How to use SPI

We provide some examples for SPI URVEPi usage. Python, C++ and Node.js. Below you can see how to use SPI with this languages.

How to use URVEPi SPI with Python

For Python we need to install spidev library. It's very easy. Just type in your root account:

```
git clone https://github.com/doceme/py-spidev.git
cd py-spidev
apt-get install python3-dev nano
sudo python3 setup.py build
sudo python3 setup.py install
```

Then you can write simple app in Python:

```
nano spitest.py
```

Now paste this source code and save:

```
import spidev
spi = spidev.SpiDev()
spi.open(0, 0)
spi.max_speed_hz = 500000
spi.mode = 0
tx_data = [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x40, 0x00, 0x00, 0x00, 0x00, 0x95,
        0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
        0xDE, 0xAD, 0xBE, 0xEF, 0xBA, 0xAD, 0xF0, 0x0D]
rx_data = spi.xfer2(tx_data)
print(rx_data)
spi.close()
```

```
python3 spitest.py
```

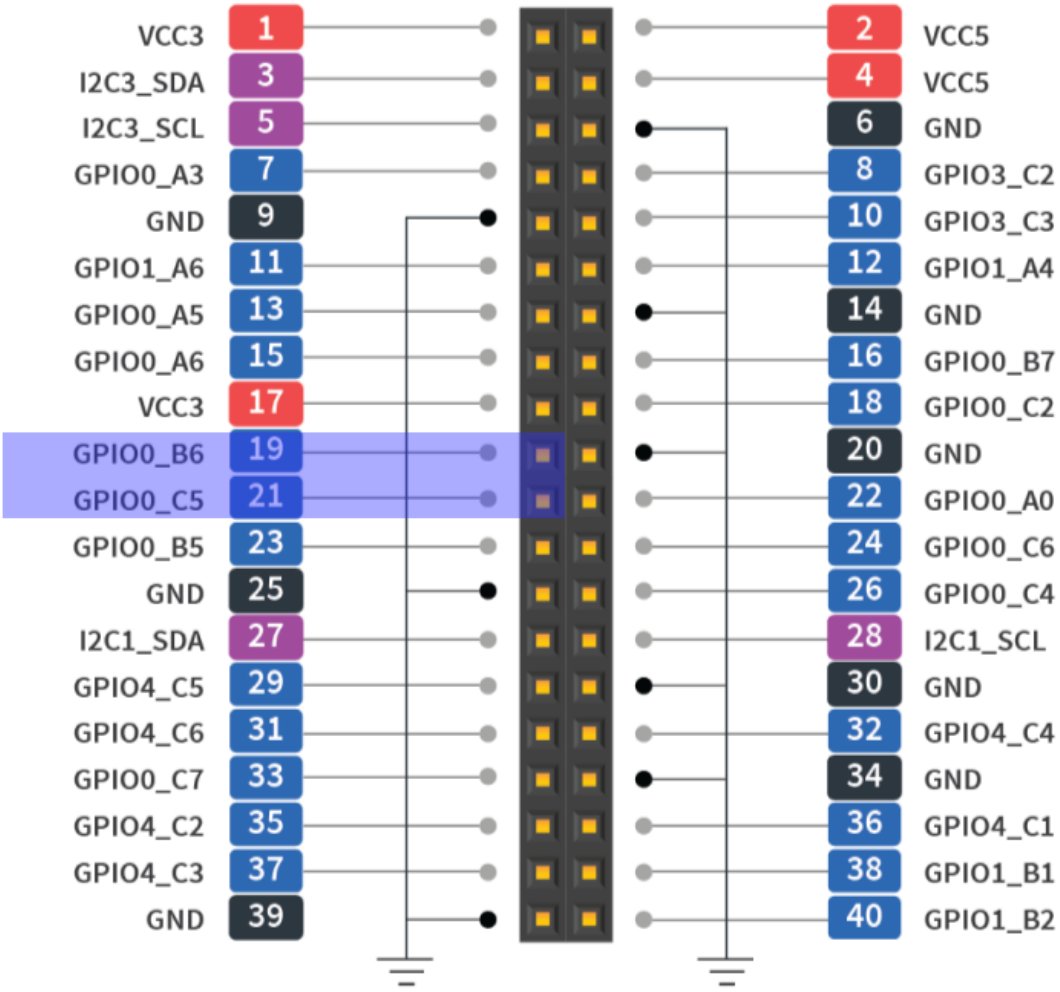
Or you can send and receive text

```
import spidev
spi = spidev.SpiDev()
spi.open(0, 0)
spi.max_speed_hz = 500000
spi.mode = 0
tx_data = "URVEPi rocks!".encode('utf-8')
rx_data = spi.xfer2(tx_data)
print(bytes(rx_data).decode('utf-8'))
spi.close()
```

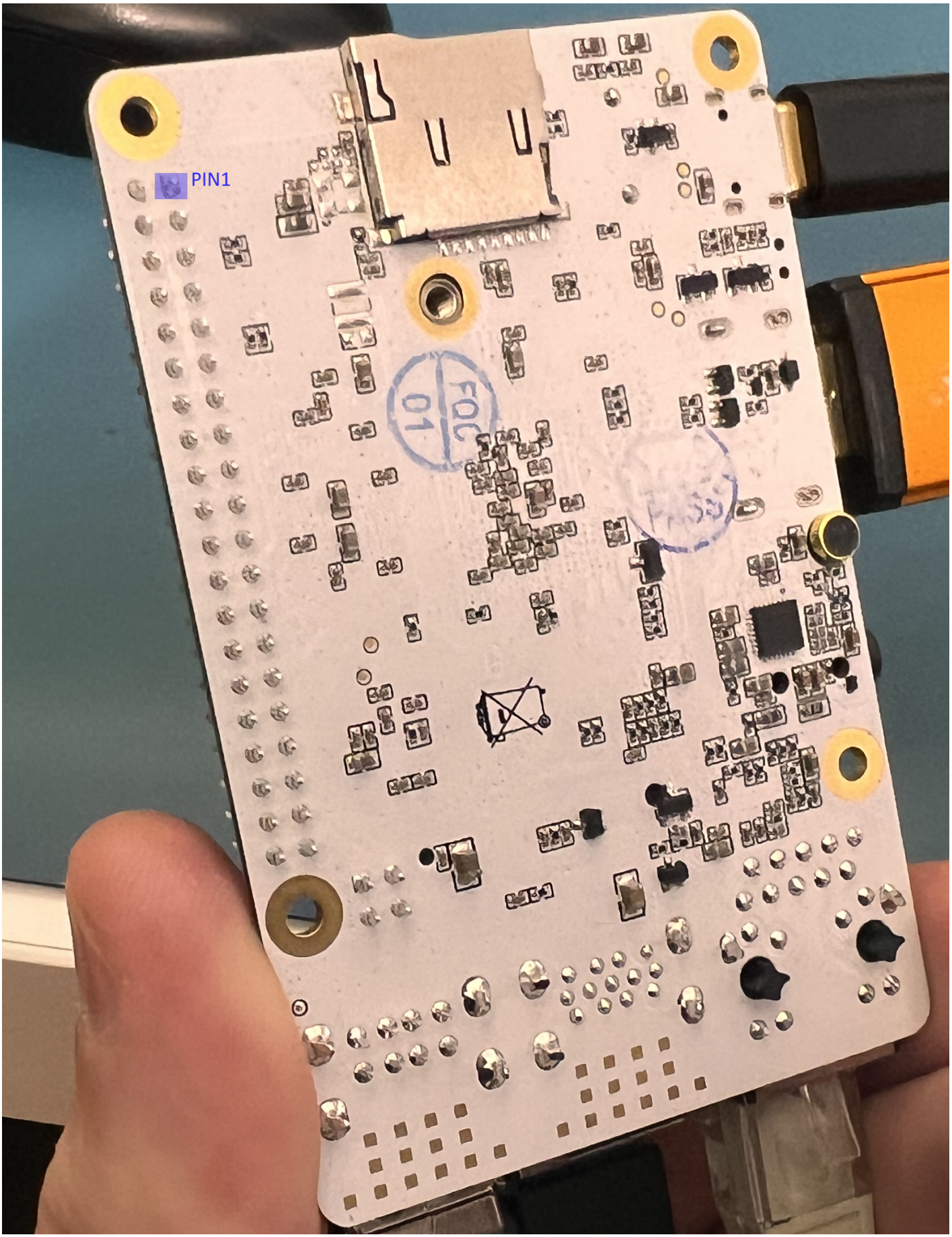
```
python3 spitest2.py
URVEPi rocks!
```

How to connect SPI to URVEPi

To test and run SPI pins on URVEPi, you need short-circuit Pin19 (SPI input) and Pin21 (SPI output). As we want push and get data with SPI bus, we will send something though Pin21 and we will receive it on Pin19.



Pin1 is the one with a square.


PIN1
FOC
01

How to use URVEPi SPI with C++

```
apt-get install build-essential nano
nano spidevtest.c   # paste source code from below
gcc spidevtest.c -o spidev_test
chmod +x spidev_test
./spidev_test -D /dev/spidev0.0
```

Below source code of spidevtest.c

```
/*
 * SPI testing utility (using spidev driver)
 *
 * Copyright (c) 2007  MontaVista Software, Inc.
 * Copyright (c) 2007  Anton Vorontsov <avorontsov@ru.mvista.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License.
 *
 * Cross-compile with cross-gcc -I/path/to/cross-kernel/include
 */

#include <stdint.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/types.h>
#include <linux/spi/spidev.h>

#define ARRAY_SIZE(a) (sizeof(a) / sizeof((a)[0]))

static void pabort(const char *s)
{
	perror(s);
```

```
	abort();
}

static const char *device = "/dev/spidev0.0";
static uint8_t mode;
static uint8_t bits = 8;
static uint32_t speed = 500000;
static uint16_t delay;

static void transfer(int fd)
{
	int ret;
	uint8_t tx[] = {
		0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
		0x40, 0x00, 0x00, 0x00, 0x00, 0x95,
		0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
		0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
		0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
		0xDE, 0xAD, 0xBE, 0xEF, 0xBA, 0xAD,
		0xF0, 0x0D,
	};
	uint8_t rx[ARRAY_SIZE(tx)] = {0, };
	struct spi_ioc_transfer tr = {
		.tx_buf = (unsigned long)tx,
		.rx_buf = (unsigned long)rx,
		.len = ARRAY_SIZE(tx),
		.delay_usecs = delay,
		.speed_hz = speed,
		.bits_per_word = bits,
	};

	ret = ioctl(fd, SPI_IOC_MESSAGE(1), &tr);
	if (ret < 1)
		pabort("can't send spi message");

	for (ret = 0; ret < ARRAY_SIZE(tx); ret++) {
		if (!(ret % 6))
			puts("");
		printf("%.2X ", rx[ret]);
```

```
	}
	puts("");
}

static void print_usage(const char *prog)
{
	printf("Usage: %s [-DsbdlHOLC3]\n", prog);
	puts("  -D --device   device to use (default /dev/spidev1.1)\n"
	     "  -s --speed    max speed (Hz)\n"
	     "  -d --delay    delay (usec)\n"
	     "  -b --bpw      bits per word \n"
	     "  -l --loop     loopback\n"
	     "  -H --cpha     clock phase\n"
	     "  -O --cpol     clock polarity\n"
	     "  -L --lsb      least significant bit first\n"
	     "  -C --cs-high  chip select active high\n"
	     "  -3 --3wire    SI/SO signals shared\n");
	exit(1);
}

static void parse_opts(int argc, char *argv[])
{
	while (1) {
		static const struct option lopts[] = {
			{ "device",  1, 0, 'D' },
			{ "speed",   1, 0, 's' },
			{ "delay",   1, 0, 'd' },
			{ "bpw",     1, 0, 'b' },
			{ "loop",    0, 0, 'l' },
			{ "cpha",    0, 0, 'H' },
			{ "cpol",    0, 0, 'O' },
			{ "lsb",     0, 0, 'L' },
			{ "cs-high", 0, 0, 'C' },
			{ "3wire",   0, 0, '3' },
			{ "no-cs",   0, 0, 'N' },
			{ "ready",   0, 0, 'R' },
			{ NULL, 0, 0, 0 },
		};
		int c;
```

```
    c = getopt_long(argc, argv, "D:s:d:b:lHOLC3NR", lopts, NULL);

    if (c == -1)
        break;

    switch (c) {
    case 'D':
        device = optarg;
        break;
    case 's':
        speed = atoi(optarg);
        break;
    case 'd':
        delay = atoi(optarg);
        break;
    case 'b':
        bits = atoi(optarg);
        break;
    case 'l':
        mode |= SPI_LOOP;
        break;
    case 'H':
        mode |= SPI_CPHA;
        break;
    case 'O':
        mode |= SPI_CPOL;
        break;
    case 'L':
        mode |= SPI_LSB_FIRST;
        break;
    case 'C':
        mode |= SPI_CS_HIGH;
        break;
    case '3':
        mode |= SPI_3WIRE;
        break;
    case 'N':
        mode |= SPI_NO_CS;
```

```
		break;
	case 'R':
		mode |= SPI_READY;
		break;
	default:
		print_usage(argv[0]);
		break;
	}
}

int main(int argc, char *argv[])
{
	int ret = 0;
	int fd;

	parse_opts(argc, argv);

	fd = open(device, O_RDWR);
	if (fd < 0)
		pabort("can't open device");

	/*
	 * spi mode
	 */
	ret = ioctl(fd, SPI_IOC_WR_MODE, &mode);
	if (ret == -1)
		pabort("can't set spi mode");

	ret = ioctl(fd, SPI_IOC_RD_MODE, &mode);
	if (ret == -1)
		pabort("can't get spi mode");

	/*
	 * bits per word
	 */
	ret = ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits);
	if (ret == -1)
		pabort("can't set bits per word");
```

```
	ret = ioctl(fd, SPI_IOC_RD_BITS_PER_WORD, &bits);
	if (ret == -1)
		pabort("can't get bits per word");

	/*
	 * max speed hz
	 */
	ret = ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
	if (ret == -1)
		pabort("can't set max speed hz");

	ret = ioctl(fd, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
	if (ret == -1)
		pabort("can't get max speed hz");

	printf("spi mode: %d\n", mode);
	printf("bits per word: %d\n", bits);
	printf("max speed: %d Hz (%d KHz)\n", speed, speed/1000);

	transfer(fd);

	close(fd);

	return ret;
}
```

How to use SPI with node.js

You need to install node.js

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/install.sh | bash
nvm install node
reboot # we need to initialize nvm, or just run once again terminal
npm install pi-spi
```

Create and run file spitest.js

```
nano spitest.js
```

```
const Spi = require('pi-spi');
const spi = Spi.initialize('/dev/spidev0.0');
spi.clockSpeed(500000);
spi.dataMode(0);
const txData = Buffer.from('URVEPi rocks!', 'utf-8');
spi.transfer(txData, txData.length, function(err, rxData) {
  if (err) throw err;
  console.log(rxData.toString('utf-8'));
  spi.close(function(err) {
    if (err) throw err;
  });
});
```

```
node spitest.js
URVEPi rocks!
```

Revision #3
Created 29 March 2023 20:10:59 by Import
Updated 7 April 2023 06:34:02 by Import