Serial Expansion HAT User Manual

OVERVIEW

Serial Expansion HAT for Raspberry Pi, I2C Interface, Provides 2-ch UART and 8 GPIOs

FEATURES

- Raspberry Pi connectivity, compatible with Raspberry Pi Zero/Zero W/Zero WH/2B/3B/3B+
- Onboard SC16IS752, expands 2-ch UART and 8 programmable GPIO through I2C, no extra pin required
- It is stackable up to 16 modules by setting the address jumper, that means up to 32-ch UART
- Onboard multi LEDs for indicating the UART working status
- Reserved I2C control pins, allows to work with other control boards
- Comes with development resources and manual (examples in C and python)

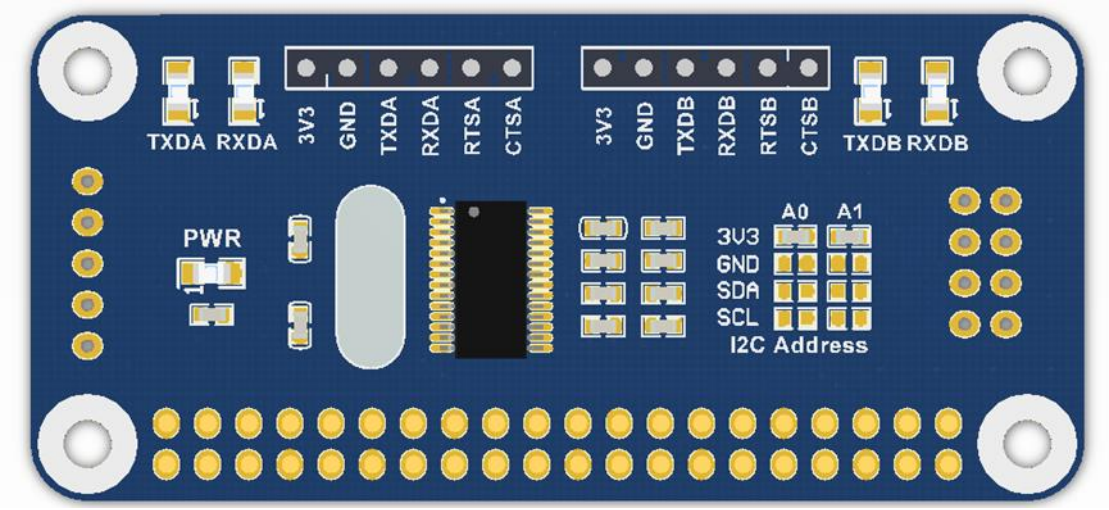
SPECIFICATION

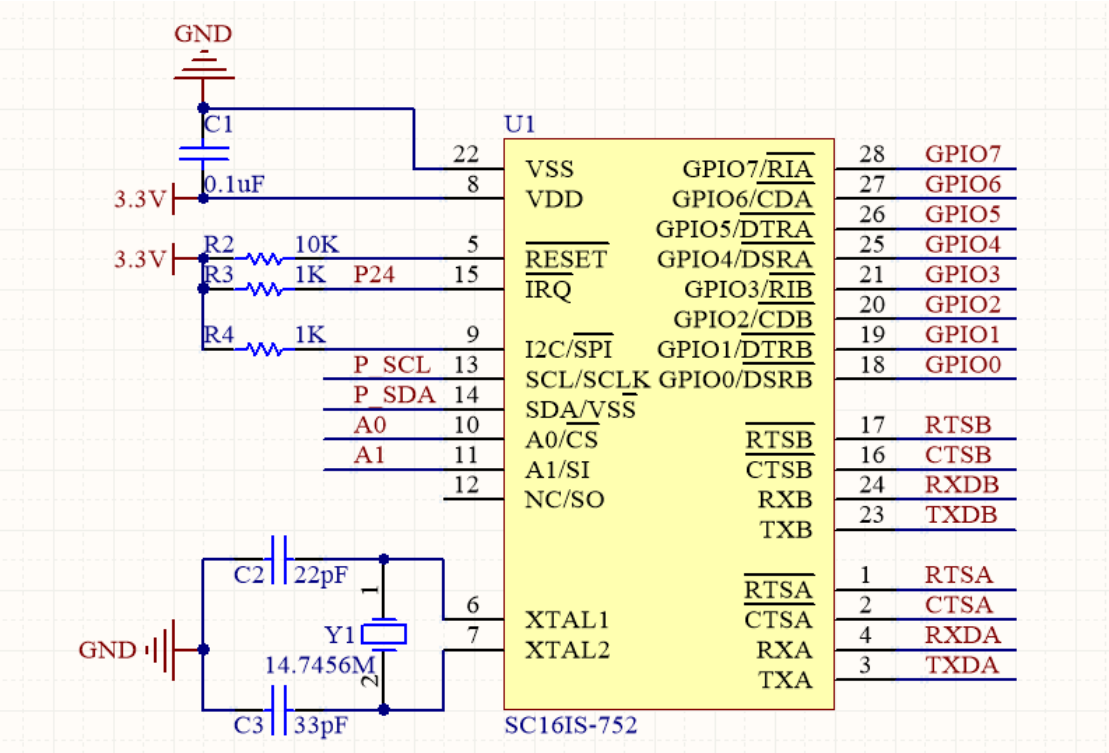
- Operating voltage: 3.3V
- Expansion chip: SC16IS752
- Control interface: I2C
- Dimension: 65mm x 30mm
- Mounting hole size: 3.0mm

CONTENT

HARDWARE

The SC16IS752 is an I2C-bus/SPI bus interface to a dual-channel high performance UART. It also provides the application with 8 additional programmable I/O pins. This module uses I2C interface by default, and its device address is hardware configurable by A0 and A1.





PINOUT

PIN	Description
3V3	3.3V
GND	Ground
TXDA	Transmit end of Channel A
RXDA	Receive end of Channel A
RTSA	Request to send of Channel A
CTSA	Clear to send of Channel A
TXDB	Transmit end of Channel B
RXDB	Receive end of Channel B
RTSB	Request to send of Channel B
CTSB	Clear to send of Channel B

LED

PWR: Power indicator

TXDA: Channel A transmit indicator

RXDA: Channel A receive indicator

TXDB: Channel B transmit indicator

RXDB: Channel B receive indicator

I2C DEVICE ADDRESS SETTING

I2C device address can be configured by changing status of A0 and A1, that is welding 0R resister to them according to this table:

Table 32. SC16IS752/SC16IS762 address map

A1	A0	SC16IS752/SC16IS762 I²C address (hex)[1]
V_{DD}	V_{DD}	0x90 (1001 000X)
V_{DD}	V_{SS}	0x92 (1001 001X)
V_{DD}	SCL	0x94 (1001 010X)
V_{DD}	SDA	0x96 (1001 011X)
V_{SS}	V_{DD}	0x98 (1001 100X)
V_{SS}	V_{SS}	0x9A (1001 101X)
V_{SS}	SCL	0x9C (1001 110X)
V_{SS}	SDA	0x9E (1001 111X)
SCL	V_{DD}	0xA0 (1010 000X)
SCL	V_{SS}	0xA2 (1010 001X)
SCL	SCL	0xA4 (1010 010X)
SCL	SDA	0xA6 (1010 011X)
SDA	V_{DD}	0xA8 (1010 100X)
SDA	V_{SS}	0xAA (1010 101X)
SDA	SCL	0xAC (1010 110X)
SDA	SDA	0xAE (1010 111X)

[1] X = logic 0 for write cycle; X = logic 1 for read cycle.

For details, please refer to datasheet: Page39

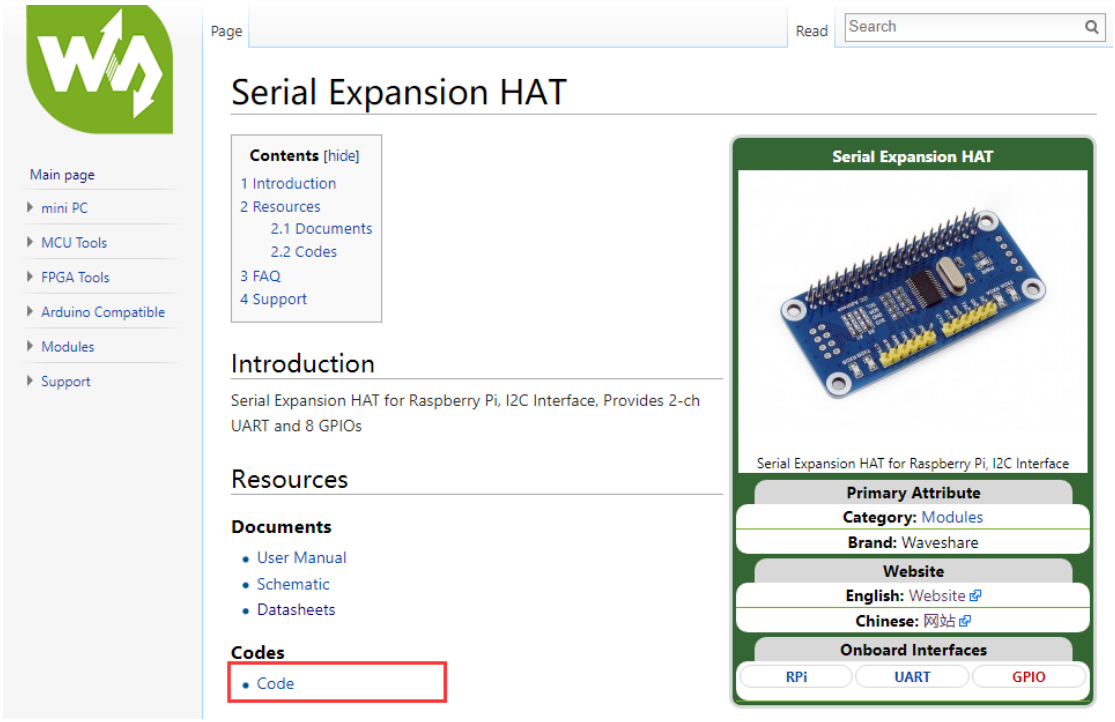
The I2C address in table are 8bits, however, the actual address is 7bits, you need to right-shift one bit to get the actual I2C address. For example, if you connect A1 and A0 to Vdd, the address of module is 0x90 according to the table, to get the actual address you need to right-shift the data from 1001 000X to 100 1000, that is 0x48.

【Note】This module A0 and A1 are default welded to 3.3V, with I2C address 0x48

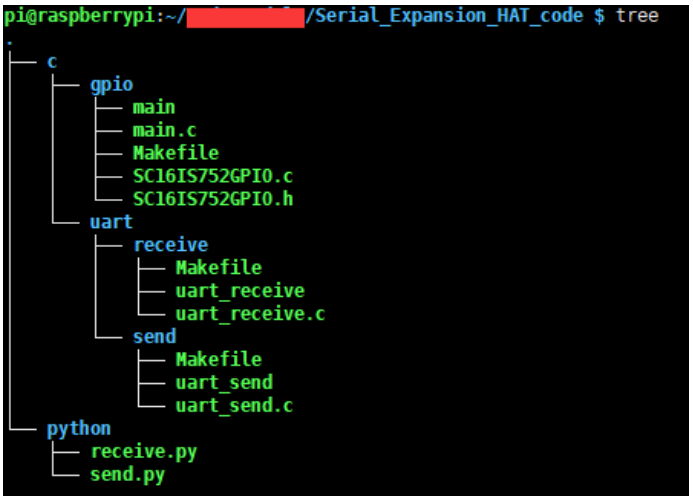
HOW TO USE

DOWNLOAD DEMO CODES

Visit WaveShare Wiki, search with key words “Serial Expansion HAT”, open it and download demo codes:



Extract and copy folders to Raspberry Pi.



LIBRARIES INSTALLATION (REQUIRE NETWORK)

1 Install wiringPi

1.1 Open Terminal of Raspbian(Ctrl+T), clone wiringPi from github

```
git clone git://git.drogon.net/wiringPi
```

1.2 Install

```
cd wiringPi

./build
```

2 Install python libraries

2.1 Install python-dev

```
sudo apt-get install python-dev
```

2.2 Install RPi.GPIO

```
sudo apt-get install python-rpi.gpio
```

2.3 Install smbus, which is I2C interfaces library

```
sudo apt-get install python-smbus
```

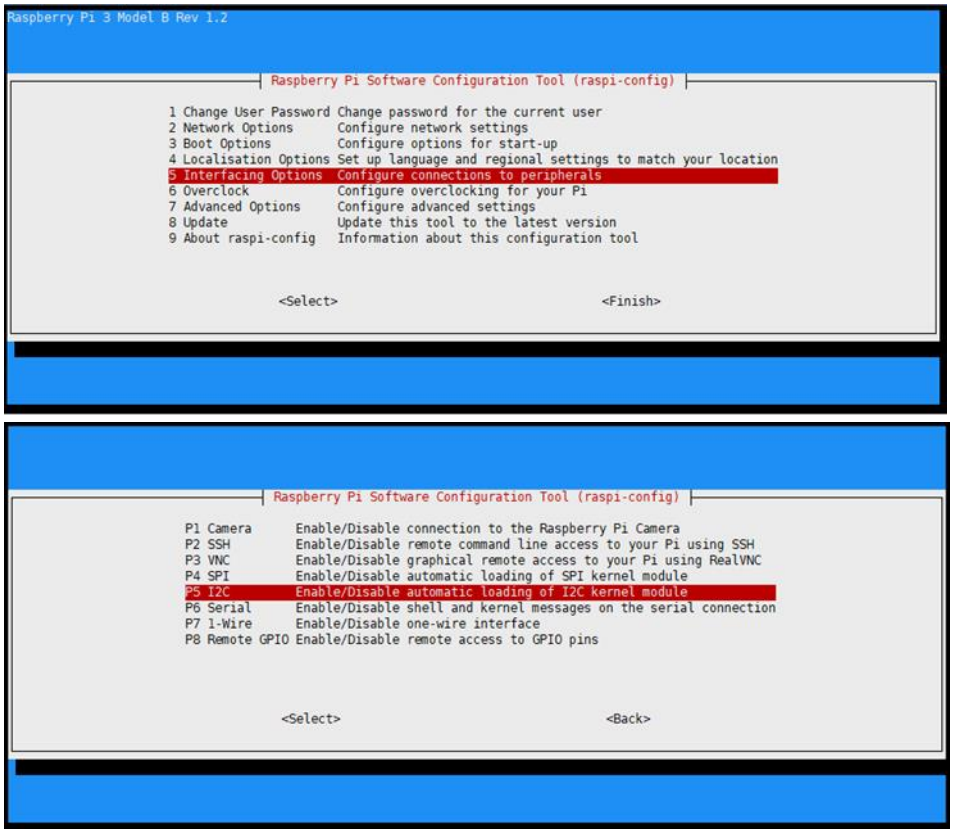
2.4 Install spidev, which is SPI interfaces library

```
sudo apt-get install python-spidev
```

ENABLE I2C INTERFACE

1 Execute command: **sudo raspi-config**

2 Choose: Interfacing Options->I2C->Yes



3 Append this line to end of /boot/config.txt file: **sudo nano /boot/config.txt**

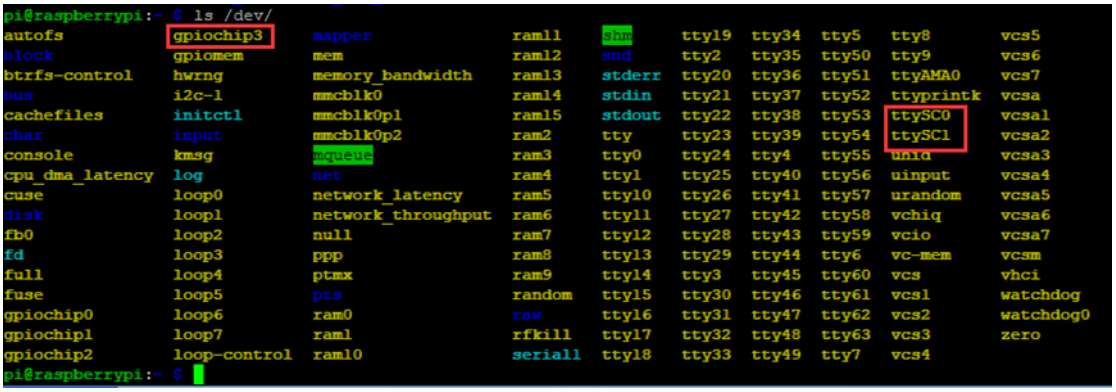
```
dtoverlay=sc16is752-i2c,int_pin=24,addr=0x48

# addr is different according to status of A0/A1, default 0X48
```

4 reboot

```
sudo reboot
```

5 After rebooting, you can execute command: ls /dev to check if SC16IS752 has been enabled to kernel.

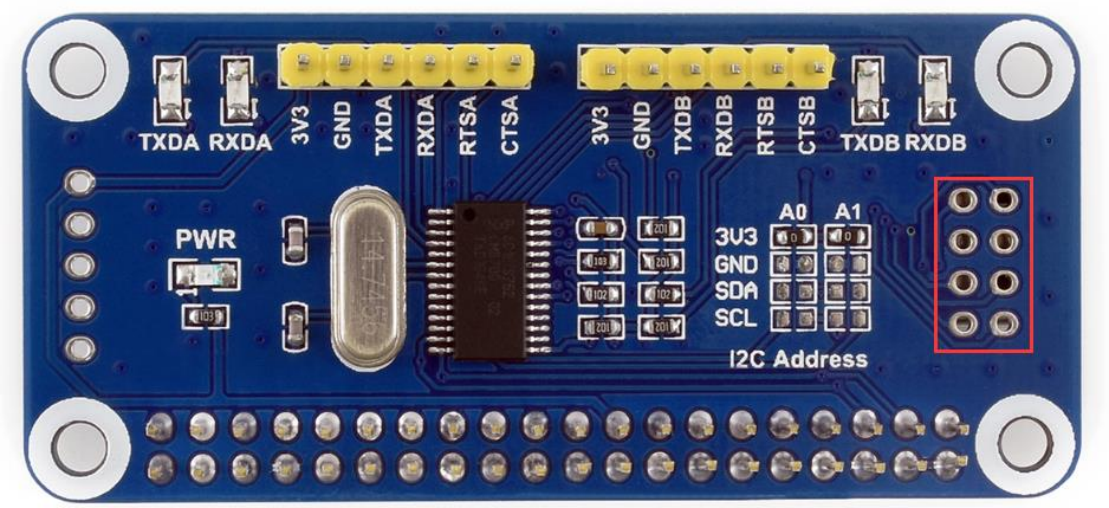


DEMO CODES

We provide demo codes for this module, based on C codes and python.

C/GPIO/

There are 8 GPIO which are expanded. You can connect LEDs to these GPIOs when testing.



1. Open the folder of the demo code

```
cd Serial_Expansion_HAT_code/c/gpio/

#change the path of you didn’ t put the code in /home/pi
```

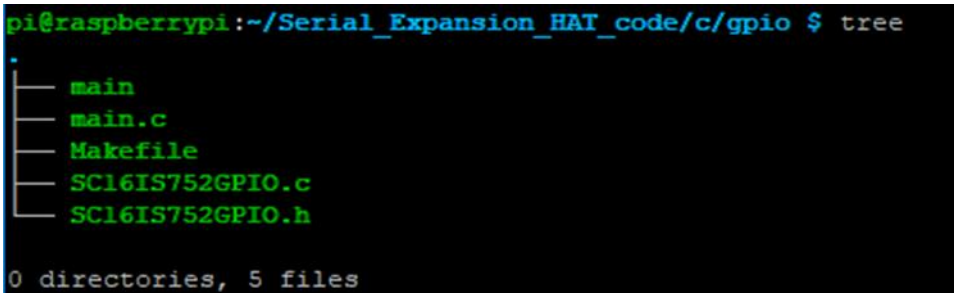
2. Compile and run the code

```
make

sudo ./main
```

3. After running the demo code, code will light on/off 8 LEDs one by one.

FILES



main.c: main function

SC16IS752GPIO.c(.h): functions control IO

Makefile: Codes compilation

CODES ANALYSIS

Functions in SC6IS752GPIO.c:

int GPIOExport(int Pin): Export GPIO

int GPIOUnexport(int Pin): Unexport GPIO

int GPIODirection(int Pin, int Dir): Set direction of GPIO

int GPIORead(int Pin): Read value of GPIO

int GPIOWrite(int Pin, int value), Write value to GPIO

Files used by these functions are all in /sys/class/gpio, and according to use guide, the GPIO generated by SC16IS752 is coded 504. So, the codes of 8 GPIO are from 504 to 511.

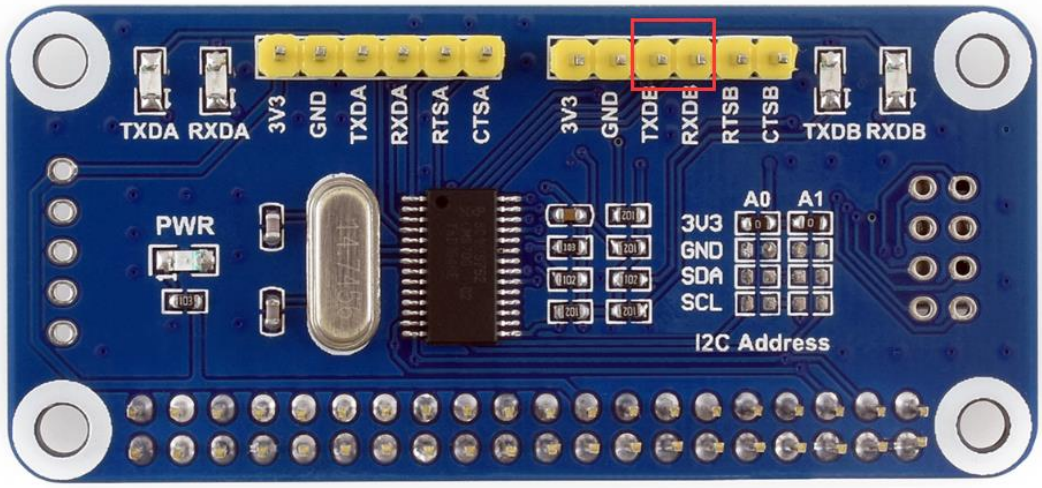
Functions in main.c:

GPIO_Init(): Initialize 8 GPIO.

GPIO_Exit(): Unexport GPIOs

C/UART/

To test UART demo code, you need to use two Serial Expansion HAT and two Raspberry Pi, one is set as receiver and another is sender. You can also connect with serial module to PC if you have no other Raspberry Pi and Expansion HAT. Connect RXB of Serial Expansion HAT to TXB/TX of other module, and TXB of Serial Expansion HAT to RXB/RX of another module.



1. Open the folder of code

```
cd Serial_Expansion_HAT_code/c/uart/

#change the path of you didn’ t put the code in /home/pi
```

2. Two codes, receiver and sender, choose the one you want to run

```
cd receive/

# cd send/
```

3. Compile and run the code

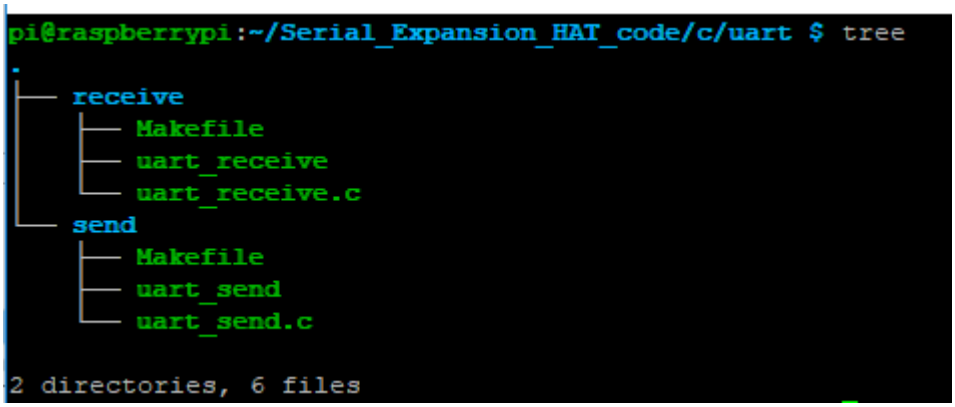
```
make

sudo ./uart_receive

# sudo ./uart_send
```

4. After running the code. Receiver: keep receiving and print data received; Sender: send characters

FILES



/receive:

Makefile: Code compilation, execute command make to compile project

uart_receive.c: Receive function

uart_receive: Executable file, generated after command make

/send:

Makefile: Code compilation, execute command make to compile project

uart_send.c: Send function

uart_send: Executable file, generated after command make

SEND CODE

1. Initialize wiringPi

```
if(wiringPiSetupGpio() < 0) { //use BCM2835 Pin number table
        printf("set wiringPi lib failed     !!! \r\n");
        return 1;
    } else {
        printf("set wiringPi lib success    !!! \r\n");
    }
```

2. Open serial

```
  int fd;
    if((fd = serialOpen (UART_DEV1, 115200)) < 0) {
        printf("serial err\n");
        return -1;
    }
```

3. Clear buffer

```
serialFlush(fd);
serialPrintf(fd,"\r");
```

4. Send a string

```
char *buf = "abcdefgh";
serialPuts(fd, buf);
```

5. Close serial

```
serialClose(fd);
```

RECEIVE CODE

1. Initialize wiringPi

```
    if(wiringPiSetupGpio() < 0) { //use BCM2835 Pin number table
        printf("set wiringPi lib failed    !!! \r\n");
        return 1;
    } else {
        printf("set wiringPi lib success   !!! \r\n");
}
```

2. Open serial

```
if((fd = serialOpen (UART_DEV2, 115200)) < 0) {
        printf("serial err\n");
        return -1;
}
```

```
#define UART_DEV1      "/dev/ttySC0"
#define UART_DEV2      "/dev/ttySC1"
```

3. Receive and print

```
for (;;) {
putchar(serialGetchar(fd));
}
```

PYTHON

1. Open the folder of code

```
cd Serial_Expansion_HAT_code/pythont/

#change the path of you didn’ t put the code in /home/pi
```

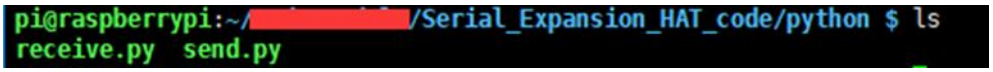
2. Compile and run

```
make

sudo python receive.py

#or sudo python send.py
```

FILES



【Note】Only serial codes provided

RECEIVE CODE

1. Open serial

```
ser = serial.Serial("/dev/ttySC1",115200,timeout=1)
```

2. Clear buffer

```
ser.flushInput()
```

3. Read data

```
ser.inWaiting()
```

4. Read specified bytes of data

```
ser.read(ser.inWaiting())
```

SEND CODE

1. Open serial

```
ser = serial.Serial("/dev/ttySC1",115200,timeout=1)
```

2. Define data sent

```
ser = serial.Serial("/dev/ttySC1",115200,timeout=1)
```

3. Write the data to serial

```
ser.write(command)
```