Interfacing a matrix keypad with a Raspberry Pi 4 using its GPIO pins involves setting up a row-column scanning mechanism to detect key presses.

1. Hardware Connection:

* Connect the row pins of the matrix keypad to a set of GPIO pins on the Raspberry Pi 4.
* Connect the column pins of the matrix keypad to another set of GPIO pins on the Raspberry Pi 4.
* No external power supply is typically required for the keypad itself, as it's a simple array of switches.

2. Software Implementation (Python with RPi.GPIO):

* Import RPi.GPIO: Begin by importing the necessary library for GPIO control.

Python

import RPi.GPIO as GPIO

import time

* **Define Pin Layout:** Map the keypad's rows and columns to the chosen Raspberry Pi GPIO pins.

Python

ROW\_PINS = [2, 3, 4, 17] *# Example GPIO pins for rows*

COL\_PINS = [27, 22, 10, 9] *# Example GPIO pins for columns*

KEYPAD\_LAYOUT = [

['1', '2', '3', 'A'],

['4', '5', '6', 'B'],

['7', '8', '9', 'C'],

['\*', '0', '#', 'D']

]

* **GPIO Setup:**
  + Set the row pins as outputs and initially set them to a high or low state (depending on your scanning logic).
  + Set the column pins as inputs, often with internal pull-up or pull-down resistors enabled to provide a stable default state when no key is pressed.

Python

GPIO.setmode(GPIO.BCM) *# Use Broadcom pin-numbering scheme*

for pin in ROW\_PINS:

GPIO.setup(pin, GPIO.OUT)

GPIO.output(pin, GPIO.HIGH) *# Or LOW, depending on logic*

for pin in COL\_PINS:

GPIO.setup(pin, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP) # Or PUD\_DOWN

* **Scanning Logic:**
  + Iterate through each row. For the current row, set its corresponding GPIO pin to an active state (e.g., LOW if using pull-ups on columns).
  + Then, scan each column. If a key in that row and column is pressed, the column's input pin will change its state (e.g., go LOW if previously pulled HIGH).
  + Identify the pressed key by its row and column index and reference the KEYPAD\_LAYOUT array.
  + Reset the row pin to its inactive state before moving to the next row.
  + Include a small delay after detecting a press to debounce the input.

Python

def read\_keypad():

for r\_idx, row\_pin in enumerate(ROW\_PINS):

GPIO.output(row\_pin, GPIO.LOW) *# Activate current row (assuming pull-ups on columns)*

for c\_idx, col\_pin in enumerate(COL\_PINS):

if GPIO.input(col\_pin) == GPIO.LOW: *# Key pressed*

key = KEYPAD\_LAYOUT[r\_idx][c\_idx]

GPIO.output(row\_pin, GPIO.HIGH) *# Deactivate row*

return key

GPIO.output(row\_pin, GPIO.HIGH) *# Deactivate row*

return None

try:

while True:

key = read\_keypad()

if key:

print(f"Key pressed: {key}")

time.sleep(0.2) *# Debounce delay*

time.sleep(0.05) *# Short delay between scans*

except KeyboardInterrupt:

GPIO.cleanup() # Clean up GPIO settings on exit

This method allows the Raspberry Pi to continuously scan the keypad and detect individual key presses based on the row-column intersection.

For a Raspberry Pi 4, a row-column key switch array (commonly a 4x4 or 4x3 matrix keypad) can be wired to the GPIO header to read user input. This method uses fewer GPIO pins than wiring each button individually, and the Raspberry Pi's internal resistors can simplify the wiring.

**Matrix keypad wiring**

A standard 4x4 matrix keypad uses eight pins: four for the rows and four for the columns. You will connect these eight pins to eight of the Raspberry Pi's GPIO pins.

**Components needed**:

* **Raspberry Pi 4**
* **4x4 matrix keypad** (e.g., a 16-button membrane keypad)
* **Breadboard**
* **Jumper wires**

**Wiring steps**:

1. **Identify pins:** Determine which of your keypad's eight pins correspond to the four rows and four columns. This may require a multimeter and a continuity test.
2. **Connect rows:** Plug the four row pins of the keypad into a row on your breadboard.
3. **Connect columns:** Plug the four column pins of the keypad into a separate row on your breadboard.
4. **Connect to Pi:** Using jumper wires, connect the keypad pins on the breadboard to the Raspberry Pi's GPIO pins. Note the specific GPIO pins you choose for each row and column for use in your code. For a 4x4 keypad, a possible arrangement could be:
   1. **Keypad Row 1 → GPIO pin 4**
   2. **Keypad Row 2 → GPIO pin 17**
   3. **Keypad Row 3 → GPIO pin 27**
   4. **Keypad Row 4 → GPIO pin 22**
   5. **Keypad Col 1 → GPIO pin 5**
   6. **Keypad Col 2 → GPIO pin 6**
   7. **Keypad Col 3 → GPIO pin 13**
   8. **Keypad Col 4 → GPIO pin 19**
5. **No external resistors needed:** You do not need to add external pull-up or pull-down resistors, as the software library will configure the Raspberry Pi's internal resistors.

**How the matrix is scanned**

The Raspberry Pi detects button presses by "scanning" the rows and columns.

1. **Scanning procedure:** The code sets one row's GPIO pin to HIGH (3.3V) at a time, while keeping the other rows LOW.
2. **Checking columns:** After setting a row HIGH, the code checks the state of each of the column pins. If a button is pressed, it creates a connection between the currently HIGH row and its corresponding column, causing the column's voltage to go HIGH.
3. **Decoding the key:** By identifying which row was set HIGH and which column read HIGH, the program can determine exactly which key was pressed.

**Python code using pad4pi library**

The pad4pi library simplifies the process of reading from a matrix keypad by handling the row-column scanning logic for you.

**Installation:**

sh

sudo pip3 install pad4pi

Use code with caution.

**Python script (keypad.py):**

This example code uses the GPIO pin mapping from the wiring steps above.

To create a Python script to read the keypad, you can use the pad4pi library. First, import the necessary libraries: from pad4pi import rpi\_gpio and import time. Define your keypad layout as a list of lists, for example, KEYPAD = [["1", "2", "3", "A"], ["4", "5", "6", "B"], ["7", "8", "9", "C"], ["\*", "0", "#", "D"]]. Define the BCM GPIO pins used for the rows (ROW\_PINS = [4, 17, 27, 22]) and columns (COL\_PINS = [5, 6, 13, 19]).

Next, create a keypad object using factory = rpi\_gpio.KeypadFactory() and keypad = factory.create\_keypad(keypad=KEYPAD, row\_pins=ROW\_PINS, col\_pins=COL\_PINS). Define a function to handle key presses, for example, def printKey(key): print("Key Pressed:", key). Register this function as the key press handler using keypad.registerKeyPressHandler(printKey).

Finally, include a loop to keep the program running and detecting key presses, and include a try...except KeyboardInterrupt...finally block to allow you to exit the script with Ctrl+C and ensure GPIO cleanup.

You can find the full script in the referenced web document.