BBC micro:bit
Bit:Commander Pushbuttons

Introduction

The 4 pushbuttons on the Bit:Commander are arranged as they would be on a game controller, where they might be used for shooting, action buttons, jumping, selecting and so on. They are also conveniently placed if you want to use them for directional control. When the screen is small that is sometimes better than a joystick.

Programming

The buttons don't need any special treatment. You just need to remember which pins are connected to which button. The relevant section of the pinout is,

Component	Pin
Red Pushbutton	12
Blue Pushbutton	15
Green Pushbutton	14
Yellow Pushbutton	16

Simple State Check

In MicroPython, you don't have any events. For something like a button press, which happens at an arbitrary moment in time, you need to have your program keep checking the state of the button. In its simplest form, you can write a program like this,

```
from microbit import *

while True:
    btn_red = pin12.read_digital()
    if btn_red:
        display.show(Image.HAPPY)
    else:
        display.show(Image.SAD)
    sleep(20)
```

The program has an infinite loop, a while loop with a stopping condition that cannot be met. The state of the red button is read. If it is held down, a happy face is shown, otherwise a sad face is shown. The small delay determines the frequency with which the button is checked or 'polled'.

State Change

You don't always want to have a user holding a button down for something to happen. Often you want to trigger something once when a button is pressed. To do this, you can use a 'follower' variable to keep track of

BBC Microbit

Collapse All Expand All

+ Block Editor - The Basics
+ Block Editor - Components
+ Kodu - micro:bit Worlds
+ JavaScript Blocks
+ JavaScript Blocks - Exercises
+ Blocks - Bit:Bot
+ Blocks - Bit:Commander
+ MicroPython - Starting Off
+ MicroPython - Examples
+ MicroPython - Components
+ MicroPython - Breakout Boards
+ MicroPython - Exercises
+ MicroPython - Pi Accessories
+ MicroPython - Bit:Bot
- MicroPython - Bit:Commander
  - Bit:Commander
  - The Joystick
  - The Neopixels
  - The Potentiometer
  - The Pushbuttons
  - The Buzzer
  - Evasion Game
  - Light's Out Game
  - Simon Game
  - Bit:Bot/Robot Controller
  - Text Entry
  - Unicorn Commander
+ MicroPython - Projects
+ MicroPython - Visual Basic
+ Other - Odds & Ends

the previous button state.

```
from microbit import *

last = 0
while True:
    btn_red = pin12.read_digital()
    if btn_red!=last:
        if btn_red:
            # pressed
            display.show(Image.HAPPY)
        else:
            #released
            display.show(Image.SAD)
    last = btn_red
    sleep(20)
```

In this program, the display is only changed if the button state changes. When the program starts, there is nothing on the display. The moment that the button is pressed, it changes to a happy face. A sad face is shown when the button is released.

In the previous example, the display was being updated every single iteration, even if the button wasn't being used at all. In this version, the display only gets updated when the button state changes.

Press Or Release

Each time you read a button, there are 4 things you could determine. You can work it if it wasn't touched since your last reading. You can know if it was pressed, released or held down too. In some projects, I want to be able to trigger an event for a single press of the button. Depending on the application, it might feel more natural to do this as the button is released, rather than when it is pressed.

Since each button state could be stored as a bit, I like to form a binary number from them, by left shifting the last reading and adding it to the current. This makes an integer from 0 to 3. By examining the value of that integer, you can determine a few more things about your button reading to decide how to respond.

```
from microbit import *

last = 0
while True:
    btn_red = pin12.read_digital()
    e = (last<<1) + btn_red
    if e==0:
        display.show(Image.SAD)
    elif e==1:
        print("Pressed")
    elif e==2:
        print("Released")
    else:
        display.show(Image.HAPPY)
    last = btn_red
    sleep(20)
```

All Button States

We can also use a binary pattern to represent all of the button states. That gives you a single integer that you can examine to find out the state of any button. This program prints out the integer in denary and binary.

```
from microbit import *

# read the buttons and use binary 4 bits to represent
# the button states
def get_btns():
    pattern = 0
    for i,p in enumerate([pin12,pin15,pin14,pin16]):
        pattern += p.read_digital() << i
    return pattern

while True:
    b = get_btns()
    print(b, '{:04b}'.format(b))
    sleep(50)
```

Using a follower with the 4 bit integer lets you know if there was a change to the state of any button. That can be particularly useful to avoid having a display refreshing too often or a musical note changing.

More Button Stuff

This program was written when I was trying to work out a tidy approach to using the buttons together.

```
# Reading the buttons on the Bit:Commander
from microbit import *

# read the buttons and use binary 4 bits to represent
# the button states
def get_btns():
    pattern = 0
    for i,p in enumerate([pin12,pin15,pin14,pin16]):
        pattern += p.read_digital() << i
    return pattern

# function to compare current and last readings
# and return a list of buttons pressed
def btn_pressed(prev, current):
    # concatenate each pair of bits
    evts = [((prev >> i & 1)<<1) + (current >> i & 1) for i in range(4)]
    # return the bit positions equalling 1 (01)
    return [i for i,e in enumerate(evts) if e==1]

# function to compare current and last readings
# and return a list of buttons released
```

```
def btn_released(prev, current):
    # concatenate each pair of bits
    evts = [((prev >> i & 1)<<1) + (current >> i & 1) for i in range(4)]
    # return the bit positions equalling 2 (10)
    return [i for i,e in enumerate(evts) if e==2]

# button colour names
btns = ["Red", "Blue", "Green", "Yellow"]

# store previous button state for each reading
last = 0
while True:
    btnstate = get_btns()
    if btnstate!=last:
        # get presses and releases
        p = btn_pressed(last, btnstate)
        r = btn_released(last, btnstate)
        # match up button names
        pp = [btns[i] for i in p]
        rr = [btns[i] for i in r]
        print("State:", btnstate,"Pressed:",pp, p, "Released:",rr,r)
    last = btnstate
    sleep(20)
```

I use the function to make a binary integer of the button states in most of the programs where I use the Bit:Commander buttons. I also find the line labelled **# concatenate each pair of bits** very useful. It creates a list of the 2 bit integers for each button.