09a ZRX543 4x4 Keypad Introduction

Nicholas Bruzzese

January 6, 2025

What is a 4x4 Keypad?

The ZRX543 4x4 keypad is a matrix-style keypad with 16 buttons arranged in a 4x4 grid. It uses a combination of rows and columns to detect button presses. Each button connects one row to one column, enabling the identification of the pressed button.

Key Features

- **Compact Design**: 16 keys arranged in a small footprint.
- **Matrix Configuration**: Reduces the number of GPIO pins required.
- **Versatility**: Commonly used in security systems, robotics, and more.

Hardware Requirements

- Raspberry Pi (any model with GPIO pins).
- ZRX543 4x4 keypad.
- Jumper wires for connections.

Wiring Diagram

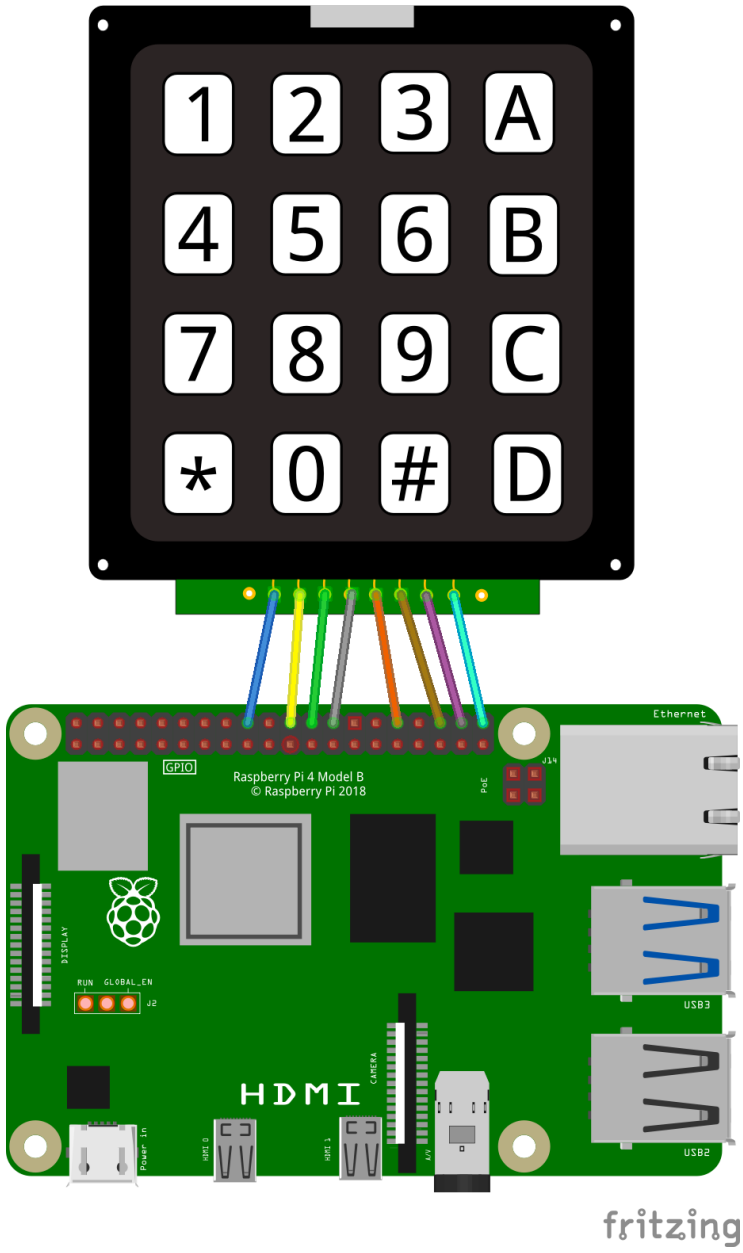

Figure 1: Wiring Diagram

Connections

The ZRX543 4x4 keypad has 8 pins: 4 rows (R1 to R4) and 4 columns (C1 to C4). These need to be connected to the GPIO pins of the Raspberry Pi. The connections are as follows:

Keypad Pin	GPIO Pin
R1 (A8)	GPIO 24
R2 (A7)	GPIO 25
R3 (A6)	GPIO 8
R4 (A5)	GPIO 7
C1 (A4)	GPIO 12
C2 (A3)	GPIO 16
C3 (A2)	GPIO 20
C4 (A1)	GPIO 21

Ensure the keypad is connected securely, and double-check the pin mappings before proceeding.

Code Explanation

The Python script below scans the keypad, detects button presses, and prints the corresponding key.

1. GPIO Initialization

The rows are configured as output pins, and the columns are configured as input pins with internal pull-down resistors to detect button presses.

```
import RPi.GPIO as GPIO
import time

# GPIO pin definitions
L1 = 24 # Row 1
L2 = 25 # Row 2
L3 = 8  # Row 3
L4 = 7  # Row 4
C1 = 12 # Column 1
C2 = 16 # Column 2
C3 = 20 # Column 3
C4 = 21 # Column 4

# GPIO setup
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup([L1, L2, L3, L4], GPIO.OUT)
GPIO.setup([C1, C2, C3, C4], GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
```

2. Scanning the Keypad

The `readLine` function activates one row at a time and checks all columns for a signal. If a signal is detected, the corresponding key is printed.

```
def readLine(line, characters):
GPIO.output(line, GPIO.HIGH)
if GPIO.input(C1) == 1:
print(characters[0])
if GPIO.input(C2) == 1:
print(characters[1])
if GPIO.input(C3) == 1:
print(characters[2])
if GPIO.input(C4) == 1:
print(characters[3])
GPIO.output(line, GPIO.LOW)
```

3. Main Loop

The main loop continuously scans the keypad by calling `readLine` for each row. The detected key is printed to the console.

```
try:
while True:
readLine(L1, ["1", "2", "3", "A"])
readLine(L2, ["4", "5", "6", "B"])
readLine(L3, ["7", "8", "9", "C"])
readLine(L4, ["*", "0", "#", "D"])
time.sleep(0.1)
except KeyboardInterrupt:
print("\nApplication stopped!")
```

Running the Project

1. Save the code as `keypad.py`.
2. Run the script:

```
python3 keypad.py
```

3. Press buttons on the keypad. The corresponding key will be printed in the terminal.

Experiment Ideas

- **Key Combination Detection**: Add logic to detect specific key sequences, such as a password.
- **Interactive Menus**: Combine the keypad with an LCD to create a basic user interface.
- **Custom Actions**: Trigger GPIO actions (e.g., turning on an LED) based on specific keys.

Applications

- Security systems (e.g., digital locks).
- Menu-driven systems for user interaction.
- Custom input devices for Raspberry Pi projects.

With this project, you've built a solid foundation for using keypads with Raspberry Pi. Want to take it further? Try integrating the keypad into a more complex system like a smart home controller or a robotics interface!