:GAME ZIP™ 64 pour BBC micro:bit

www.kitronik.co.uk/5626

Le :GAME ZIP 64 est une manette de jeu programmable pour BBC micro:bit. Il dispose de 64 LED adressables de couleur disposées dans un affichage 8 x 8, d'un buzzer piézo pour retour audio, d'un moteur vibrant pour retour tactile et de 6 boutons de saisie. P19, P20 et LED DOUT sont positionnées avec un encombrement standard de 2,54mm. Chacune de ces broches possède également la tension et les embases GND requises. Le BBC micro:bit est raccordé via un connecteur latéral de slot de carte standard.

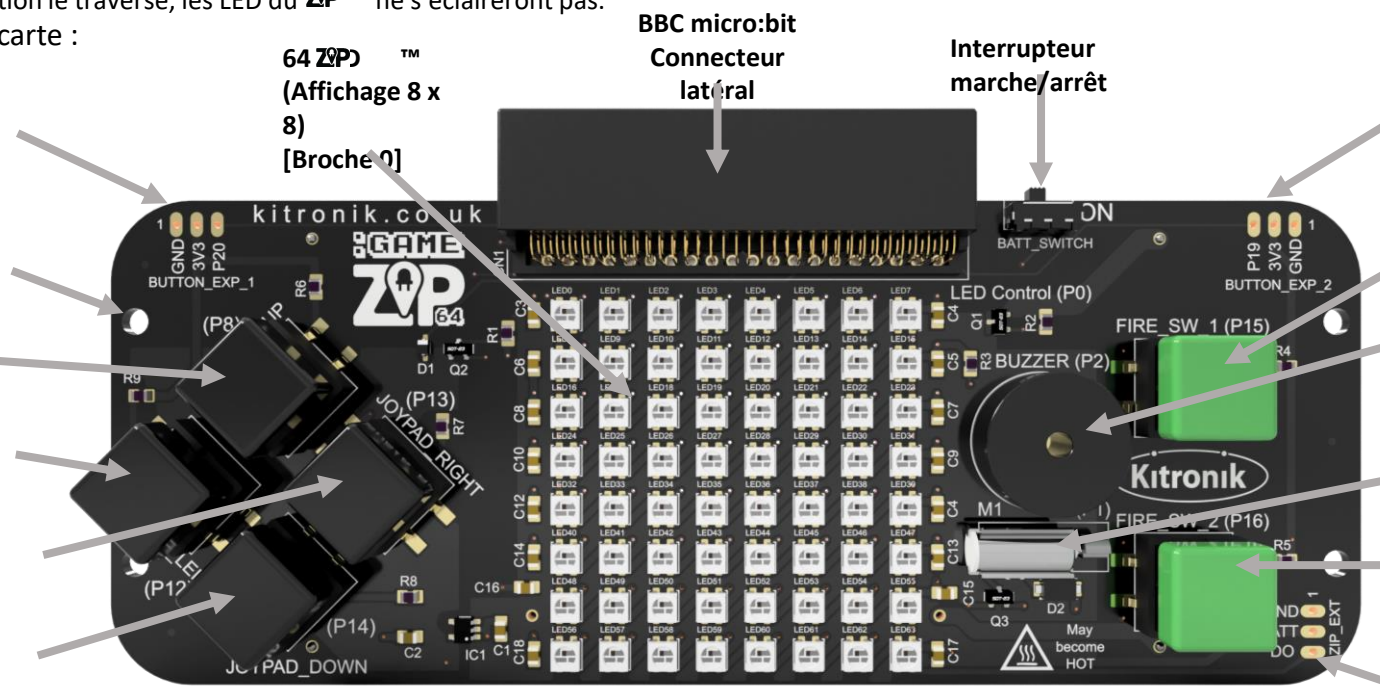
La carte délivre également du **courant régulé** qui est transmis aux connecteurs 3 Vcc et GND **pour alimenter le BBC micro:bit connecté**, supprimant ainsi la nécessité d'alimenter le BBC micro:bit séparément. Afin de protéger le BBC micro:bit si l'alimentation le traverse, les LED du ZIP ™ ne s'éclaireront pas.

Insérer un BBC micro:bit :

Pour utiliser le :GAME ZIP™ 64, le BBC micro:bit doit être inséré fermement dans le connecteur latéral, en veillant à ce que l'affichage LED du BBC micro:bit soit orienté dans le même sens que l'affichage LED du :GAME ZIP ™ .

Exemples : Pour obtenir des jeux pour débutants ou des idées, rendez-vous sur : http://www.kitronik.co.uk/5626

Agencement de la carte :



Embases d'extension des broches :
Gauche – GND
Centre – 3,3 V
Droite – Broche 20

4 x trous de fixation M3

Manette vers le haut [Broche 8]

Manette vers la gauche [Broche 12]

Manette vers la droite [Broche 13]

Manette vers le bas [Broche 14]

Arrière : 3 x supports de piles AA

64 ZIP ™ (Affichage 8 x 8) [Broche 0]

BBC micro:bit Connecteur latéral

Interrupteur marche/arrêt

Embases d'extension des broches :
Gauche – Broche 19
Centre – 3,3 Vcc
Droite – GND

Bouton-poussoir 1 [Broche 15]

Buzzer piézo [Broche 2]

Moteur vibrant [Broche 1]

Bouton-poussoir 2 [Broche 16]

ZIP ™ Embases d'extension :
Haut – GND
Centre - + BATT V
Bas – DOUT

Attention :

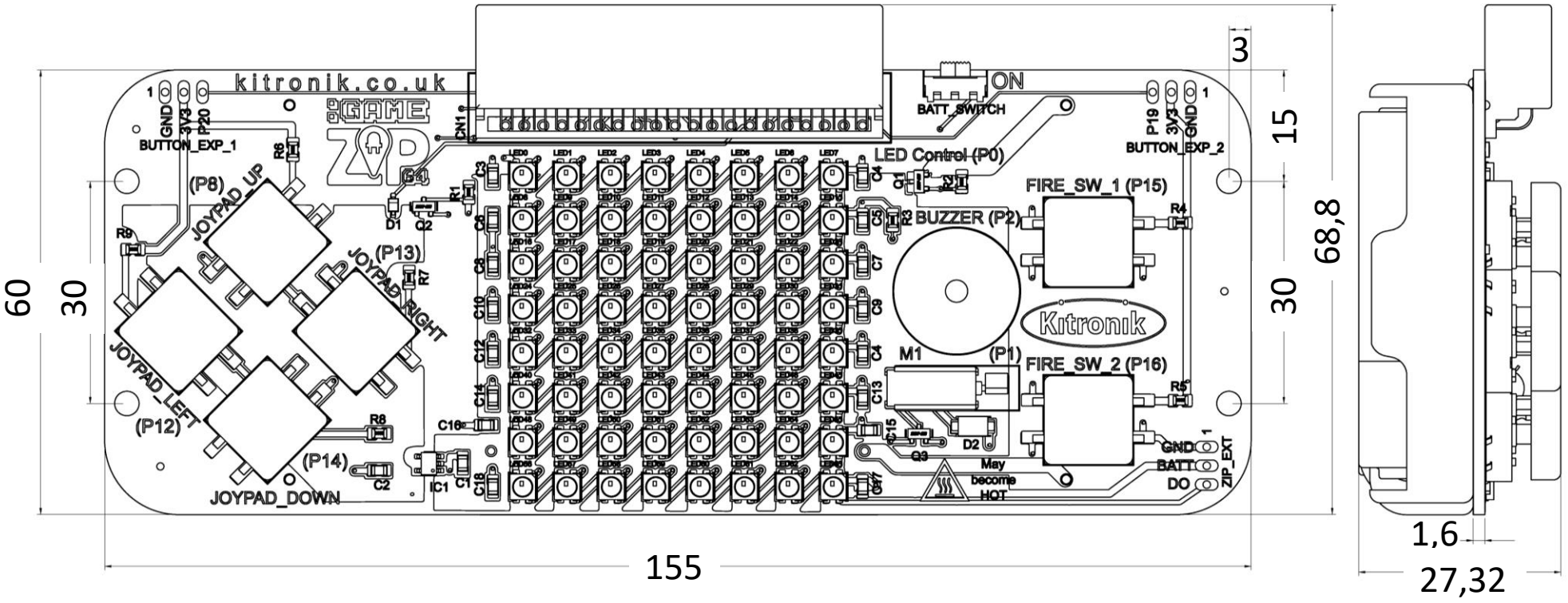
ZIP Les LED ™ peuvent chauffer si elles sont utilisées avec une luminosité élevée pendant des périodes prolongées.

:GAME ZAP™ 64 pour BBC micro:bit

www.kitronik.co.uk/5626

Dimensions de la carte :

(Toutes les mesures sont données en mm)



:GAME ZIP™ 64 pour BBC micro:bit

www.kitronik.co.uk/5626

Informations électriques

Tension de service (Vcc) [LED ZIP]	+3,5 V – +5,3 V
Tension régulée [BBC micro:bit, boutons, moteur vibrant]	+3,3 V
Courant maxi (LED ZIP fonctionnant en pleine luminosité RVB)	1,6 A (21 mA par LED ZIP, 250 mA max. avec tension rég. de + 3,3 V)
Nombre de LED ZIP	64
Nombre de canaux externes	3 (1 x LED ZIP, 2 x broches E/S I2C, courant nominal de chaque broche E/S + 3,3 V à 5 mA)

Remarque sur les canaux externes :

Des précautions doivent être prises lors de l'utilisation des connexions externes des broches 19 et 20 en tant que GPIO, car des problèmes peuvent survenir avec les périphériques I2C sur le BBC micro:bit (par exemple : boussole et accéléromètre).

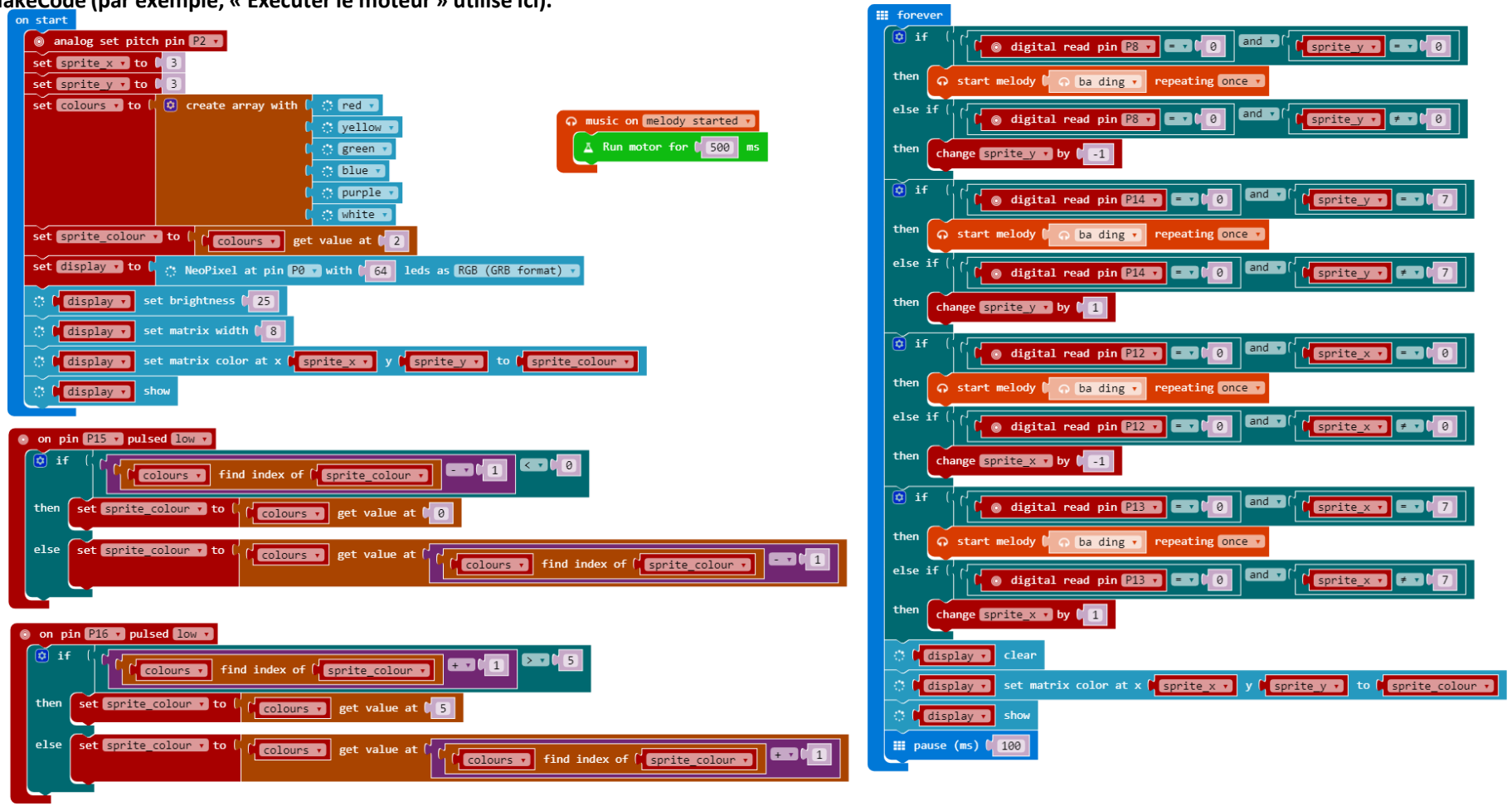
Vue arrière avec BBC micro:bit et piles :

:GAME ZAP™ 64 pour BBC micro:bit

www.kitronik.co.uk/5626

Code de l'éditeur de blocs Microsoft MakeCode

Ce programme a été créé dans l'éditeur de blocs Microsoft MakeCode pour le BBC micro:bit. Il crée un lutin à pixel unique qui peut être déplacé autour de l'affichage à l'aide des boutons de la manette et dont la couleur est modifiée à l'aide des boutons-poussoirs. Lorsque le lutin atteint le bord de l'affichage, le moteur vibre et le buzzer retentit brièvement. **Remarque : certains blocs personnalisés de Kitronik sont disponibles pour le :GAME ZAP™ 64 sur Microsoft MakeCode (par exemple, « Exécuter le moteur » utilisé ici).**



:GAME ZAP™ 64 pour BBC micro:bit

www.kitronik.co.uk/5626

Code de l'éditeur MicroPython

Ce programme a été créé dans l'éditeur MU MicroPython pour le BBC micro:bit. Il fournit exactement la même fonctionnalité que le programme de blocs MakeCode.

```
from microbit import *
import neopixel
import music

# Enable ZIP LEDs to use x & y values
def zip_plot(x, y, colour):
    zip_led[x+(y*8)] = (colour[0], colour[1], colour[2])

# Function to play tune on buzzer and run motor for 500ms
def hit_edge():
    music.play(music.BA_DING, pin2, False)
    pin1.write_digital(1)
    sleep(500)
    pin1.write_digital(0)

# Setup variables and initial ZIP LED display
zip_led = neopixel.NeoPixel(pin0, 64)
sprite_x = 3
sprite_y = 3

# Colours: Red, Yellow, Green, Blue, Purple, White
colours = [[20, 0, 0], [20, 20, 0], [0, 20, 0], [0, 0, 20], [20, 0, 20], [20, 20, 20]]
sprite_colour = colours[3]
zip_plot(sprite_x, sprite_y, sprite_colour)
zip_led.show()

# While loop to run forever
while True:
    # Check button presses
    if pin8.read_digital() == 0 and sprite_y == 0:
        hit_edge()
    elif pin8.read_digital() == 0 and sprite_y != 0:
        sprite_y = sprite_y - 1
    if pin14.read_digital() == 0 and sprite_y == 7:
        hit_edge()
    elif pin14.read_digital() == 0 and sprite_y != 7:
        sprite_y = sprite_y + 1

    if pin12.read_digital() == 0 and sprite_x == 0:
        hit_edge()
    elif pin12.read_digital() == 0 and sprite_x != 0:
        sprite_x = sprite_x - 1

    if pin13.read_digital() == 0 and sprite_x == 7:
        hit_edge()
    elif pin13.read_digital() == 0 and sprite_x != 7:
        sprite_x = sprite_x + 1

    if pin15.read_digital() == 0:
        if colours.index(sprite_colour) - 1 < 0:
            sprite_colour = colours[0]
        else:
            sprite_colour = colours[(colours.index(sprite_colour) - 1)]

    if pin16.read_digital() == 0:
        if colours.index(sprite_colour) + 1 > 5:
            sprite_colour = colours[5]
        else:
            sprite_colour = colours[(colours.index(sprite_colour) + 1)]

    # Clear and redisplay the NeoPixels after each button press check
    zip_led.clear()
    zip_plot(sprite_x, sprite_y, sprite_colour)
    zip_led.show()

    # 100ms pause before restarting the while loop
    sleep(100)
```