PicoMacroPAD

Introducere

Un Macropad programabil cu scopul de a eficientiza lucrul pe calculator. Am plecat de la dorinta de a controla mai eficient functionalitati ale laptopului meu in timp ce lucrez precum:

- Control audio pentru device-uri in/out;
- Performare task-uri repetitive prin simpla apasare a unui buton;
- Posibilitatea de extindere a astfel de functionalitati;

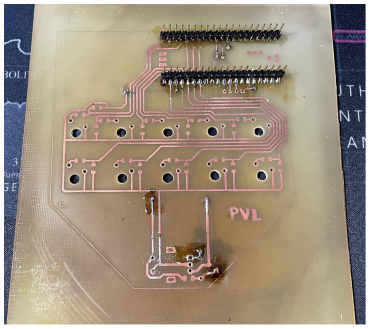
Acest proiect este util pentru minimizarea timpului mort

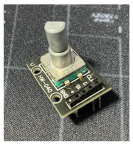
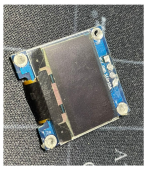
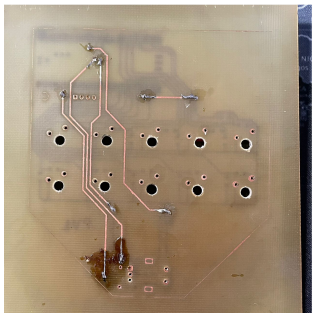
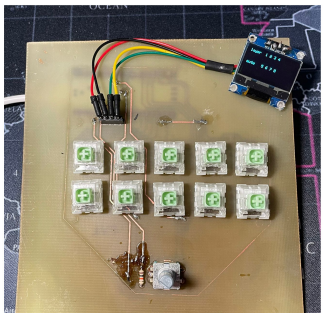
Descriere generală

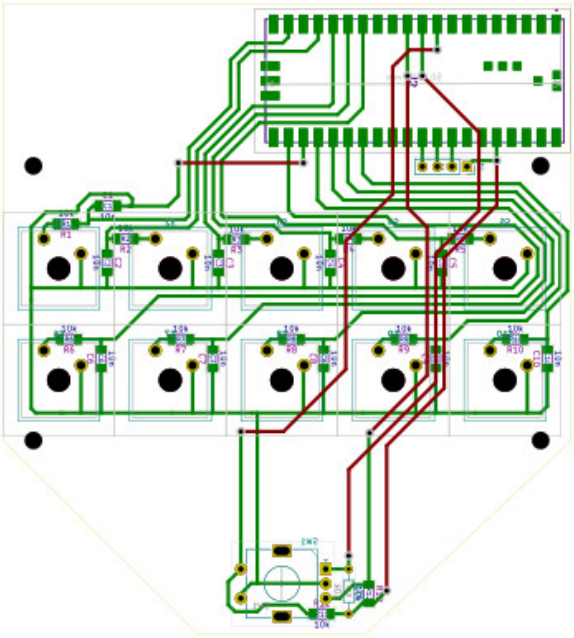
Hardware Design

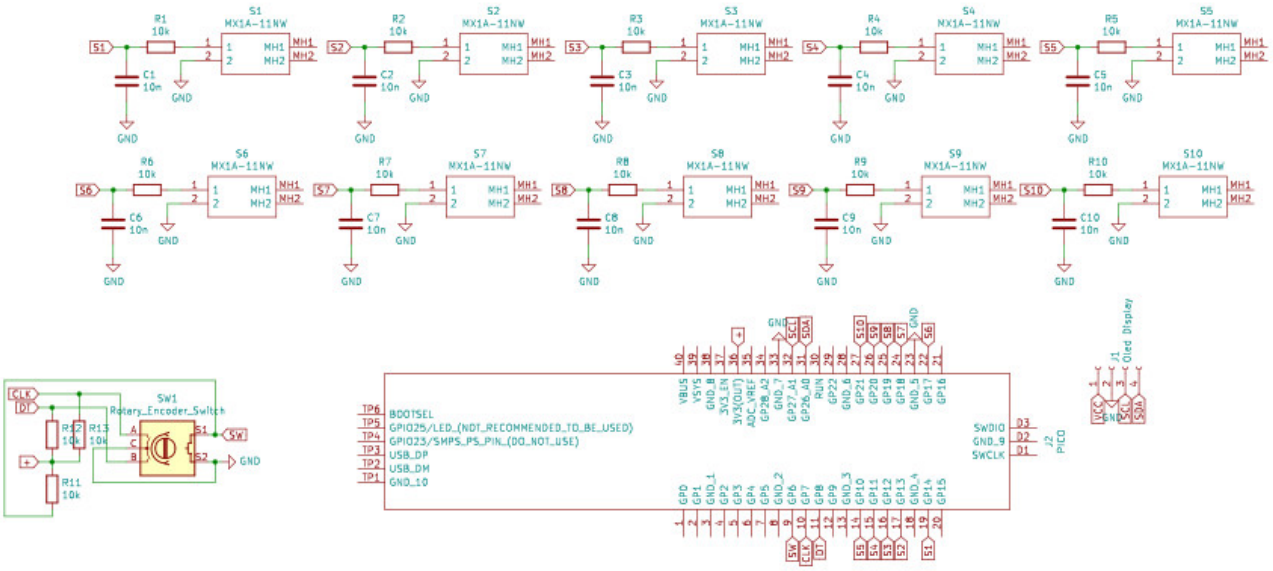
Lista componente:

- Rasberry Pi Pico
- Display OLED 128×64 pixels
- Rotary Encoder
- 10 Kailh Box Jade mechanical switches
- Hand made 2 layer copper etched pcb
- 10 12k SMD 1206 resistors
- 2 10k SMD 0805 resistors
- 1 10k Through Hole resistor
- 10 10nf SMD 0805 capacitors
- Pin headers and wires







Software Design

Codul a fost dezvoltat folosind limbajul CircuitPython in Thonny. Biblioteci externe:

- Adafruit Framebuf
- Adafruit SSD1306
- Adafruit HID
- Adafruit ImageLoader

Codul citeste semnale de la cele 10 butoane si de la codorul rotativ, inclusiv butonul acestuia

Pentru a programa functionalitatea celor 10 butoane, trebuie definita o matrice de 5×2 cu functionalitatea pentru foecare butoane. Functionalitatea butoanelor este preluata din biblioteca Adafruit HID.

Pentru controlul OLED-ului se foloseste biblioteca Adafruit SSD1306 si Adafruit ImageLoader. Se pot scrie mesaje caracter cu caracter, se pot incarca imagini sau se pot seta pixeli arbitrar pe o matrice de 128×64 pixeli.

```
import rotaryio
import board
import digitalio
import displayio
import adafruit_imageload
import usb_hid
from adafruit_hid.consumer_control import ConsumerControl
from adafruit_hid.consumer_control_code import ConsumerControlCode
import time
from digitalio import DigitalInOut, Direction, Pull
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keycode import Keycode
```

```
import busio
import adafruit_ssd1306

#display
i2c = busio.I2C(board.GP27, board.GP26)
display = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c)


# left encoder
buttonL = digitalio.DigitalInOut(board.GP6)
buttonL.direction = digitalio.Direction.INPUT
buttonL.pull = digitalio.Pull.UP

encoderL = rotaryio.IncrementalEncoder(board.GP7, board.GP8)

cc = ConsumerControl(usb_hid.devices)

button_stateL = None
last_positionL = encoderL.position

# keyboard

led = DigitalInOut(board.LED)
led.direction = Direction.OUTPUT
led.value = True

kbd = Keyboard(usb_hid.devices)
cck = ConsumerControl(usb_hid.devices)


pins = [
    board.GP13,
    board.GP12,
    board.GP11,
    board.GP10,
    board.GP16,
    board.GP17,
    board.GP18,
    board.GP19,
    board.GP20,
    board.GP21,
]

MEDIA = 1
KEY = 2
META = 3

# functionality of each layer
keymap_layers = [{
    (0): (KEY, [Keycode.ONE]),
    (1): (KEY, [Keycode.TWO]),
```

```
    (2): (KEY, [Keycode.THREE]),
    (3): (KEY, [Keycode.FOUR]),
    (4): (META, [Keycode.CAPS_LOCK]),
    (5): (KEY, (Keycode.SHIFT, Keycode.GUI, Keycode.M)),
    (6): (KEY, [Keycode.FIVE]),
    (7): (KEY, [Keycode.SIX]),
    (8): (KEY, [Keycode.SEVEN]),
    (9): (KEY, [Keycode.EIGHT]),
},
{
    (0): (KEY, [Keycode.NINE]),
    (1): (KEY, [Keycode.ZERO]),
    (2): (KEY, (Keycode.SHIFT, Keycode.NINE)),
    (3): (KEY, (Keycode.SHIFT, Keycode.ZERO)),
    (4): (META, [Keycode.CAPS_LOCK]),
    (5): (KEY, (Keycode.SHIFT, Keycode.GUI, Keycode.M)),
    (6): (KEY, (Keycode.SHIFT, Keycode.THREE)),
    (7): (KEY, (Keycode.SHIFT, Keycode.FOUR)),
    (8): (KEY, (Keycode.SHIFT, Keycode.SEVEN)),
    (9): (KEY, (Keycode.SHIFT, Keycode.EIGHT)),
}]

keymap = keymap_layers[0]

layer = True
mute = False

# display layers based on keyboard layer
def change_layer(mute):
    global keymap
    if layer:
        display.fill(0)
        keymap = keymap_layers[not layer]
        display.text("layer  1 2 3 4", 0, 0, 1)
        if mute == True:
            display.text("unmute 5 6 7 8", 0, 32, 1)
        else:
            display.text("mute   5 6 7 8", 0, 32, 1)
        display.show()
    else:
        display.fill(0)
        keymap = keymap_layers[not layer]
        display.text("layer  9 0 ( )", 0, 0, 1)
        if mute == True:
            display.text("unmute # $ & *", 0, 32, 1)
        else:
            display.text("mute   # $ & *", 0, 32, 1)
        display.show()

switches = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
for i in range(10):
    switches[i] = DigitalInOut(pins[i])
    switches[i].direction = Direction.INPUT
    switches[i].pull = Pull.UP

switch_state = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

# init
change_layer(mute)

while True:
    # left encoder
    current_positionL = encoderL.position
    position_changeL = current_positionL - last_positionL
    if position_changeL > 0:
        for _ in range(position_changeL):
            cc.send(ConsumerControlCode.VOLUME_DECREMENT)
        #print(current_positionL)
    elif position_changeL < 0:
        for _ in range(-position_changeL):
            cc.send(ConsumerControlCode.VOLUME_INCREMENT)
        #print(current_positionL)
    last_positionL = current_positionL
    if not buttonL.value and button_stateL is None:
        button_stateL = "pressed"
    if buttonL.value and button_stateL == "pressed":
        cc.send(ConsumerControlCode.MUTE)
        button_stateL = None

    # keyboard
    for button in range(10):
        if switch_state[button] == 0:
            if not switches[button].value:
                try:
                    if keymap[button][0] == KEY:
                        if button == 5:
                            mute = not mute
                            change_layer(mute)
                        kbd.press(*keymap[button][1])
                    else:
                        if button == 4:
                            layer = not layer
                            change_layer(mute)
                        else:
                            cck.send(keymap[button][1])
                except ValueError:  # deals w six key limit
                    pass
                switch_state[button] = 1

        if switch_state[button] == 1:
            if switches[button].value:
```

```
                try:
                    if keymap[button][0] == KEY:
                        kbd.release(*keymap[button][1])

                except ValueError:
                    pass
                switch_state[button] = 0

    time.sleep(0.01)  # debounce
```

Rezultate Obţinute

Am dobandit skill-ul de a face un PCB de mana. Am produs un tool pe care il voi folosi in timpul lucrului de zi cu zi

Concluzii

Rezultatul proiectului este un punct bun de inceput prototiparea unui produs care poate fi adus pe piata

Download

picomacropad_surse_pavel-vlad_mateescu.zip

Bibliografie/Resurse

Resurse Software

- Adafruit's CircuitPython docs: https://docs.circuitpython.org/en/latest/shared-bindings/index.html
- Adafruit's CircuitPython libraries catalog: https://learn.adafruit.com/welcome-to-circuitpython/circuitpython-libraries
- I2C OLED tutorial: https://lastminuteengineers.com/oled-display-arduino-tutorial/

Resurse Hardware.

- Raspberry Pi Pico datasheet: https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf
- Generic I2C OLED datasheet: https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf
- Generic Rotary Encoder description page: https://components101.com/modules/KY-04-rotary-encoder-pinout-features-datasheet-working-application-alternative

Export to PDF

From:
http://ocw.cs.pub.ro/courses/ - **CS Open CourseWare**

Permanent link:
http://ocw.cs.pub.ro/courses/pm/prj2022/arosca/picomacropad

Last update: **2022/05/27 22:12**