

blacklist i2c-bcm2708

Jeśli tak, to należy go edytować i wykomentować je, wstawiając znak „#” na ich początkach. W nowszych wersjach Raspbiana konfigurator powinien też zmienić treść pliku **/boot/config.txt**. Powinny się w nim znaleźć linijki:

dtparam=i2c1=on

dtparam=i2c_arm=on

Jeśli w trakcie sprawdzania konieczne było wprowadzenie jakichkolwiek zmian, system wymaga ponownego restartu.

Po uruchomieniu poprawnie skonfigurowanego Raspberry PI i podłączeniu do niego układu MPR121 można sprawdzić, czy całość się ze sobą komunikuje. W tym celu przydatne będzie polecenie **i2cdetect**. W przypadku Raspberry PI z 512 MB pamięci RAM należy je wywołać:

sudo i2cdetect –y 1

W starszym modelu z 256 MB pamięci RAM na potrzeby I²C wykorzystano port 0 a nie port 1, w związku z czym detekcję należałoby wykonać poleceniem:

sudo i2cdetect –y 0

Polecenia te przeszukują urządzenia podmontowane w **/dev/i2c-1** lub **/dev/i2c-0** w poszukiwaniu podłączonych komponentów. Układ MPR121 jest skonfigurowany do pracy pod adresami 0x5A, 0x5B, 0x5C lub 0x5D, w zależności od sposobu podłączenia pinu ADDR. W przypadku płytki firmy Sparkfun domyślny adres to 0x5A, ale można go zmienić zmieniając ustawienie zworek. Pin ADDR można zewrzeć z pinami VSS, VDD, SDA albo

Listing 1. Fragment biblioteki do obsługi pliku MPR121. Pominięto w nim definicje stałych

```
import smbus
bus = smbus.SMBus(1)
# Pominięte definicje rejestrów MPR121
# Pominięte stałe globalne

# Funkcje:
def readData(address):
    MSB = bus.read_byte_data(address, 0x00)
    LSB = bus.read_byte_data(address, 0x01)
    #touchData = (MSB << 8) | LSB
    touchData = MSB;
    return touchData;

def setup(address):
    bus.write_byte_data(address, ELE_CFG, 0x00)
    # Filtrowanie dla wskazań powyżej pojemności bazowej
    bus.write_byte_data(address, MHD_R, 0x01)
    bus.write_byte_data(address, NHD_R, 0x01)
    bus.write_byte_data(address, NCL_R, 0x00)
    bus.write_byte_data(address, FDL_R, 0x00)
    # Filtrowanie dla wskazań poniżej pojemności bazowej
    bus.write_byte_data(address, MHD_F, 0x01)
    bus.write_byte_data(address, NHD_F, 0x01)
    bus.write_byte_data(address, NCL_F, 0xFF)
    bus.write_byte_data(address, FDL_F, 0x02)
    #Ustawianie granic wykrywania rozpoczęcia i zakończenia dotyku
    #dla wszystkich 12 elektrod
    bus.write_byte_data(address, ELE0_T, TOU_THRESH)
    bus.write_byte_data(address, ELE0_R, REL_THRESH)
    bus.write_byte_data(address, ELE1_T, TOU_THRESH)
    bus.write_byte_data(address, ELE1_R, REL_THRESH)
    bus.write_byte_data(address, ELE2_T, TOU_THRESH)
    bus.write_byte_data(address, ELE2_R, REL_THRESH)
    bus.write_byte_data(address, ELE3_T, TOU_THRESH)
    bus.write_byte_data(address, ELE3_R, REL_THRESH)
    bus.write_byte_data(address, ELE4_T, TOU_THRESH)
    bus.write_byte_data(address, ELE4_R, REL_THRESH)
    bus.write_byte_data(address, ELE5_T, TOU_THRESH)
    bus.write_byte_data(address, ELE5_R, REL_THRESH)
    bus.write_byte_data(address, ELE6_T, TOU_THRESH)
    bus.write_byte_data(address, ELE6_R, REL_THRESH)
    bus.write_byte_data(address, ELE7_T, TOU_THRESH)
    bus.write_byte_data(address, ELE7_R, REL_THRESH)
    bus.write_byte_data(address, ELE8_T, TOU_THRESH)
    bus.write_byte_data(address, ELE8_R, REL_THRESH)
    bus.write_byte_data(address, ELE9_T, TOU_THRESH)
    bus.write_byte_data(address, ELE9_R, REL_THRESH)
    bus.write_byte_data(address, ELE10_T, TOU_THRESH)
    bus.write_byte_data(address, ELE10_R, REL_THRESH)
    bus.write_byte_data(address, ELE11_T, TOU_THRESH)
    bus.write_byte_data(address, ELE11_R, REL_THRESH)
    # Ustawianie filtrów i włączenie wszystkich elektrod
    bus.write_byte_data(address, FIL_CFG, 0x04)
    bus.write_byte_data(address, ELE_CFG, 0x0C)
```

Listing 2. Główna pętla programu, odpowiadająca za analizę danych z MPR121 i wydawanie poleceń związanych z odtwarzaniem dźwięku

```
touches = [0,0,0,0,0,0];
while True:
    if (GPIO.input(7)): # Wejście przerwania ma stan wysoki
        pass
    else: # Wejście przerwania ma stan niski
        touchData = mpr121.readData(0x5a) #odczyt z sensora
        for i in range(6):
            if (touchData & (1<<i)):
                if (touches[i] == 0):
                    print( 'Burak ' + str(i) + ' został właśnie dotknięty')
                    if (i == 0):
                        kick.play()
                    elif (i == 1):
                        snare.play()
                    elif (i == 2):
                        openhh.play()
                    elif (i == 3):
                        closedhh.play()
                    elif (i == 4):
                        clap.play()
                    elif (i == 5):
                        cymbal.play()
                    touches[i] = 1;
            else:
                if (touches[i] == 1):
                    print( 'Burak ' + str(i) + ' przestał być dotykany')
                    touches[i] = 0;
```

SCL, przy czym nie można zwierać z więcej niż jednym na raz.

Obsługa MPR121

Układ MPR121 pozwala na podłączenie nawet 12 elektrod, ale w projekcie wykorzystano tylko 6 z nich. Co więcej, 8 z 12 elektrod może posłużyć jako uniwersalne wejścia i wyjścia – np. do zasilania diod LED, ale tego również nie wykorzystano. MPR121 ma natomiast bogate funkcje w zakresie wykrywania dotyku, a nawet może posłużyć do wykrywania zbliżającego się obiektu, jeśli wszystkie elektrody znajdują się blisko siebie. Informacje o dotyku są przekazywane w 12 kanałach, a w wypadku uruchomienia wykrywania zbliżania się, udostępniany jest symulowany, 13 kanał.

Co ważne, MPR121 ma wbudowane zaawansowane cyfrowe mechanizmy filtrowania odbieranych sygnałów, dzięki czemu praca z nim jest bardzo łatwa. Dotyk wykrywany jest poprzez monitorowanie pojemności na elektrodach – na każdej niezależnie. Układ wykrywa pojemność od 10 pF do 2000 pF z dokładnością do 0,01 pF. Dzieje się to poprzez zmienianie ilości ładunku i czasu, przez który jest wprowadzany na poszczególne elektrody. Następnie układ mierzy napięcie na pinach, do których podłączone są elektrody – jest ono odwrotnie proporcjonalne do pojemności. Całość odbywa się w cyklach. Na końcu każdego z cykli napięcie to jest próbkowane przez wbudowany 10-bitowy przetwornik analogowo-cyfrowy i przetwarzane.

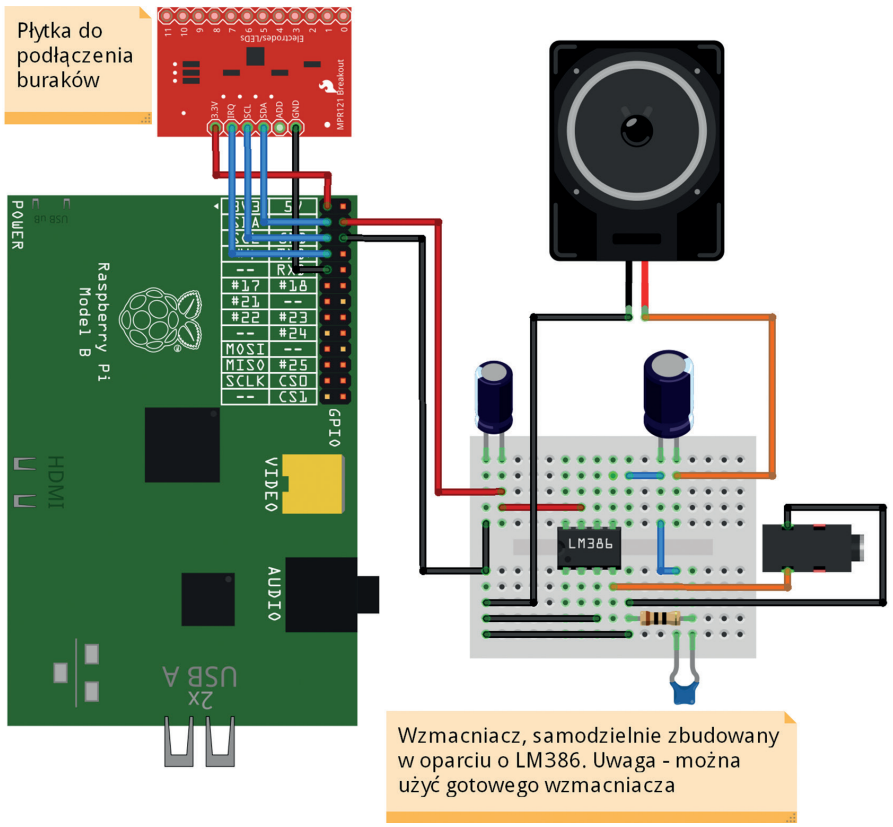
Układ informuje zarówno o dotknięciu podłączonego obiektu, jak i o zakończeniu dotyku. Wystąpienie tych zdarzeń jest determinowane w oparciu o porównanie chwilowej pojemności z bazową. Ta natomiast jest stale śledzona przez MPR121, tak by uwzględnić fluktuacje tła. Różnica pomiędzy aktualną wartością bazową, a wartością chwilową, która ma decydować o wykryciu dotknięcia może być zdefiniowana przez użytkownika – niezależnie dla każdej elektrody. Co ważne, możliwe jest też zdefiniowanie czasu ślepego, by uniknąć wykrywania fałszywych zdarzeń, wywołanych np. przez drgania, zaraz po faktycznym dotyku.

Obsługa układu MPR121 sprowadza się do jego konfiguracji, a następnie cyklicznego odczytywania danych. W tym celu użyteczne są dwie funkcje biblioteki, które przedstawiono na **listingu 1**. Najpierw należy wywołać funkcję **setup()**, podając jej skonfigurowany adres MPR121. W niniejszym przykładzie włączane są wszystkie elektrody, mimo że faktycznie używanych jest tylko 6. Funkcja **readData()** zwraca wczytane dane o wykrytym dotyku.

Dane te są następnie używane w pętli w programie głównym, którą pokazano na **listingu 2**. Dotychczasowy stan elektrod

Listing 3. Fragment kodu głównego skryptu, służący do inicjalizacji i konfiguracji biblioteki pygame na potrzeby odtwarzania 6 różnych dźwięków buraków

```
import pygame
pygame.mixer.pre_init(44100, -16, 12, 512)
pygame.init()
kick = pygame.mixer.Sound('samples/kick.wav')
kick.set_volume(.65);
snare = pygame.mixer.Sound('samples/snare.wav')
snare.set_volume(.65);
openhh = pygame.mixer.Sound('samples/open.wav')
openhh.set_volume(.65);
closedhh = pygame.mixer.Sound('samples/closed.wav')
closedhh.set_volume(.65);
clap = pygame.mixer.Sound('samples/clap.wav')
clap.set_volume(.65);
cymbal = pygame.mixer.Sound('samples/cymbal.wav')
cymbal.set_volume(.65);
```



Rysunek 4. Schemat wykonanych połączeń, z uwzględnieniem modułu audio

(a więc tego, czy buraki były dotknięte) jest zapisywany w tablicy **touches**. Działająca w nieskończoność pętla powoduje wczytanie informacji z czujnika do zmiennej **touchData** (polecenie **touchData = mpr121.readData(0x5a)**), po czym dla pierwszych 6 elektrod dokonywane jest sprawdzenie, czy wykrywają one aktualnie dotyk, czy nie. Jeśli tak, program odczytuje dotychczasowy stan buraka i gdy wynika z niego, że burak jeszcze przed chwilą nie był dotykany, włącza funkcję odtwarzającą przypisany danemu warzywu dźwięk oraz informuje o dotknięciu wpisując informację w konsoli (na ekranie, gdyby takowy był podłączony). Następnie aktualizuje stan danego buraka w tablicy **touches**.

Jeśli z świeżo wczytanych danych wynika natomiast, że dany burak nie jest aktualnie dotykany, program także sprawdza dotychczasowy stan i informuje w konsoli o ewentualnym zaprzestaniu dotknięcia, po czym aktualizuje tablicę **touches**.

Obsługa dźwięku

Pozostaje jeszcze kwestia obsługi dźwięku. W tym celu potrzebna jest biblioteka **pygame**. Jeśli nie jest ona domyślnie zainstalowana w pobranej wersji systemu operacyjnego, trzeba ją ręcznie doinstalować, ale standardowo jest obecnie w Raspbianie.

Obsługa **pygame** na potrzeby tego projektu jest bardzo prosta. Wystarczy zaimportować bibliotekę, skonfigurować parametry miksera dźwięku, zainicjalizować bibliotekę, stworzyć obiekty dźwiękowe klasy **pygame.mixer.Sound**, które będą wskazywać na odpowiedni plik z próbką dźwięku do odtworzenia oraz zawierać ustawienie odnośnie pożądanej głośności. Polecenia te widać na **listingu 3**. Samo odtwarzanie dźwięku polega na wywołaniu funkcji **pygame.mixer.Sound.play()**, dla danego, stworzonego wcześniej obiektu dźwiękowego.

Warto zauważyć, że pliki audio są zapisane w formacie **.wav**, a ustawiona głośność jest identyczna i stała dla każdego z nich.

Podsumowanie i ocena projektu

Cały kod programu, wraz z plikami audio, znajduje się pod adresem: *https://github.com/scottgarner/BeetBox*. Program uruchamia się poleceniem:

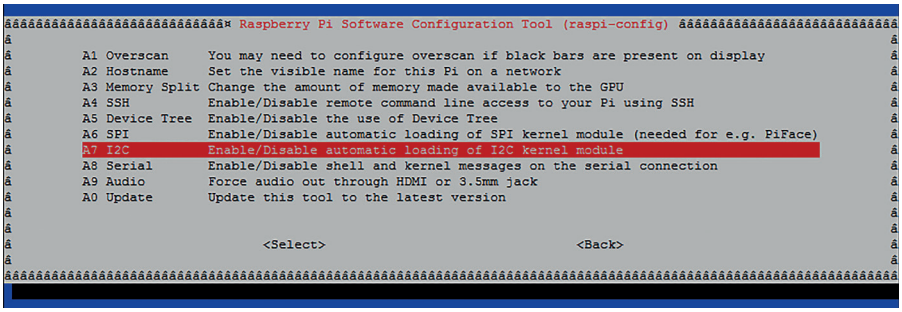
sudo python beetbox.py

Elementy potrzebne do zmontowania BeetBoxa pokazano na **fotografii 6**, a zbliżenie na gotowy projekt na **fotografii 6**.

Autorowi należy pogratulować pomysłu – zarówno idea, zgrabnie dobrana nazwa i wykonanie są bardzo udane. Trudno nam oceniać walory dzieła z punktu widzenia sztuki, ale wydaje się, że jeśli ktoś opracuje coś co wygląda intrygująco, co wcześniej nie powstało, albo przynajmniej nie zostało spopularyzowane, to można to nazwać sztuką i z tego punktu widzenia BeetBox również wydaje się udany. Natomiast z punktu widzenia inżynierskiego można wyobrazić sobie wiele modyfikacji.

Po pierwsze, projekt jest zasilany z użyciem zasilacza podłączanego do sieci elektrycznej. Oznacza to, że z pudełka będzie wystawał kabel, którego na zdjęciach gotowego urządzenia nie widać. Na pewno psuje to wrażenia użytkownika, od razu niszcząc wizję prostej skrzynki na warzywa. Zamiast tego można byłoby zastosować akumulator z przetwornicami tak, by całość można było naładować i odłączyć od zasilania. Prawie nic nie stoi też na przeszkodzie, by użyć pakietu wymiennych baterii, dzięki czemu nie trzeba byłoby zostawiać miejsca na wtyczkę. Wystarczyłoby jedną ze ścianek zrobić wysuwaną, na potrzeby wymiany pakietu zasilającego.

Niestety, przy tak stworzonym programie, akumulator czy baterie szybko zostałyby wyczerpane. Wynika to przede wszystkim z braku jakiegokolwiek sterowania włączeniem i wyłączeniem urządzenia oraz z tego, że autor nie zastosował żadnych trybów uśpienia. Problematyczne jest też zapotrzebowanie na prąd głośnika, ale zależy ono od parametrów całej sekcji audio. Dobrym pomysłem wydaje się po prostu umieszczenie włącznika wewnątrz otwieranej obudowy.



Rysunek 5. Zrzut ekranu konfiguracyjnego Raspberry PI

Sam program również mocno obciąża procesor, gdyż działa w pętli bez jakiegokolwiek odpoczynku. Dotknięcie buraków monitorowane jest nieprzerwanie, a dane pobierane na tyle często, na ile pozwala magistrala I²C. Aby zmniejszyć pobór prądu można byłoby wprowadzić tryb uśpienia, w którym po dłuższym czasie nieużywania (brak zmiany stanu buraków) procesor rzadziej odpytywałby układ MPR121. Np. po 30 sekundach bez interakcji z użytkownikiem, można byłoby wprowadzić 1-sekundowe opóźnienie przed każdorazowym odpytaniem sensora dotykowego, znacząco zmniejszając w ten sposób obciążenie procesora. W momencie wykrycia pierwszego dotyku po przerwie, urządzenie wracałoby do normalnego trybu. Alternatywnie można wykorzystać jedną z nieużywanych elektrod do stworzenia niepozornego przycisku dotykowego, służącego jako włącznik i wyłącznik. Celem zaoszczędzenia energii można również wyłączyć nieużywane elementy – hub USB i HDMI, dostępne w zależności od wersji Raspberry PI.

Projekt można też rozwinąć, korzystając z faktu, że MPR121 ma aż 12 elektrod. Co więcej, układ ten pozwala na konfigurację 1 z 4 adresów dla magistrali I²C, a więc łącznie do jednego Raspberry PI, bez problemu można podłączyć właśnie 4 takie podzespoły, co daje razem 48 wykrywanych punktu dotyku. To umożliwia już stworzenie całkiem pokaźnej klawiatury muzycznej – czy to z marchewek i pietruszki, czy z ogórków lub pomidorów – zależy tylko od inwencji twórczej wykonawcy.

Pewną trudność może stanowić natomiast fakt, że MPR121 został już oficjalnie wycofany z produkcji. W ofercie NXP, które niedawno kupiło firmę Freescale jest podobny, ale mniej rozbudowany układ: MPR03x. Pozwala on na podłączenie jedynie trzech elektrod, ale dostępny jest w dwóch wersjach (MPR031 i MPR032), różniących się wpisanym na stałe adresem w magistrali I²C (odpowiednio 0x4A i 0x4B). Oznacza to, że używając tylko nowych komponentów i bazując pod tym względem na ofercie NXP można stworzyć maksymalnie 6-burakowego BeetBoxa. Z drugiej strony, używając MPR03x w połączeniu z czterema MPR121, dzięki temu, że wszystkie one mają odmienne adresy, da się w jednym systemie z jedną magistralą I²C stworzyć nawet 54-klawiszowego BeetBoxa. Wystarczy to również do budowy 4-oktawowego keyboardu.

W wersji z 6 burakami można też wykorzystać pozostałe 6 elektrod i podpiąć je do tych samych warzyw, ale ustawiając na nich zupełnie inne progi wykrywania dotyku, dzięki czemu – jeśli dobrze poeksperymentować, może udałoby się uzależniać wydawany dźwięk od tego, jak buraki zostały dotknięte.

Praktycznie nic nie stoi na przeszkodzie, by Raspberry PI podłączyć do sieci bezprzewodowej, by za pomocą buraków sterować innymi urządzeniami elektronicznymi. A korzystając z przykładu implementacji klawiatury Bluetooth HID, opublikowanego również w tym numerze EP, można takiego BeetBoxa np. bardzo łatwo podłączyć do telefonu komórkowego.

Dla maksymalizacji efektu zaskoczenia, buraki można by było umieścić w małych doniczkach, a te włożyć do w pełni obudowanej skrzynki BeetBoxa, po czym całość przysypać suchą ziemią do kwiatów, by zasłonić krawędzie pojedynczych doniczek. Trzeba byłoby się tylko upewnić, że obwód pomiędzy poszczególnymi burakami nie jest zamknięty poprzez otaczającą ich ziemię. Być może udałoby się nawet stworzyć prawdziwe, rosnące w ten sposób warzywa (jedna wbita w nie szpilka nie powinna bardzo utrudniać wzrostu), które pomimo podlewania zachowywałyby swoją zdolność do detekcji dotyku, a w konsekwencji do korzystania z prawdziwie żywego BeetBoxa.

Marcin Karbowniczek, EP

Fotografia 6. Gotowy projekt – warto zwrócić uwagę na sposób wykonania obudowy