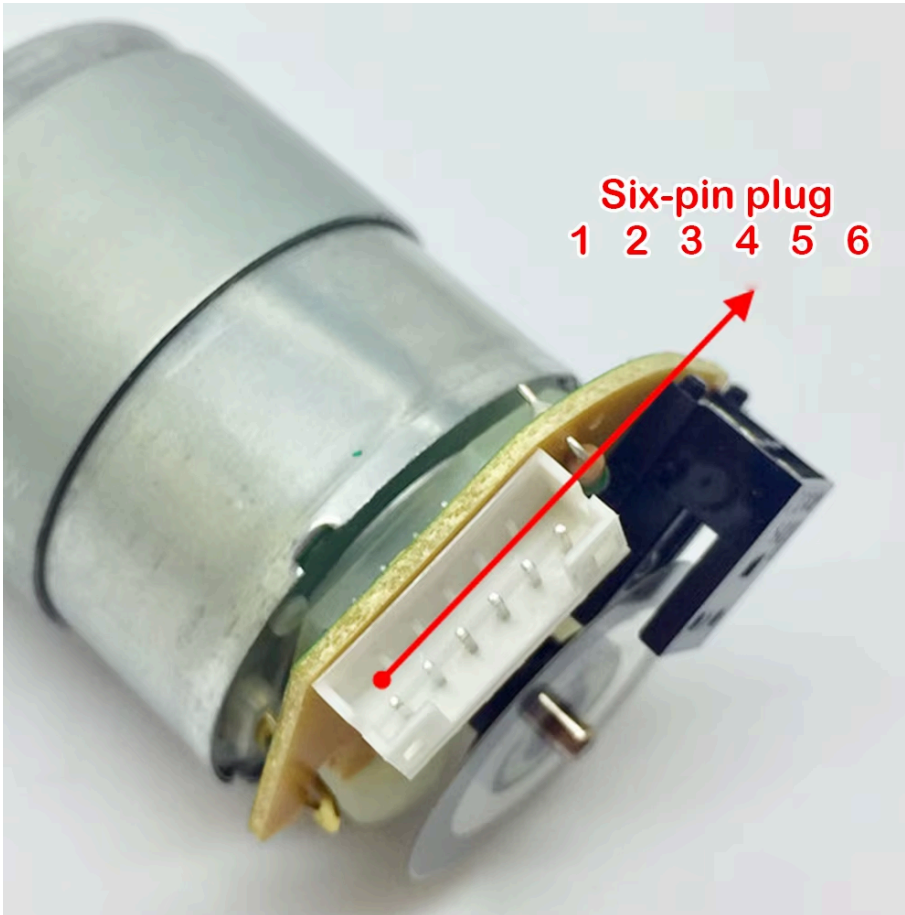


The motor pinout



Don't trust the colors of the cable!

1. Motor line -
2. Motor line +
3. Encoder VCC (3.3V)
4. Encoder GND
5. Encoder output A phase
6. Encoder output B phase

The encoder has **334 lines**

The diagram illustrates the pinout of the Raspberry Pi Pico 2, a compact microcontroller board. The 40-pin GPIO header is organized as follows:

- Pins 1-20 (Left Side):**
 - 1: UART0 TX (Purple), I2C0 SDA (Blue), SPI0 RX (Pink), GP0 (Green)
 - 2: UART0 RX (Purple), I2C0 SCL (Blue), SPI0 CSn (Pink), GP1 (Green)
 - 3: GND (Black)
 - 4: I2C1 SDA (Blue), SPI0 SCK (Pink), GP2 (Green)
 - 5: I2C1 SCL (Blue), SPI0 TX (Pink), GP3 (Green)
 - 6: UART1 TX (Purple), I2C0 SDA (Blue), SPI0 RX (Pink), GP4 (Green)
 - 7: UART1 RX (Purple), I2C0 SCL (Blue), SPI0 CSn (Pink), GP5 (Green)
 - 8: GND (Black)
 - 9: I2C1 SDA (Blue), SPI0 SCK (Pink), GP6 (Green)
 - 10: I2C1 SCL (Blue), SPI0 TX (Pink), GP7 (Green)
 - 11: UART1 TX (Purple), I2C0 SDA (Blue), SPI1 RX (Pink), GP8 (Green)
 - 12: UART1 RX (Purple), I2C0 SCL (Blue), SPI1 CSn (Pink), GP9 (Green)
 - 13: GND (Black)
 - 14: I2C1 SDA (Blue), SPI1 SCK (Pink), GP10 (Green)
 - 15: I2C1 SCL (Blue), SPI1 TX (Pink), GP11 (Green)
 - 16: UART0 TX (Purple), I2C0 SDA (Blue), SPI1 RX (Pink), GP12 (Green)
 - 17: UART0 RX (Purple), I2C0 SCL (Blue), SPI1 CSn (Pink), GP13 (Green)
 - 18: GND (Black)
 - 19: I2C1 SDA (Blue), SPI1 SCK (Pink), GP14 (Green)
 - 20: I2C1 SCL (Blue), SPI1 TX (Pink), GP15 (Green)
- Pins 21-40 (Right Side):**
 - 21: GP16 (Green), SPI0 RX (Pink), I2C0 SDA (Blue), UART0 TX (Purple)
 - 22: GP17 (Green), SPI0 CSn (Pink), I2C0 SCL (Blue), UART0 RX (Purple)
 - 23: GND (Black)
 - 24: GP18 (Green), SPI0 SCK (Pink), I2C1 SDA (Blue)
 - 25: GP19 (Green), SPI0 TX (Pink), I2C1 SCL (Blue)
 - 26: GP20 (Green), I2C0 SDA (Blue)
 - 27: GP21 (Green), I2C0 SCL (Blue)
 - 28: GND (Black)
 - 29: GP22 (Green)
 - 30: RUN (Pink)
 - 31: GP26 (Green), ADC0 (Dark Green), I2C1 SDA (Blue)
 - 32: GP27 (Green), ADC1 (Dark Green), I2C1 SCL (Blue)
 - 33: GND (Black), AGND (Dark Green)
 - 34: GP28 (Green), ADC2 (Dark Green)
 - 35: ADC_VREF (Dark Green)
 - 36: 3V3(OUT) (Red)
 - 37: 3V3_EN (Pink)
 - 38: GND (Black)
 - 39: VSYS (Red)
 - 40: VBUS (Red)

Legend:

- Power (Red)
- Ground (Black)
- UART / UART (default) (Purple)
- GPIO, PIO, and PWM (Green)
- ADC (Dark Green)
- SPI / SPI (default) (Pink)
- I2C / I2C (default) (Blue)
- System Control (Light Pink)
- Debugging (Orange)

The board also features a USB-C port, a BOOTSEL button, a LED (GP25), and a DEBUG header with SWCLK, GND, and SWDIO pins.

The Raspberry Pi Pico 2 breakout board contains an [RP2350](#) microcontroller, with 2 ARM [M33 cores](#) (FPU!) at 150MHz, 512k RAM, and 4M flash. Or it can use [2 RISC-V cores](#). It uses a MSD bootloader, so when you plug it into your computer with the BOOTSEL button pressed, it will appear as a thumb drive named RP2350. Compiled code has the extension .uf2. Drag a .uf2 file onto the RP-RP2 drive and the Pico will program itself.

Reading the encoder over USB

Program the Pico with `pico2_encoder_to_usb.uf2`. Connect the encoder phase A to the Pico GP14, and encoder phase B to the Pico GP15, and the encoder VCC 3.3V to the Pico 3V3(OUT) pin and the encoder GND to the pico GND. The Pico will enumerate as a USB device (on Windows, something like COM4, on Mac/Linux, something like `/dev/tty.usbmodem1101`). Open the port in Putty or screen, and the Pico will print the quadrature count and velocity at 10Hz. For 334 encoder lines lines, the Pico will record $334 \times 4 = 1336$ counts per revolution.

Reading the encoder over serial

Program the Pico with `pico2_encoder_to_serial.uf2`.

Keep the encoder A and B pins in the Pico GP14 and GP15 pins.

Remove the Pico USB cable. Connect the Pico GND to the PIC gnd, and connect the Pico 3V3(OUT) to the PIC 3.3V pin so that the PIC board is powering the Pico.

The Pico is programmed to communicate over UART on pins GP0/UART0TX and GP1/UART0RX with a baud of 230400.

Connect the Pico TX to the PIC U2RX (pin B1), and the Pico RX to the PIC U2TX (pin B0).

Add `encoder.c` and `encoder.h` to your PIC project folder. Call `UART2_Startup()` in `main()`.

When you print 'a' from the PIC to the Pico, the Pico will print back the encoder count as an integer followed by a newline.

When you print 'b' from the PIC to the Pico, the Pico will reset the encoder count to 0 and will not reply.

The PIC will use the UART2 RX interrupt to read every letter that comes from the PICO. When the PIC gets a newline, it will `sscanf` the encoder count out of the character array, and set a flag variable to 1.

After using `writeUART2("a")`, wait for the `get_encoder_flag()` function to return a 1, and then use the `set_encoder_flag(0)` function to clear the flag variable, and access the encoder count using the `get_encoder_count()` function.

```
// read encoder count
WriteUART2("a");
while(!get_encoder_flag()){
    set_encoder_flag(0);
}
char m[50];
int p = get_encoder_count();
sprintf(m, "%d\r\n", p);
NU32DIP_WriteUART1(m);
```