Raspberry Pi Resources

Our resources for other geeks, designers and engineers.

Using The SPI Interface

Adam

Resources

https://www.raspberrypi.org/documentation/hardware/raspberrypi/spi/README.md

Enabling The SPI Port

The SPI port needs to be enabled in Rasbian before it can be used. See here.

Leave the IO pins used unconfigured (do not set them as inputs or outptus).

Using The SPI Port With The BCM2835 library by Mike McCauley

This uses the same library as used for the IO pins – see here.

```
//Setup SPI pins
bcm2835_spi_begin();

//Set CS pins polarity to low
bcm2835_spi_setChipSelectPolarity(BCM2835_SPI_CS0, 0);
bcm2835_spi_setChipSelectPolarity(BCM2835_SPI_CS1, 0);

//Set SPI clock speed
//  BCM2835_SPI_CLOCK_DIVIDER_65536 = 0,       ///< 65536 = 262.144us = 3.814697260kHz (to
//  BCM2835_SPI_CLOCK_DIVIDER_32768 = 32768,   ///< 32768 = 131.072us = 7.629394531kHz
//  BCM2835_SPI_CLOCK_DIVIDER_16384 = 16384,   ///< 16384 = 65.536us = 15.25878906kHz
//  BCM2835_SPI_CLOCK_DIVIDER_8192  = 8192,    ///< 8192 = 32.768us = 30/51757813kHz
//  BCM2835_SPI_CLOCK_DIVIDER_4096  = 4096,    ///< 4096 = 16.384us = 61.03515625kHz
//  BCM2835_SPI_CLOCK_DIVIDER_2048  = 2048,    ///< 2048 = 8.192us = 122.0703125kHz
//  BCM2835_SPI_CLOCK_DIVIDER_1024  = 1024,    ///< 1024 = 4.096us = 244.140625kHz
//  BCM2835_SPI_CLOCK_DIVIDER_512   = 512,     ///< 512 = 2.048us = 488.28125kHz
//  BCM2835_SPI_CLOCK_DIVIDER_256   = 256,     ///< 256 = 1.024us = 976.5625MHz
//  BCM2835_SPI_CLOCK_DIVIDER_128   = 128,     ///< 128 = 512ns = = 1.953125MHz
//  BCM2835_SPI_CLOCK_DIVIDER_64    = 64,      ///< 64 = 256ns = 3.90625MHz
//  BCM2835_SPI_CLOCK_DIVIDER_32    = 32,      ///< 32 = 128ns = 7.8125MHz
//  BCM2835_SPI_CLOCK_DIVIDER_16    = 16,      ///< 16 = 64ns = 15.625MHz
//  BCM2835_SPI_CLOCK_DIVIDER_8     = 8,       ///< 8 = 32ns = 31.25MHz
//  BCM2835_SPI_CLOCK_DIVIDER_4     = 4,       ///< 4 = 16ns = 62.5MHz
//  BCM2835_SPI_CLOCK_DIVIDER_2     = 2,       ///< 2 = 8ns = 125MHz, fastest you can get
//  BCM2835_SPI_CLOCK_DIVIDER_1     = 1,       ///< 1 = 262.144us = 3.814697260kHz, same a
bcm2835_spi_setClockDivider(BCM2835_SPI_CLOCK_DIVIDER_128);
```

Tweet Follow Admin

Categories

open all | close all

- Arduino
- Bash
- Command Line
- Other Boards
- Pi Hardware
- Pi Operating Systems
- Programming In Assembler
- Programming in C/C++
  - .Compilers and IDE's
  - .Distribution of applications
  - .Libraries
  - Arguments
  - Audio
  - Bluetooth
  - Boost C++ Libraries
  - Classes
  - Config files etc
  - Console
  - Databases
  - DateTime
  - Delays
  - Display
  - Exit
  - File Input and Output
  - GUI
  - I2C
  - IO Pins
  - json

```
//Set SPI data mode
//  BCM2835_SPI_MODE0 = 0,  // CPOL = 0, CPHA = 0, Clock idle low, data is clocked in on r
//  BCM2835_SPI_MODE1 = 1,  // CPOL = 0, CPHA = 1, Clock idle low, data is clocked in on f
//  BCM2835_SPI_MODE2 = 2,  // CPOL = 1, CPHA = 0, Clock idle high, data is clocked in on
//  BCM2835_SPI_MODE3 = 3,  // CPOL = 1, CPHA = 1, Clock idle high, data is clocked in on
bcm2835_spi_setDataMode(BCM2835_SPI_MODE0);   //(SPI_MODE_# | SPI_CS_HIGH)=Chip Select act

//Set with CS pin to use for next transfers
bcm2835_spi_chipSelect(BCM2835_SPI_CS0);

//Transfer 1 byte
//uint8_t data;
//data = bcm2835_spi_transfer((uint8_t)0x55);

//Transfer many bytes
char data_buffer[10];
int Count;
for (Count = 0; Count < 10; Count++)
  data_buffer[Count] = 0x80 + Count;
bcm2835_spi_transfern(&data_buffer[0], 10);     //data_buffer used for tx and rx

//Return SPI pins to default inputs state
//bcm2835_spi_end();
```

Using The SPI Port Without the BCM2835 Library

This working example is based on the excellent example here.

```
#include <fcntl.h>        //Needed for SPI port
#include <sys/ioctl.h>      //Needed for SPI port
#include <linux/spi/spidev.h> //Needed for SPI port
#include <unistd.h>     //Needed for SPI port
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <iostream>
#include <unistd.h>
#include <cstring>

int spi_cs0_fd;       //file descriptor for the SPI device
int spi_cs1_fd;       //file descriptor for the SPI device
unsigned char spi_mode;
unsigned char spi_bitsPerWord;
unsigned int spi_speed;
```


- Memory
- Null
- Pipes
- PWM
- Random
- Scheduler
- Security
- Semaphores
- Signal Handling
- Speed
- SPI
  - Using The SPI Interface
- Strings
- TCP/IP
- Termination
- Threads
- Timing
- Troubleshooting
- UART Serial Port
- .Getting Your RPi Ready For C P
- C General
- Running Your Exe
- Programming In PHP and JavaScrip
- Programming in Python
- Projects
- Software and Utilities

Pages

- Contact us
- Electronic Designers in the South East of England

```
//***********************************
//***********************************
//********** SPI OPEN PORT **********
//***********************************
//***********************************
//spi_device  0=CS0, 1=CS1
int SpiOpenPort (int spi_device)
{
  int status_value = -1;
    int *spi_cs_fd;


    //----- SET SPI MODE -----
    //SPI_MODE_0 (0,0)  CPOL = 0, CPHA = 0, Clock idle low, data is clocked in on rising edg
    //SPI_MODE_1 (0,1)  CPOL = 0, CPHA = 1, Clock idle low, data is clocked in on falling ed
    //SPI_MODE_2 (1,0)  CPOL = 1, CPHA = 0, Clock idle high, data is clocked in on falling e
    //SPI_MODE_3 (1,1)  CPOL = 1, CPHA = 1, Clock idle high, data is clocked in on rising, e
    spi_mode = SPI_MODE_0;

    //----- SET BITS PER WORD -----
    spi_bitsPerWord = 8;

    //----- SET SPI BUS SPEED -----
    spi_speed = 1000000;    //1000000 = 1MHz (1uS per bit)


    if (spi_device)
      spi_cs_fd = &spi_cs1_fd;
    else
      spi_cs_fd = &spi_cs0_fd;


    if (spi_device)
      *spi_cs_fd = open(std::string("/dev/spidev0.1").c_str(), O_RDWR);
    else
      *spi_cs_fd = open(std::string("/dev/spidev0.0").c_str(), O_RDWR);

    if (*spi_cs_fd < 0)
    {
        perror("Error - Could not open SPI device");
        exit(1);
    }

    status_value = ioctl(*spi_cs_fd, SPI_IOC_WR_MODE, &spi_mode);
    if(status_value < 0)
    {
        perror("Could not set SPIMode (WR)...ioctl fail");
        exit(1);
    }

    status_value = ioctl(*spi_cs_fd, SPI_IOC_RD_MODE, &spi_mode);
    if(status_value < 0)
    {
      perror("Could not set SPIMode (RD)...ioctl fail");
```

```
      exit(1);
    }

    status_value = ioctl(*spi_cs_fd, SPI_IOC_WR_BITS_PER_WORD, &spi_bitsPerWord);
    if(status_value < 0)
    {
      perror("Could not set SPI bitsPerWord (WR)...ioctl fail");
      exit(1);
    }

    status_value = ioctl(*spi_cs_fd, SPI_IOC_RD_BITS_PER_WORD, &spi_bitsPerWord);
    if(status_value < 0)
    {
      perror("Could not set SPI bitsPerWord(RD)...ioctl fail");
      exit(1);
    }

    status_value = ioctl(*spi_cs_fd, SPI_IOC_WR_MAX_SPEED_HZ, &spi_speed);
    if(status_value < 0)
    {
      perror("Could not set SPI speed (WR)...ioctl fail");
      exit(1);
    }

    status_value = ioctl(*spi_cs_fd, SPI_IOC_RD_MAX_SPEED_HZ, &spi_speed);
    if(status_value < 0)
    {
      perror("Could not set SPI speed (RD)...ioctl fail");
      exit(1);
    }
    return(status_value);
}



//************************************
//************************************
//********** SPI CLOSE PORT **********
//************************************
//************************************
int SpiClosePort (int spi_device)
{
  int status_value = -1;
    int *spi_cs_fd;

    if (spi_device)
      spi_cs_fd = &spi_cs1_fd;
    else
      spi_cs_fd = &spi_cs0_fd;


    status_value = close(*spi_cs_fd);
    if(status_value < 0)
    {
      perror("Error - Could not close SPI device");
```

```
      exit(1);
    }
    return(status_value);
}



//***************************************
//***************************************
//********** SPI WRITE & READ DATA **********
//***************************************
//***************************************
//data    Bytes to write.  Contents is overwritten with bytes read.
int SpiWriteAndRead (int spi_device, unsigned char *data, int length)
{
  struct spi_ioc_transfer spi[length];
  int i = 0;
  int retVal = -1;
    int *spi_cs_fd;

    if (spi_device)
      spi_cs_fd = &spi_cs1_fd;
    else
      spi_cs_fd = &spi_cs0_fd;

  //one spi transfer for each byte

  for (i = 0 ; i < length ; i++)
  {
    memset(&spi[i], 0, sizeof (spi[i]));
    spi[i].tx_buf        = (unsigned long)(data + i); // transmit from "data"
    spi[i].rx_buf        = (unsigned long)(data + i) ; // receive into "data"
    spi[i].len           = sizeof(*(data + i)) ;
    spi[i].delay_usecs   = 0 ;
    spi[i].speed_hz      = spi_speed ;
    spi[i].bits_per_word = spi_bitsPerWord ;
    spi[i].cs_change = 0;
  }

  retVal = ioctl(*spi_cs_fd, SPI_IOC_MESSAGE(length), &spi) ;

  if(retVal < 0)
  {
    perror("Error - Problem transmitting spi data..ioctl");
    exit(1);
  }

  return retVal;
}
```

Useful Resources

http://hertaville.com/2013/07/24/interfacing-an-spi-adc-mcp3008-chip-to-the-raspberry-pi-using-c/

Issues

"Error – Problem transmitting spi data..ioctl: Invalid argument

Due to changes in the underlying library the spi_ioc_transfer struct now needs to be initialised to NULL, and a hacky fix is to add this to the beginning of the for loop (this has been done in the code example above)

```
memset(&spi[i], 0, sizeof (spi[i]));
```

USEFUL?
We benefit hugely from resources on the web so we decided we should try and give back some of our knowledge and resources to the community by opening up many of our company's internal notes and libraries through mini sites like this. We hope you find the site helpful.

Please feel free to comment if you can add help to this page or point out issues and solutions you have found, but please note that we do not provide support on this site. If you need help with a problem please use one of the many online forums.

venkatesh • 5 years ago

http://mitchtech.net/raspbe...

1 • Reply • Share ›

Tom Ritchford • 4 years ago

Yowza, this pattern:

std::string("/dev/spidev0.1").c_str(),

is terrible in many ways - for one thing, that std::string will get destroyed and thus that pointer won't work past that line!

Why not just use "/dev/spidev0.1" ??? There's no advantage to constructing that string!

3 • Reply • Share ›

Adam Mod ➚ Tom Ritchford • 4 years ago

Hi, thanks for the comment. Its based on a copy of another resource and has worked for us no problem. I'll see if your alternative works when I'm next using the SPI port.

1 • Reply • Share ›

anw1652 ➚ Adam • 4 years ago

My C++ is a little rusty (though I've spent many, many years as programming C++, just not in the past couple or three years), and I'm not familiar with the new 2011 standard, but while I think Tom is correct, I don't think this string is used again outside this function; therefore it doesn't matter. Unless something changed in the

"newer" C++, the string will survive throughout its declared scope, which is this function.

I'm just starting RPi development and found this site (and, thanks so much). That construction is useless, though (again, unless I'm missing something), and, therefore, when I shamelessly plagiarized this code and modified it into a real (though probably antiquated) C++ class, I deleted it.

Now, I haven't as yet tested it...

Reply • Share ›

Jon W ➚ anw1652 • 3 years ago

If the string has a name, it survives to the end of scope for that name. However, the line quoted just constructs a temporary string, and then calls c_str() on it, and then holds on to that, without giving the std::string a name. Such temporaries die at the end of the expression. Thus, the pointer returned by c_str() is undefined after the expression. It may "work" for now, but it's certainly not "correct" -- if you use a debugging memory manager that clears all memory when it's freed, for example, it would probably stop working.

Reply • Share ›

anw1652 ➚ Jon W • 3 years ago

Yes, you're right: the scope isn't even this function; but when the cstr is created, the pointer is pushed on the stack, and will survive through the open call, right? I know you can pass a constant, literal char* and be perfectly safe in this circumstance (like the form 'some_function("string_here")'). A pointer to the temporary string is pushed on the stack, and it and the string are good until the called function returns and the stack is cleaned up.

Reply • Share ›

varun gonsalves ➚ Adam • 4 months ago

This is the error I come across when I try to run the Code given above..

I really dont understand what the problem could be, after reading a little I came

I really dont understand what the problem could be, after reading a little I came across this thread for such errors, but that did not help tho : http://stackoverflow.com/qu...

The error I am stuck at :

/usr/lib/gcc/arm-linux-gnueabihf/4.9/../../../arm-linux-gnueabihf/crt1.o: In function `_start':
/build/glibc-g3vikB/glibc-2.19/csu/../ports/sysdeps/arm/start.S:119: undefined reference to `main'
collect2: error: ld returned 1 exit status

Reply • Share ›

Dave Winder • 4 years ago

is the struct really set up to need redundantly setting all these for every byte?

```
spi[i].len = sizeof(*(data + i)) ;
spi[i].delay_usecs = 0 ;
spi[i].speed_hz = spi_speed ;
spi[i].bits_per_word = spi_bitsPerWord ;
spi[i].cs_change = 0;
```

1 Reply • Share ›

Adam Mod → Dave Winder • 4 years ago

Can you share a better method?

Reply • Share ›

Dave Winder → Adam • 4 years ago

SCRATCH THIS

Not one that would work with spidev. After looking at the code there, it seems it does expect these for every byte. It just doesnt seem like they are thinks that would change in the middle of a transaction and therefore rather than passing an array of structs it seems you would pass a struct that had those settings followed by the array of data. Speed and bits per word are set up in initialization as well.

Reply • Share ›

Matt Van de Werken ➚ Dave Winder • a year ago

I disagree. The actual transfer line only operates on the first member of the spi array, so the extra initialisation is wasted. See my code above.

^ | v • Reply • Share ›

Dave Winder ➚ Adam • 4 years ago

The answer is here: http://www.cs.fsu.edu/~bake...
The length in SPI_IOC_MESSAGE is meant to be the number of spi_ioc_transfers. So you are correctly setting up length single byte transfers. If you made it one length byte transfer then all that struct filling would not be needed.
Anyway, thanks this made me learn something.

^ | v • Reply • Share ›

Matt Van de Werken ➚ Dave Winder • a year ago

I agree. The whole function SpiWriteAndReadcontains redundant code.

The below should work better:

```
//*****************************************
//*****************************************
//********** SPI WRITE & READ DATA **********
//*****************************************
//*****************************************
//data Bytes to write. Contents is overwritten with bytes read.
int SpiWriteAndRead (int spi_device, unsigned char *data, int length)
{
struct spi_ioc_transfer spi;
int i = 0;
int retVal = -1;
int *spi_cs_fd;

if (spi_device)
spi_cs_fd = &spi_cs1_fd;
```

see more

Reply • Share ›

Adam Mod ➚ Matt Van de Werken • a year ago

Hi Matt thanks, will look into it the next time we're working with SPI...

Reply • Share ›

NicP • 3 years ago

Something I encountered which may be useful to someone - I've been wrestling with a Pi and 2 x 23S17 port expanders for weeks and have just got to the bottom of the problems. To cut to the chase, my Pi (driven from a standard USB adaptor) has a 5V line around 4.70 volts. A tad low. The 2 x 23S17's have been sitting on a breadboard for weeks with an independent (and accurate) 5v supply. I lose random bits on read, usually the low bit. Writes appear fine. If the 23S17's are run from the Pi, all is fine - but also behave if I drop the nice, accurate, 5v line with a simple silicon diode. My other problem was the well documented 'A2 is never ignored irrespective of HAEN'. The USB adaptor probably needs a good check because 4.7V seems very low.. Anyway perhaps this is useful to someone..

Reply • Share ›

NicP ➚ NicP • 3 years ago

This wasn't the supply, though there is quite a lot of 30KHz noise on the Pi 3.3v line. To recap, running two MCP23S17s powered from the Pi always worked, but when powered separately, I would get read errors occasionally. Or so I thought.

Well no - they weren't read errors. Repeated reads of the same registers was consistent, turned out data was getting lost going out from the Pi.
I now have the two chips running happily on 5V with the return (MISO) line appropriately limited to keep it within the Pi 3.3 volts.
The fix was to add a 74HC buffer on each of the outgoing lines, MOSI, SCK and CS. (actually a 74HC04, six inverters wired as 3 buffers, it was idly sitting in the corner of the breadboard all the time).
I'm quite surprised this needed buffering, it's only running at 4MHz.

Reply • Share ›

Timothy Parez • 3 years ago

I was following this guide and using the first code snippet (using the library). Everything compiles and I'm able to execute the code but as soon as the library is being used the application doesn't do anything (no compile or runtime exception either). For example, printf("Hello"); works if I do not call any of the bcm2835 methods, even if the printf is before the first call to any of those methods. If I remove all those calls, but still include bcm2835.h the printf output works fine. Any idea what's going on here or how to get error information? (and I realize this is a vague description sorry)

Reply • Share ›

topherCantrell • 3 years ago

I checked the spi "mode" value on a scope.

The comments in the code about CPHA are backwards. CPHA=0 means the data is valid on the first (zeroth) edge of the clock pulse (idle to active). CPHA=1 means the data is valid on the second (oneth) edge of the clock pulse (active to idle).

The breakdown is thus:

mode 00 ... clock idle low, data valid at beginning of pulse (low to high)
mode 01 ... clock idle low, data valid at end of pulse (high to low)
mode 10 ... clock idle high, data valid at beginning of pulse (high to low)
mode 11 ... clock idle high, data valid at end of pulse (low to high)

Reply • Share ›

Adam Mod → topherCantrell • 3 years ago

Pretty sure that is not the case :-)

Reply • Share ›

Guest → Adam • 3 years ago

I'll post the images from my scope tonight. I am using spidev in python. I wonder if that makes a difference.

Reply • Share ›

topherCantrell → Adam • 3 years ago

I tried writing the value 0xAA in all four SPI modes. I took a snapshot of each mode (see attached image).

My "breakdown" above is from these graphs. Again, I am using the python wrapper (spidev) for the SPI interface.

```
>>> import spidev
>>> spi = spidev.SpiDev()
>>> spi.open(0,0)

>>> spi.mode = 0
>>> spi.writebytes([0xAA])

>>> spi.mode = 1
>>> spi.writebytes([0xAA])

>>> spi.mode = 2
>>> spi.writebytes([0xAA])

>>> spi.mode = 3
>>> spi.writebytes([0xAA])
```


Reply • Share ›

Adam Mod → topherCantrell • 3 years ago

Nice one, I hadn't noticed that there we're 2 different blocks with the description of the line states in our code above. I've fixed the error thanks

Reply • Share ›

sergiov • 2 years ago

Don't you have an example for the case Using The SPI Port Without the BCM2835 Library with only C. No C++ only in C

only C. No C++ only in C

Reply • Share ›

hanay • 2 years ago

I've been using the latter code (i.e. the one not using using the BCM2835 library), and have been getting "*Error - Problem transmitting spi data..ioctl: Invalid argument*" in the SpiWriteAndRead() function. Bizarrely, the error only occurred if something was written to std::cout before the function was called.

It turned out that the spi_ioc_transfer struct needs to be initialised to NULL, and a hacky fix is to add

```
memset (&spi[i], 0, sizeof (spi[i]));
```

to the beginning of the for loop (I guess not all the fields are assigned in the code, and those that aren't need to be set to zero). More info here: https://bugs.launchpad.net/...

Hope this helps someone, and would also be useful to update the above code with a fix (the problem actually stems back to the hertaville code).

Reply • Share ›

Adam Mod ➔ hanay • 2 years ago

Thanks Hanay, I've added it to the code

Reply • Share ›

hydranix • 2 years ago

std::string("/dev/spidev0.1").c_str() is completely useless... Why construct a string object at all? Just use "/dev/sdpidev0.1", you accomplish the exact same thing, faster and more cheaply. Safer as well.

Reply • Share ›

Curtis Newton • 2 years ago

why use spidev when one can access memory directly

Reply • Share ›

Akshay Vijapur • 10 months ago

Can i use ioctl to write data to particular register.

Like this

SPI_TransferData(SPI_BASEADDR, 4, txBuffer, 4, rxBuffer, 1);

Reply • Share ›