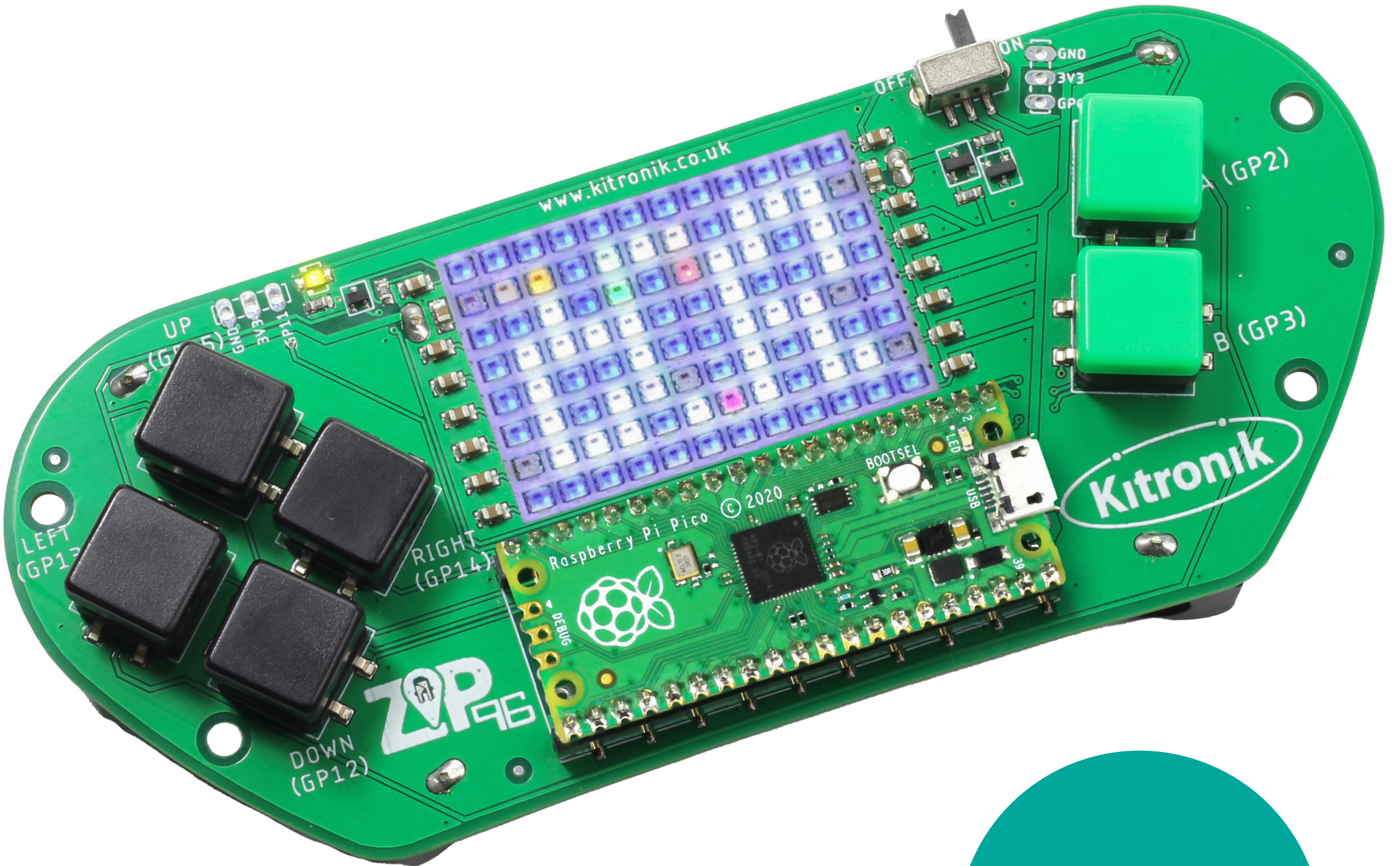
LESSON GUIDE TO THE PICO ZIP96



14+

LESSON 3:
THE A-MAZING GAME

This lesson includes curriculum mapping, practical exercises and a linked PowerPoint presentation.

www.kitronik.co.uk

TEACH YOUR STUDENTS HOW TO CREATE GAMES WITH CODE!

Kitronik

LESSON 3

INTRODUCTION & SETUP

This is the third lesson in the 'A-mazing Game' series for Pico ZIP96. Building on the movement functionality added before, this lesson will look at taking inputs from the Pico ZIP96 and using them to control the player.

CLASSROOM SETUP

Students will be working in pairs. They will need:

- Pen & Paper
- A computer/laptop with a USB port and Internet access
- Raspberry Pi Pico H
- Kitronik Pico ZIP96
- 3 x AA batteries
- A micro USB cable
- A copy of the Kitronik ZIP96 library added in step 4 of the Thonny setup, or ZIP96Pico.py in Lessons Code folder
- A copy of last lesson's code (ZIP96Pico - A-Mazing Game - Lesson 02.py in Lessons Code folder)

The teacher will be writing on the board as well as demonstrating code on a projected board (if available).

Curriculum mapping

- Understanding tools for writing programs. Using sequence, variables, data types, inputs and outputs.
- Apply iteration in program designs using loops.
- Make decisions in programs, making use of arithmetic, logic and Boolean expressions and operators. Created nested selection statements.
- Decompose problems into smaller components, and make use of subroutines to build up well-structured programs.

KEYWORDS:

TRANSLATORS, INTEGRATED DEVELOPMENT ENVIRONMENTS (IDES), ERRORS, SEQUENCE, VARIABLES, DATA TYPES, INPUTS, OUTPUTS, ITERATION, WHILE LOOPS, FOR LOOPS, NESTED STATEMENTS, DESIGN PROGRAMS, SELECTION, CONTROL STRUCTURES, LOGIC, BOOLEAN, NESTED STATEMENTS, SUBROUTINES, PROCEDURES, FUNCTIONS, MODULES, LIBRARIES, VARIABLE SCOPE, WELL-DESIGNED PROGRAMS

LESSON 3

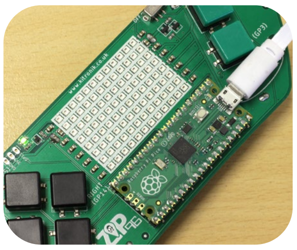
WHAT IS OUR A-MAZING GAME?

Our A-Mazing Game is a maze-based game where the player runs through a maze collecting gems. Once you have collected all the gems in the maze then you have won! But it won't be that easy, as there are enemies in the maze trying to catch you before you collect all of the gems. The three enemies each have their own level of difficulty. The first enemy moves randomly. The second enemy tries to move towards you, without trying to avoid the maze walls. The third and final enemy is smart and moves around the maze walls finding the best path to you. When an enemy catches you, you lose a life and everyone in the game is reset back to their starting positions. You have three lives to collect all the gems, and if you don't, then you lose.

SETUP

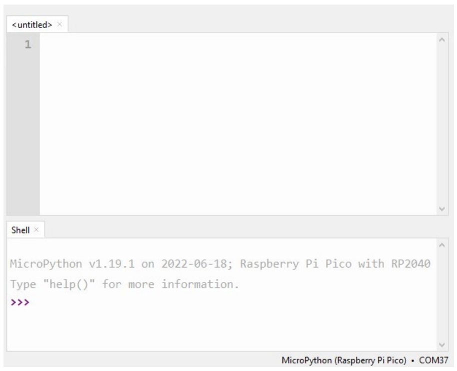
Students will need to work in pairs, having one device per pair.

1. Start by having the students setup the ZIP96 Pico by connecting it to a computer and opening up Thonny.



2. The Pico device will appear in the bottom right corner of Thonny.

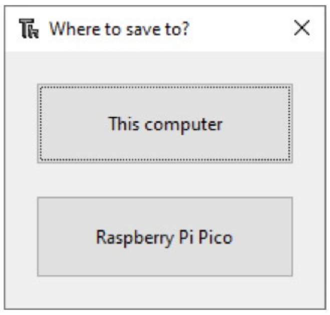
- If the device does not load automatically, try pressing the STOP icon at the top of the screen.
- If the shell does not load automatically, turn it on by checking **View > Shell**.



```
MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
```

MicroPython (Raspberry Pi Pico) • COM37

3. Create a new file by clicking **File > New** and save this to your Pico as main.py by clicking **File > Save as** and selecting **Raspberry Pi Pico.**

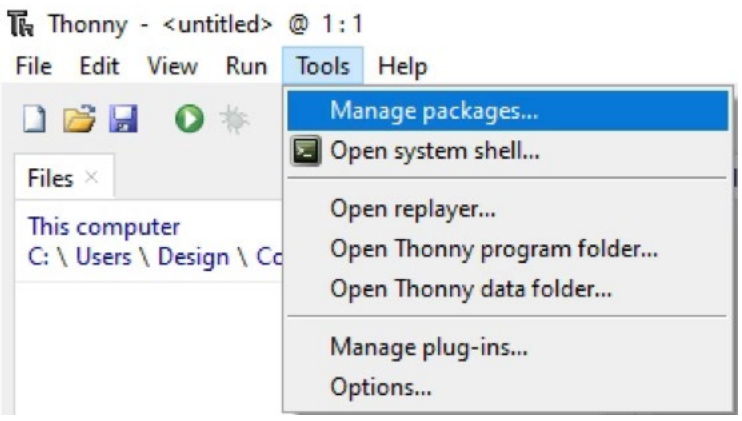


SETUP CONTINUED

4 Now we can install the ZIP96Pico library onto our Pico.

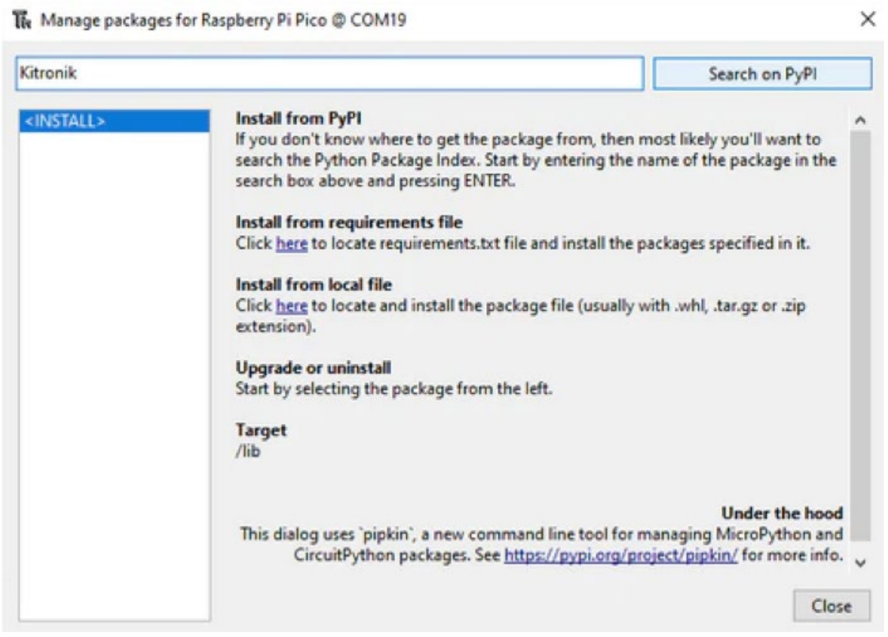
7

- To do this we need to click **Tools > Manage Packages...** from the drop down menu.



- With the Manage packages window open we can now search for Kitronik in the text box at the top, and click the Search on PyPI button. Thonny will search the Python Package Index for all the Kitronik packages.

8



Install from PyPI
If you don't know where to get the package from, then most likely you'll want to search the Python Package Index. Start by entering the name of the package in the search box above and pressing ENTER.

Install from requirements file
Click here to locate requirements.txt file and install the packages specified in it.

Install from local file
Click here to locate and install the package file (usually with .whl, .tar.gz or .zip extension).

Upgrade or uninstall
Start by selecting the package from the left.

Target
/lib

Under the hood
This dialog uses `pipkin`, a new command line tool for managing MicroPython and CircuitPython packages. See https://pypi.org/project/pipkin/ for more info.

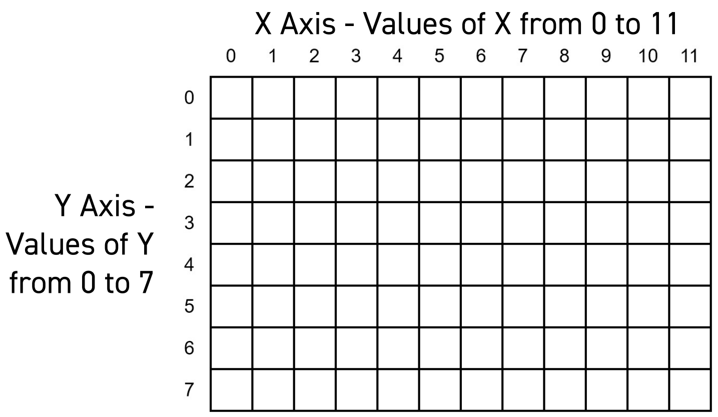
- Click on KitronikPicoZIP96 from the search results list. This will show us details about the package and from here we can click the Install button to add the package to our Pico. Thonny may ask you to confirm that you would like to install this package and we want to select Yes, we do want to install the package.

LESSON 3

MAIN LESSON

HOW OUR PICO ZIP96 SCREEN WORKS

In this lesson we are going to start moving our player around the screen. To do this it is important we understand how the screen works on the Pico ZIP96. Below is a diagram to show the layout of each LED on the Pico ZIP96 screen. The top left corner has an X value of 0 and Y value of 0. While the bottom right corner has an X value of 11 and Y value of 7.


X Axis - Values of X from 0 to 11
0 1 2 3 4 5 6 7 8 9 10 11
Y Axis - Values of Y from 0 to 7
0 1 2 3 4 5 6 7

Using this layout diagram it is easier to see how we can move the player up, down, left, and right. To move up we want to minus 1 from the Y value. To move down we want to add 1 to the Y value. To move left we want to minus 1 from the X value. And to move right we want to add 1 to the X value.

? What feature of the Pico ZIP96 can we use to control the player's movement?

CREATE GAME INPUT LOOP

Currently our player leaves a trail on the screen on where it has previously moved and is able to disappear off the edge of the screen. We will fix these problems later, but first let's get some input from the Pico ZIP96.

To get input from the buttons we are going to need to create a loop to check when a button is pressed. Before we do this we need to setup some more game variables to control how the Pico ZIP96 inputs change the game. We'll add our game variables below where we have setup our Pico ZIP96 Pico variables.

The first game variable is moveDelay and we'll use it to determine how often the player actually moves, in this case half a second. Then we create the moveX and moveY variables which we'll change when one of the buttons is pressed and use them to update which direction our player is moving. By default, set moveX to 1 and moveY to 0 meaning the player will move right across the screen.

```
# Setup game variables
moveDelay = 0.5
moveX = 1
moveY = 0
```

Curriculum mapping
Apply iteration in program designs using loops.

To move the player around the screen we need to provide some way for the buttons to change the position variables that define it. We can do this by increasing and decreasing the values of x and y when the Up, Down, Left and Right buttons on the Pico ZIP96 are pressed. We are going to use an infinite loop to do this by creating a while loop with the condition being 1 or True. This loop will run continuously until we interrupt it, which we'll do later. At the start of the loop let's add a call to the sleep function from the utime library we imported earlier.

12

```
# Start game loop
while 1:
    # Wait for moveDelay before moving
    sleep(moveDelay)
```

Make decisions in programs, making use of arithmetic, logic and Boolean expressions and operators.

To check for when a button is pressed we can access the button object from our gamer and call the pressed function, that returns True when the button has been pressed and False when it hasn't.

13

When the Up button is pressed we want to move the player towards the top of the screen and so we minus 1 from the value of y. When the Down button is pressed we want to move the player towards the bottom of the screen and so we plus 1 onto the value of y.

We then add a similar block of code to handle the movement towards the left edge of the screen when Left is pressed and towards the right edge of the screen when Right is pressed.

```
# When up pressed, change player y position by -1
if (gamer.Up.pressed()):
    moveX = 0
    moveY = -1

# When down pressed, change player y position by -1
if (gamer.Down.pressed()):
    moveX = 0
    moveY = 1

# When left pressed, change player y position by -1
if (gamer.Left.pressed()):
    moveX = -1
    moveY = 0

# When right pressed, change player y position by -1
if (gamer.Right.pressed()):
    moveX = 1
    moveY = 0
```

LESSON
3

TASK: TEST *PLAYER.MOVE* FUNCTION

14

To actually move the player we are going to call the move function and supply it our updated moveX and moveY as its input values. When a button on the Pico ZIP96 is pressed, these values will be changed and will in turn change the direction the player moves. Next, remember to call show on the screen after updating the player position.

Now try using the move function on your player object by using the Pico ZIP96 buttons. See if you can get the player to disappear off the screen.

```
# Update the player position using the move values
player.move(moveX, moveY)

# Show the updates on the screen
screen.show()
```

? How can we fix the issue where the player leaves a trail as they move around the screen? 15

Curriculum mapping
Decompose problems into smaller components, and make use of subroutines to build up well-structured programs.

CREATE GAME FUNCTION: *PLAYER.DRAWEMPTY*

16

To fix the trail of previous moves left by our player we can define a new function in the Player class called drawEmpty. This function is mostly the same as the draw function but instead of setting the LED colour to the colour stored in the object, we are going to update the LED colour to be black. This will clear the screen of our player object's current position.

```
# Draw a blank space in the current position on the screen
def drawEmpty(self):
      self.screen.setLEDMatrix(self.x, self.y, self.screen.BLACK)
```

Now we need to use the drawEmpty function at the start of the Player.move function before we update the object's position values.

```
def move(self, x, y):
      # Reset the current position on the screen
      self.drawEmpty()
```

? What are some possible solutions for stopping the player moving off the edge of the screen? 17

UPDATE GAME FUNCTION: *PLAYER.MOVE*

To fix the problem where our player can disappear off the edge of the screen we need to add some checking into our move function after the position on the screen has been updated. In our maze game we want the player to be able to wrap around the edge of the screen. This way if they move off the edge on one side of the screen, they will appear back on the other side of the screen.

18

Let's add these checks after we update the x and y values. We need to add four if statements to move, each one checks a different edge on the screen. When the value of x or y is less than zero the player has moved off the left or top of the screen and we want to move them to the right or bottom of the screen. We can do this by setting the position value to screenWidth or screenHeight minus 1. When the value of x or y is more than or equal to the screenWidth or screenHeight the player has moved off the right or bottom of the screen and we want to move them to the left or top of the screen. We can do this by setting the position value to 0.

```
self.y += y

# Check the new x position is valid (not off the edge of the screen)
if (self.x < 0): self.x = self.screenWidth - 1
if (self.x >= self.screenWidth): self.x = 0

# Check the new y position is valid (not off the edge of the screen)
if (self.y < 0): self.y = self.screenHeight - 1
if (self.y >= self.screenHeight): self.y = 0
```

TASK: TEST *PLAYER.MOVE* FUNCTION

Now try using the move function on your player object and see if you can still get the player to disappear off the screen.

LESSON 3

CONCLUSION

LESSON 03 CONCLUSION

In this lesson, you have:

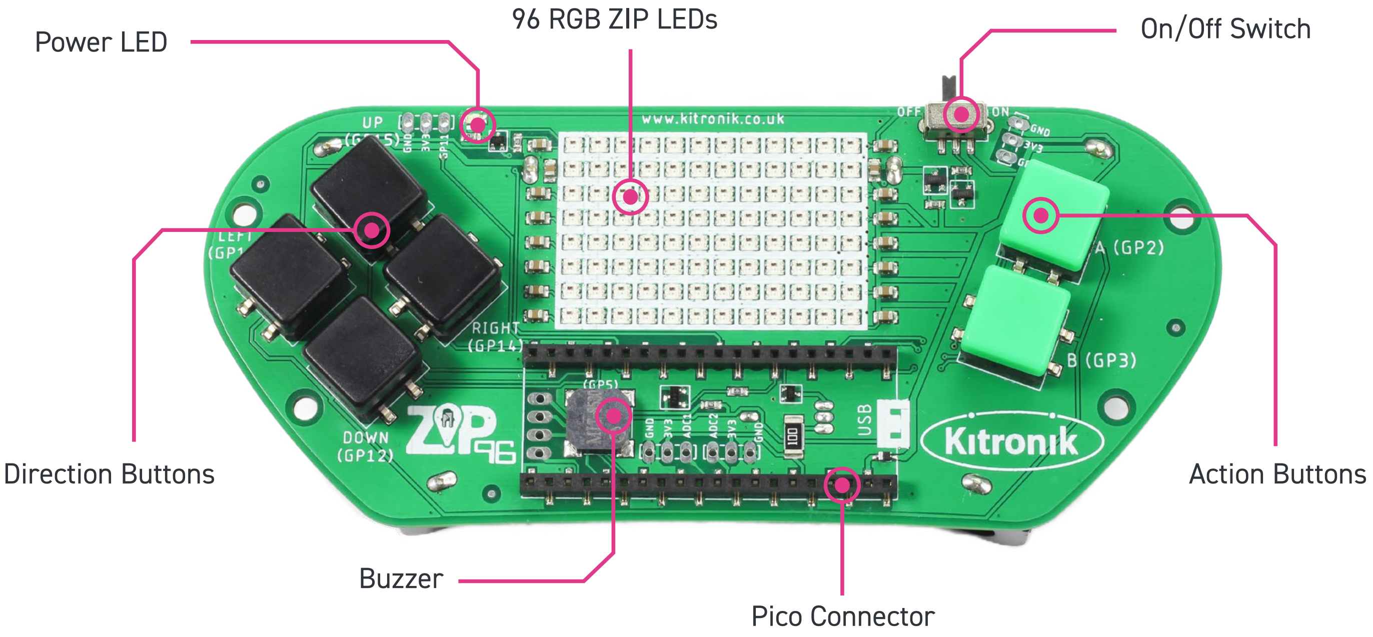
- Used a while loop to read user inputs from the Pico ZIP96
- Used if statements to control the game flow
- Responded to user inputs by updating objects on the screen
- Called methods on the Player object
- Checked position of Player object is within bounds of the maze

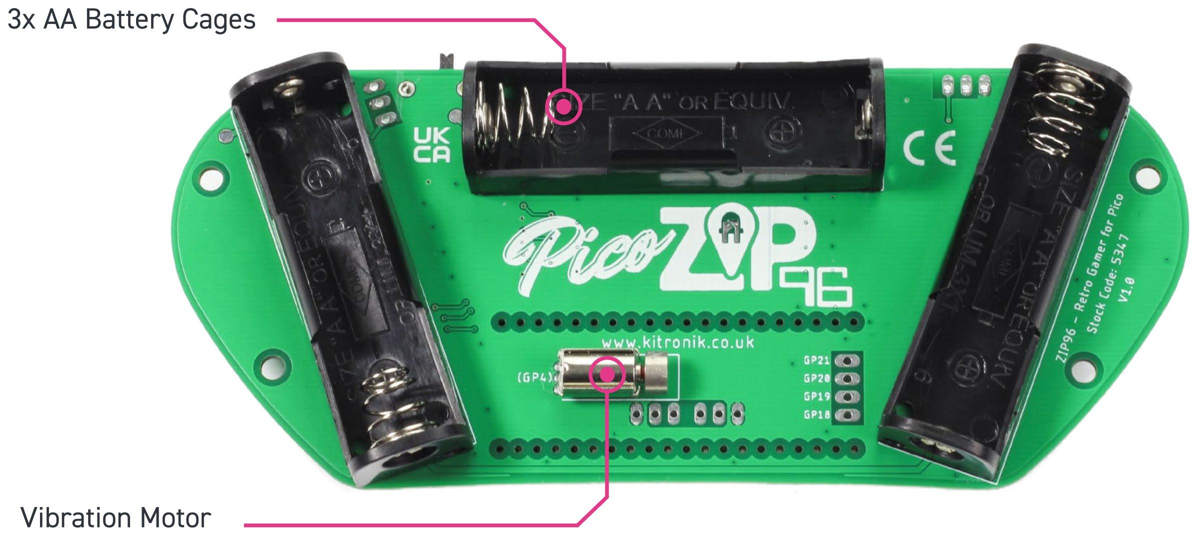
FULL LESSON CODE

The full code for each lesson can be found in the Lessons Code folder.

THE ZIP96 IS A PROGRAMMABLE RETRO GAMEPAD FOR THE RASPBERRY PI PICO.

It features 96 colour addressable LEDs arranged in a 12 x 8 display, a buzzer for audio feedback, a vibration motor for haptic feedback, and 6 input buttons. It also breaks out GP1, GP11, ADC1 and ADC2, along with a set of 3.3V and GND for each, to standard 0.1" footprints. GP18 to 21 are also broken out on a 0.1" footprint underneath the Pico. The Pico is connected via low profile 20-way pin sockets.





T: 0115 970 4243

W: www.kitronik.co.uk

E: support@kitronik.co.uk

kitronik.co.uk/twitter

kitronik.co.uk/youtube

kitronik.co.uk/facebook

kitronik.co.uk/instagram

Designed & manufactured in the UK by Kitronik

RoHS