SIMON FRASER UNIVERSITY

CMPT 433

Embedded Systems

How To Guide:

Connecting MCP23017 Port Expanders & Reed Switches

Introduction

Many embedded systems projects require reading input from multiple sensors or switches. In our CMPT 433 project—a smart chessboard—we needed to detect the physical presence of chess pieces on all 64 squares of the board. To achieve this, we installed a reed switch underneath every square. Reed switches are small magnetic sensors that close their circuit when a magnet is nearby, making them ideal for detecting the presence of magnetized chess pieces.

However, this introduced a challenge: the BeagleY-AI only supports a limited number of GPIO (General Purpose Input/Output) pins. With 64 reed switches to monitor, we far exceeded the available GPIOs. Connecting each reed switch directly to a pin would have required custom circuitry or complex multiplexing, which is error-prone and hard to debug.

Why this Guide?

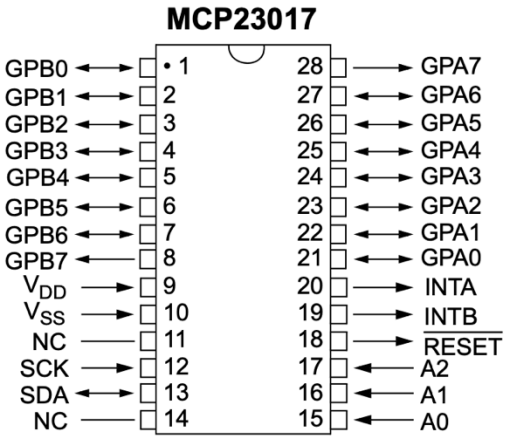
To overcome this, we used the MCP23017, a 16-bit I/O expander that communicates over the I2C bus. By using just two pins (SDA and SCL), the MCP23017 gives access to 16 digital input/output pins per chip. Even better, you can chain up to 8 MCP23017 chips together on the same I2C bus, allowing for a total of 128 I/O pins—more than enough for our chessboard application.

The MCP23017 handles digital logic cleanly, supports internal pull-up resistors (essential for proper switch state reading), and allows us to poll for sensor states efficiently. This approach not only simplifies hardware wiring but also leads to modular and scalable design, applicable to any embedded project with large I/O requirements.

Hardware Requirements

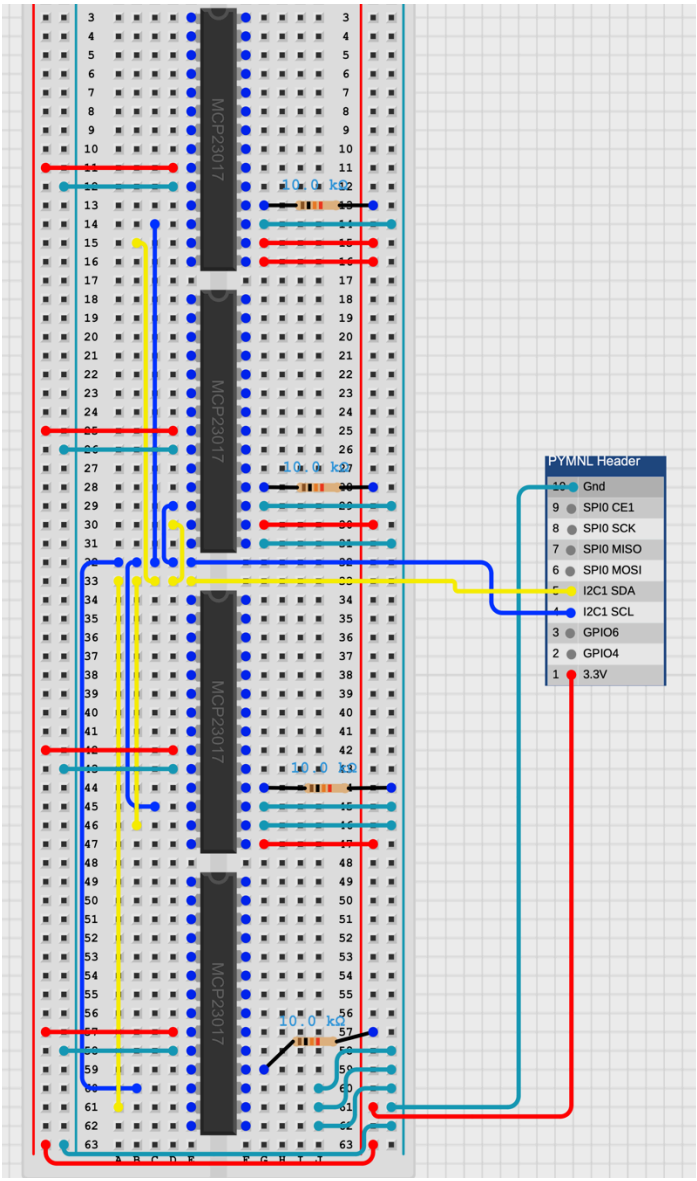
- MCP23017 I/O Expander Chip
- Reed Switches
- 10k ohm Resistor
- Jumper Wires
- Breadboard
- BeagleY AI with Zen Hat

Circuit Connection



Connecting I/O Expanders to BeagleY-AI

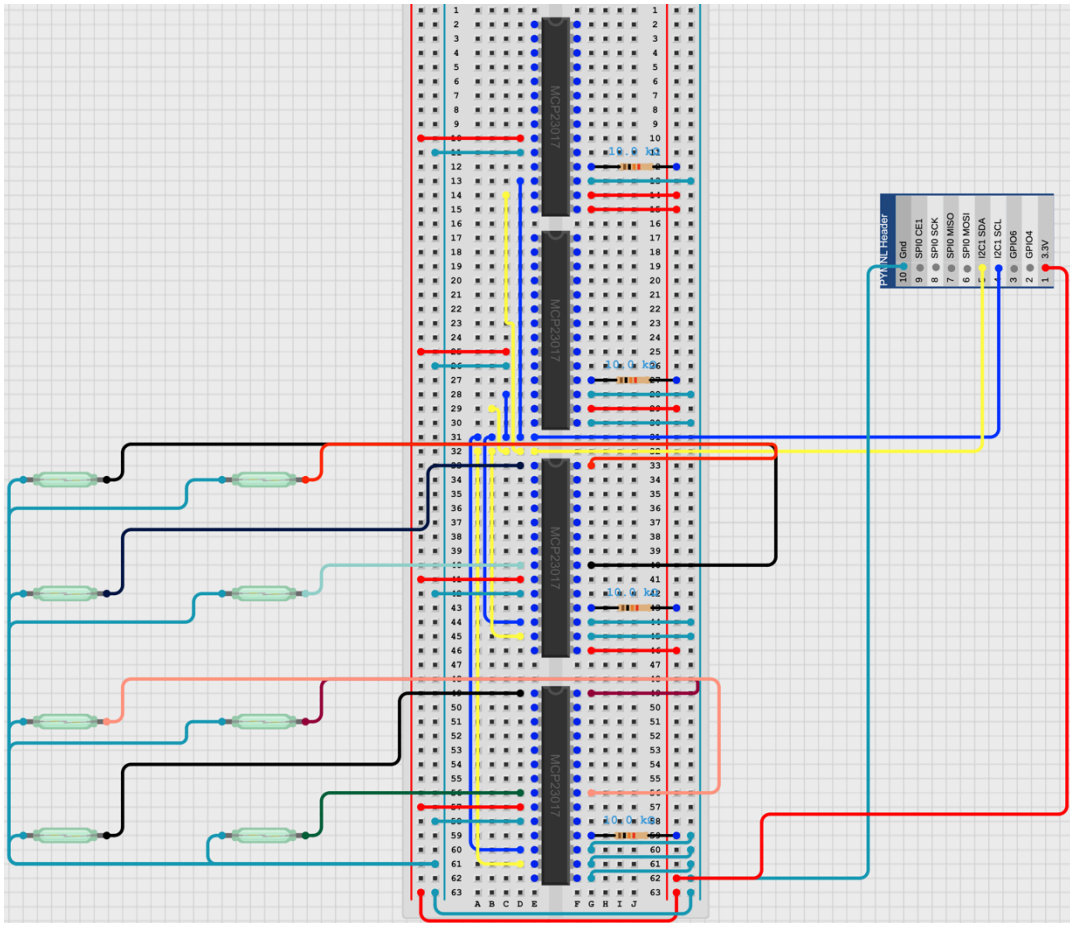
- **Power:**
  - PYMNL Pin 1 (3.3v) -> V_{DD} (Pin 9)
  - PYMNL Pin 10 (Gnd) -> V_{SS} (Pin 10)
  - MCP Pin 18 -> 10k Ω -> Pwr (3.3v)
- **I²C Lines:**
  - PYMNL Pin 4 (SLC) -> SCK (Pin 12)
  - PYMNL Pin 5 (SDA) -> SDA (Pin 13)
- **Addressing:**
  - A0, A1, A2 (Pin 15, 16, 17) to Gnd or Pwr to set I²C address (eg: all low = 0x20)



Addressing Multiple MCP23017 Chips Through Single I2C Bus

The MCP23017 supports hardware-based addressing to allow multiple chips to coexist on the same I2C bus. Each MCP23017 has three address pins: A0, A1, and A2, which determine the lower bits of the device's 7-bit I2C address. By configuring these pins as either HIGH (connected to 3.3V) or LOW (connected to GND), you can assign each MCP23017 a unique address ranging from 0x20 to 0x27. This allows you to connect up to 8 MCP23017 chips on a single I2C bus, expanding your digital I/O capabilities to 128 pins total without any additional communication wires. Just make sure each chip has a unique combination of address pin settings to avoid conflicts on the bus.

Connecting Reed Switches to MCP23017 chip



The MCP23017 provides 16 general-purpose I/O (GPIO) pins, split into two 8-bit ports: GPIOA and GPIOB. GPIOA pins (GPA0–GPA7) correspond to pins 21 to 28 on the MCP23017, while GPIOB pins (GPB0–GPB7) correspond to pins 1 to 8. Each of these pins can be configured as an input or output. To connect reed switches, wire one terminal of each switch to a GPIO pin and the other terminal to ground. When the switch closes (e.g., when a magnet is nearby), the GPIO pin is pulled low, which the MCP can detect. Internal pull-up resistors should be enabled in software to ensure reliable detection when the switches are open.

Accessing I/O Expanders Through I2C Via Command Line

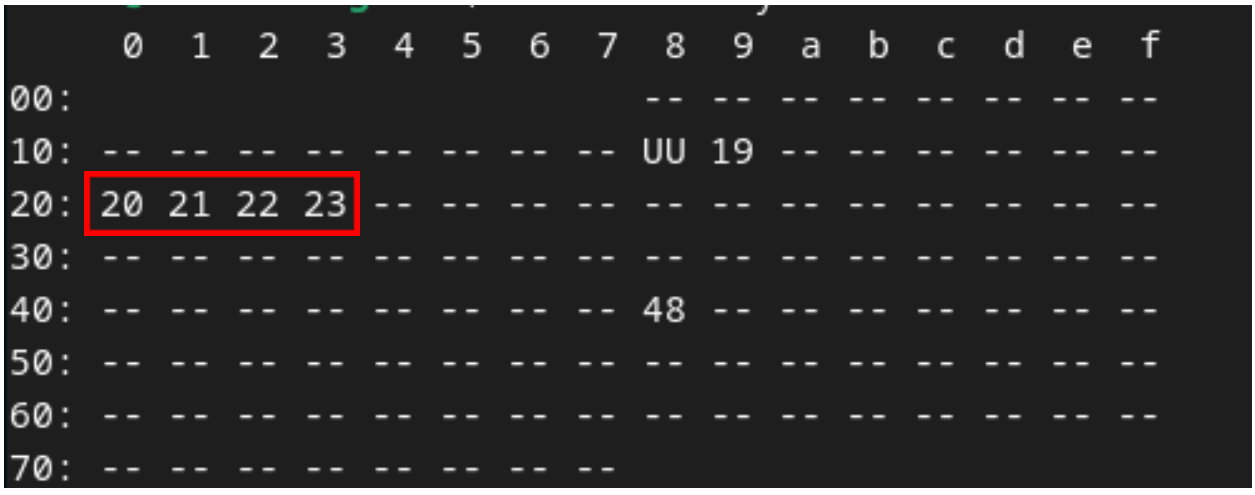
The I2C Pins on PYMNL header on the Zen Hat are connected to I2C bus 1 on BeagleY-AI. This section walks through accessing the I/O pins on MCP23017 Expanders using I2C tools on command line.

1. Install the I2C tools (if not already installed)
   ```
   (byai)$ sudo apt-get install i2c-tools
   ```

2. Display which I2C buses Linux currently has enabled
   ```
   (byai)$ i2cdetect -l
   ```

i2c-1 i2c	OMAP I2C adapter	I2C adapter
i2c-2 i2c	OMAP I2C adapter	I2C adapter
i2c-3 i2c	OMAP I2C adapter	I2C adapter
i2c-4 i2c	OMAP I2C adapter	I2C adapter
i2c-5 i2c	OMAP I2C adapter	I2C adapter

3. Display I2C devices on chosen I2C bus (here I2C1)
   ```
   (byai)$ i2cdetect -y -r 1
   ```



The MCP23017 chips are detected on 0x20, 0x21, 0x22 and 0x23 on I2C Bus 1.

Configuring and Reading Reed Switches

The reed switches could be configured and read using the I2C module. The “i2c_fd” array in the code below has addresses of the MCP23017 chips on I2C bus 1. (0x20, 0x21, ….). Make sure to initialize the i2c bus before running the code.

1. Setting I/O direction for GPOI pins

```
#define IODIRA    0x00  // I/O Direction Register for Bank A
#define IODIRB    0x01  // I/O Direction Register for Bank B
```

These are the addresses of the resistors for 2 Banks on the MCP23017 chip.

```
// Set all PORT A & PORT B pins as input
write_i2c_reg16(i2c_fd[i], IODIRA, 0xFF);
usleep(5000);

write_i2c_reg16(i2c_fd[i], IODIRB, 0xFF);
```

2. Enable Pull up resistors for the I/O pins

```
#define GPPUA     0x0C  // Pull-up Enable Register for Bank A
#define GPPUB     0x0D  // Pull-up Enable Register for Bank B
```

```
// Enable pull-up resistors on all PORT A & PORT B
write_i2c_reg16(i2c_fd[i], GPPUA, 0xFF);

usleep(5000);
write_i2c_reg16(i2c_fd[i], GPPUB, 0xFF);
```

3. Reading GPIO Pins

The portExtender_readAllPins() function reads the states of 64 reed switches connected through four MCP23017 I/O expanders on the I2C bus. Each MCP23017 provides 16 GPIO pins (8 on port A and 8 on port B), and with four chips, the function handles a total of 64 inputs. It iterates over each of the four MCP chips and reads the GPIOA and GPIOB registers using read_i2c_reg16, which returns an 8-bit value where each bit represents the state of one I/O pin. The function uses bit masking (1 << j) to check each bit and interprets a value of 0 as piece present (reed switch activated) and 1 as no piece (reed switch open).

The readings are stored in a state[8][8] matrix that represents the chessboard layout, where each row corresponds to a rank and each column to a file. Specifically, GPIOA pins are mapped to even-numbered ranks (i * 2), and GPIOB pins are mapped to the next odd-numbered rank (i * 2 + 1). This organization mirrors the physical layout of the board, making it intuitive to process the board state in software.

```
void portExtender_readAllPins(uint8_t state[8][8]){
    if (!is_initialized) {
        return;
    }

    for (int i = 0; i < 4; i++) {
        uint16_t portA = read_i2c_reg16(i2c_fd[i], GPIOA);
        usleep(5000);
        uint16_t portB = read_i2c_reg16(i2c_fd[i], GPIOB);

        for (int j = 0; j < 8; j++) {


            int rankA = i * 2;
            int file = j;
            state[rankA][file] = (portA & (1 << j)) ? 0 : 1;

            int rankB = i * 2 + 1;
            state[rankB][file] = (portB & (1 << j)) ? 0 : 1;
        }
    }
}
```