



Kontroler MIDI

Systemy Wbudowane 2023

Dominik Grzesik, Marcin Mikuła

Opis projektu

Naszym zadaniem było skonfigurować mikrokontroler, tak, aby po podłączeniu do komputera zgłosił się jako kontroler MIDI. Interfejs użytkownika powinien pozwalać na generowanie przynajmniej kilku dźwięków.



Wykorzystane platformy i sprzęt

F746ZG-Nucleo

STM32CubeIDE
STM32CubeMX

Raspberry Pi Pico W


Mu Editor
CircuitPython

Moduł klawiatury
matrycowej 4x4 od
Waveshare



STM32F746ZG-Nucleo

Przygotowanie

1. Pobraliśmy STM32CubeMX i wygenerowaliśmy strukturę projektu dla naszej płytki - F746ZG-Nucleo.
 2. Rozpoczęliśmy edycję pliku .ioc:
 - I. w Connectivity ustawiliśmy USART3
 - II. w Middlewares and Software Packs wybraliśmy FREERTOS oraz USB_DEVICE
- 

Konfiguracja kontrolera MIDI

Na samym początku wybraliśmy piny, którymi łączyliśmy płytkę z modulem klawiatury matrycowej. Piny odpowiadające kolumnom klawiatury ustawiliśmy na input i nadaliśmy własność pull-up w celu uniknięcia zakłóceń - włączyliśmy rezystory.

Tak przygotowany projekt trzeba było przekształcić na urządzenie MIDI. Ponieważ dostarczone sterowniki nie wspierały klasy MIDI Device Class, naszym zadaniem było manualnie przekształcić na nią jedną z predefiniowanych klas. Według odnalezionych przez nas [informacji](#), aby to uzyskać, należało ustawić odpowiednią klasę urządzenia w USB_DEVICE w pliku .ioc i zmodyfikować wygenerowane pliki.

Konieczne modyfikacje

1

Wybraliśmy klasę Audio Device Class, Human Interface Device Class albo Communication Device Class w USB_DEVICE.

2

Podmieniliśmy pliki `usbd_xxx.h` oraz `usbd_xxx.c` na `usbd_midi.c` oraz `usbd_midi.h` proponowane w przykładach aplikacji MIDI przez STMicroelectronics.

3

W plikach `usbd_device.c` oraz `usbd_desc.c` poprawiliśmy wszystkie wspomniane konfiguracje tak, aby wspierały urządzenie MIDI.

Efekt


Mimo dokładnego wykonania powyższych kroków dla różnych klas urządzeń jak i ostatecznie dostosowania różnych opisów rozwiązań i projektów innych osób do naszej płytki nie udało nam się skonfigurować mikrokontrolera – nie był widoczny jako urządzenie MIDI.



Raspberry Pi Pico W

W wyniku problemów z rozwijaniem projektu na platformie opisanej wcześniej, podjęliśmy decyzję o podjęciu próby implementacji projektu na innej platformie.

Przygotowanie

1. Pobraliśmy plik UF2 pozwalający na programowanie Pico:
[Raspberry Pi Pico W UF2 File](#)
 2. Przytrzymaliśmy przycisk BOOTSEL, aby zresetować płytkę i zezwolić na przekopiowanie na nią pobranego pliku
 3. Pobralismy bibliotekę AdaFruit MIDI i przekopiowaliśmy folder adafruit_midi do folderu lib na Pico W
 4. Jeśli w systemie plików nie było code.py, należało go utworzyć (u nas był)
- 

Efekt

1. Wykorzystując moduł **adafruit_midi** przekształciliśmy nasze urządzenie na kontroler MIDI.
2. Zdefiniowaliśmy listę pinów, którymi połączyliśmy płytkę z klawiaturą.

```
import board
import digitalio
import pwmio
import time
import usb_midi
import adafruit_midi
from adafruit_midi.note_on import NoteOn
from adafruit_midi.note_off import NoteOff

midi = adafruit_midi.MIDI(midi_out=usb_midi.ports[1], out_channel=0)

print("MacroPad MIDI Board")

button_pins = [board.GP0, board.GP1, board.GP2, board.GP3, board.GP4, board.GP5, board.GP6, board.GP7]
keys = [['ENT', '0', 'ESC', 'ONOFF'],
        ['7', '8', '9', 'LOCK'],
        ['4', '5', '6', 'GO'],
        ['1', '2', '3', 'STOP']]

rows = [board.GP4, board.GP5, board.GP6, board.GP7]
columns = [board.GP0, board.GP1, board.GP2, board.GP3]

keypad_rows = []
keypad_columns = []

for i in rows:
    keypad_rows.append(digitalio.DigitalInOut(i))

for i in columns:
    keypad_columns.append(digitalio.DigitalInOut(i))

col_pins = []
row_pins = []
```

```

for pin in keypad_rows:
    pin.direction = digitalio.Direction.OUTPUT
    pin.value = False
    row_pins.append(pin)

for pin in keypad_columns:
    pin.direction = digitalio.Direction.INPUT
    pin.pull = digitalio.Pull.UP
    col_pins.append(pin)

notes= [['F3', 'G3', 'A3', 'B3'],
        ['C3', 'D3', 'E3', 'D4'],
        ['G4', 'A5', 'B5', 'C5'],
        ['D5', 'E5', 'F6', 'G6']
        ]

def fix_layout(to_fix):
    n=len(to_fix)
    fixed=[-1] for _ in range(n)
    for i in range(n):
        fixed[(i+n-1)%n]=to_fix[i]
    return fixed

note_mapping=fix_layout(notes)
keys_pressed = [[False for _ in range(4)] for _ in range(4)]
def scankeys():
    for row in range(0, 4):
        for col in range(0, 4):
            row_pins[row].value = False
            key = None

            key_press = note_mapping[row][col]
            if col_pins[col].value == False and keys_pressed[row][col]==False:
                keys_pressed[row][col] = True
                print("You have pressed:", keys[row][col], key_press)
                midi.send(NoteOn(key_press, 60))

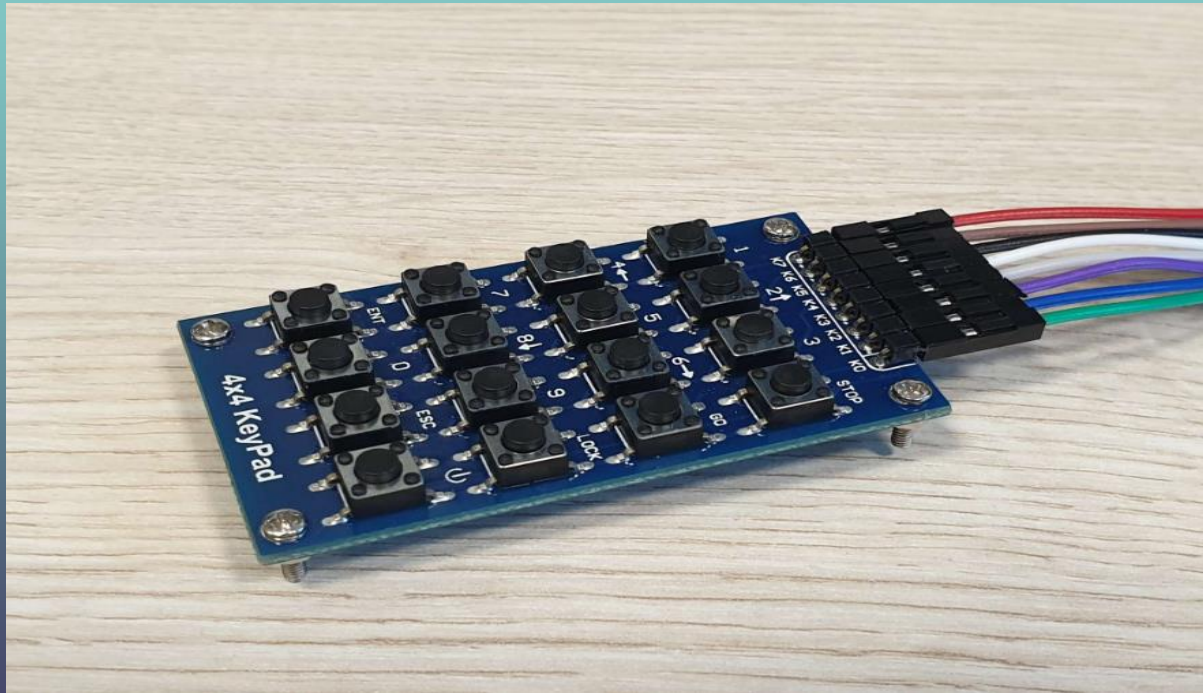
            if col_pins[col].value == True and keys_pressed[row][col] == True:
                keys_pressed[row][col] = False
                midi.send(NoteOff(key_press, 0))
            row_pins[row].value = True

while True:
    scankeys()

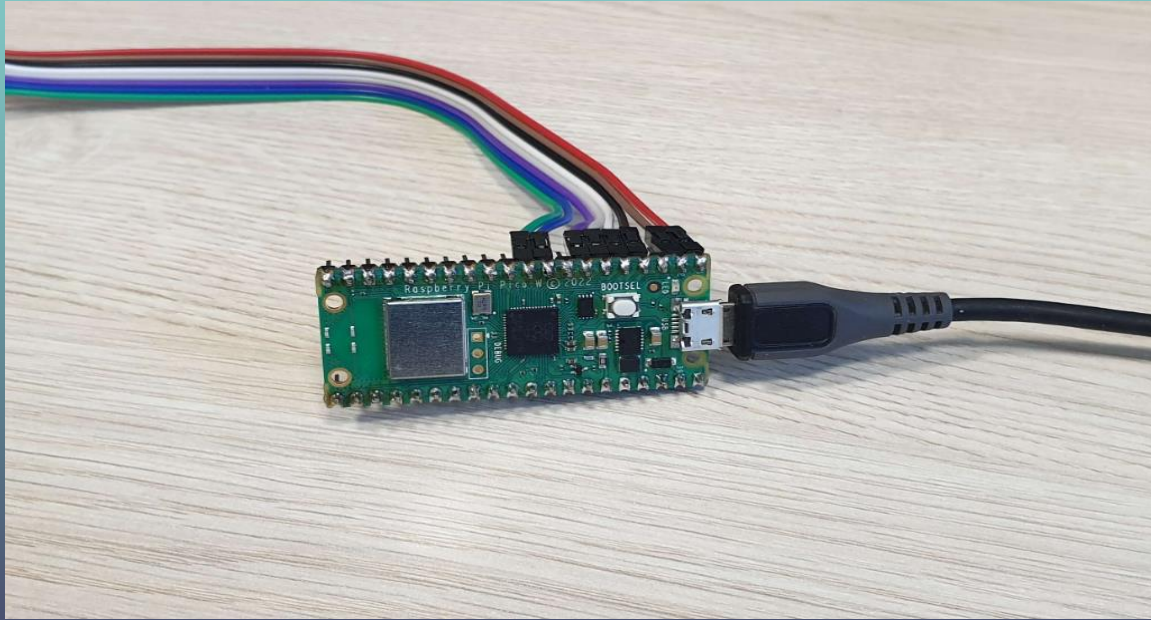
```

3. Przy użyciu modułu digitalio nadaliśmy pinom odpowiednie stany i własności. W naszym przypadku na pinach w kolumnach ustawiliśmy input oraz włączyliśmy rezystory pull up, aby uniknąć błędnych odczytów i mieć stały stan pinu w momencie kiedy przypisany mu przycisk nie jest wciśnięty. Wybrane piny w wierszach ustawiliśmy na output oraz ich wartość na false.
4. Przypisaliśmy wybrane przez nas nuty do klawiszy klawiatury.
5. Funkcja fix_layout() pomaga nam przestawić wiersze tak, aby bardziej intuicyjnie definiowało się mapę nut.
6. scankeys() nasłuchuje na wciśnięcie klawisza i pozwala na wysłanie sygnału MIDI przy pomocy funkcji midi.send. Uwzględniane jest również przytrzymanie klawisza przy wykorzystaniu tablicy keys_pressed.

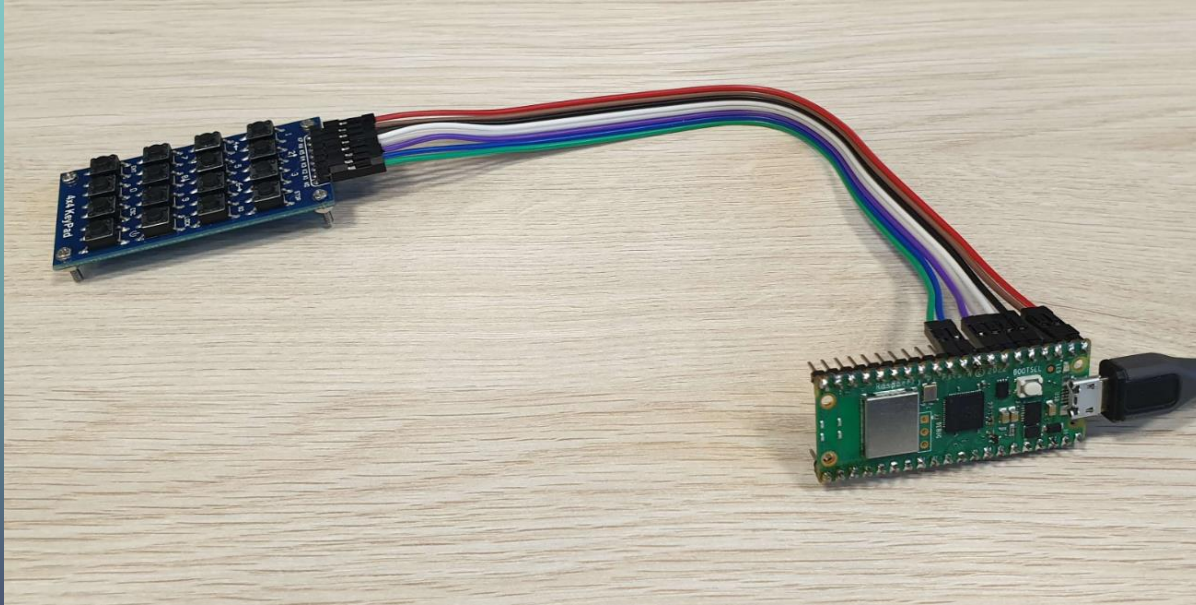
Klawiatura matrycowa



Płytki Raspberry Pi Pico W



Kontroler MIDI





Podsumowanie

Ostatecznie w wyniku naszej pracy udało się stworzyć kontroler MIDI na platformie Raspberry Pi Pico W, a jego działanie przetestowaliśmy i zaprezentowaliśmy na zajęciach.