Menu

Search:

Search

I2C and SPI

I2C and SPI

I2C (Inter Integrated Circuit Communications, pronounced I squared C) and SPI (or Serial-Peripheral interface) are protocols that can link a microcomputer to other micros or integrated circuits.

Both protocols are widely used in hobby electronics and robotics projects.

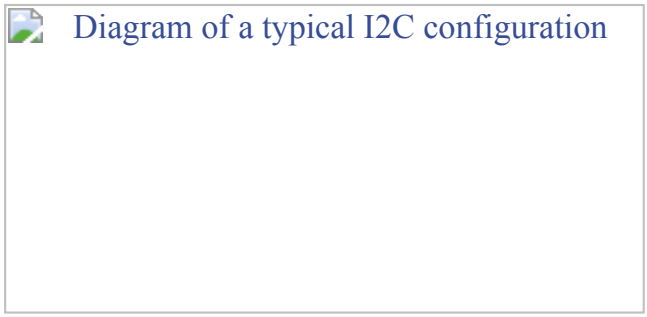
I2C was originally developed by Philips (now nxp) and is used in all sorts of equipment including virtually every tv, monitor and computer motherboard. I2C is a flexible protocol but has fairly limited bandwidth.

SPI was originally developed by Motorola (now Freescale). It's simple and fast but has some limitations compared to I2C.

Most modern micro-controllers have hardware support for both protocols. Examples include the Arduino, mbed, BeagleBone, BeagleBoard and Raspberry Pi.

Let's look at the two protocols in a little more detail.

I2C



I2C can be used to connect up to 127 nodes via a bus that only requires two data wires, known as SDA and SCL.
Since the devices need a common voltage reference (Ground) and a power line (referred to as Vcc or Vdd) the full bus has four wires, two for data and two for power. Sometimes additional lines are added, for example to support interrupts when the state of I2C devices change.

In theory the I2C bus can support multiple masters, but most micro-controllers can't. A master is usually a microcontroller, although it doesn't have to be. Slaves can be ICs or microcontrollers.

In the diagram above a RasPi is a master and there are several slaves: a Digital-to-Analog converter (DAC), an Analog-to-Digital converter (DC), and an Arduino.

I2C can be used to control a very wide range of devices. Common examples include

- Digital-to-analogue converters (DACs)
- Analogue-to-digital converters (ADCs)
- LCD displays
- OLED Screens
- Keyboards
- Motor drivers
- LED drivers
- Memory chips and cards (EEPROM, RAM, FERAM, Flash)
- bus expanders/extenders (chips with 8 or 16 I/O pins controllable via I2C)
- other microcontrollers

When the master wishes to communicate with a slave it sends a series of pulses down the SDA and SCL lines. The data that is sent includes an address that identifies the slave with which the master needs to interact. Addresses take 7 bits out of a data byte; the remaining bit specifies whether the master wishes to read (get data from a slave) or write (send data to a slave).

Some devices have an address that is entirely fixed by the manufacturer; others can be configured to take one of a range of possible addresses. When a micro-controller is used as a slave it is normally possible to configure its address by software, and for that address to take on any of the 127 possible values.

The address byte may be followed by one or more byes of data, which may go from master to slave or from slave to master.

When data is being sent on the SDA line, clock pulses are sent on the SCL line to keep master and slave

synchronised. Since the data is sent one bit at a time, the data transfer rate is one eighth of the clock rate. The original standard specified a s*tandard* clock rate of 100KHz, and most I2C chips and micro-controllers can support this. Later updates to the standard introduced a *fast* speed of 400KHz and a *high* speed of 1.7 or 3.4 MHz. The Arduino can support standard and fast speeds; the BeagleBoard has three I2C busses, one at each speed; the RasPi and BeagleBone can both support standard and fast I2C.

The fast rate corresponds to a data transfer rate of 50K bytes/sec which is too slow for some control applications. One option in that case is to use SPI instead of I2C.

SPI

The SPI interface was originally developed by Motorola (now Freescale). SPI is much simpler than I2C. Master and slave are linked by three data wires, usually called MISO, (Master in, Slave out), MOSI (Master out, Slave in) and M-CLK.

As the names suggest, the M_CLK line carries clock pulses which provide synchronisation. When a transfer is taking place, a bit of data is transferred via MOSI from Master to slave and at the same time a bit of data is transferred via MISO from Slave to Master. At the end of eight clock pulses an entire byte of data has been transferred in each direction.

Many SPI-enabled ICs and Microcontrollers can cope with data rates of over 10MHz, so transfer is much faster than with I2C. The downside is that SPI normally has no addressing capability; instead, devices are selected by means of a Chip Select signal which the master can use to enable one slave out of several connected to the SPI bus. If more than one slave exists, one chip select line is required per device, which can use precious GPIO lines on the Master.

SPI is less well specified than I2C, but the SPI module on the Broadcom chip at the heart of the Raspberry Pi is flexible enough to cope with most common SPI devices.

25 Comments

1. 
   Tan Chee Seong
   May 21, 2012 at 10:37

   Any memory chip in the market is with both SPI and I2C interfaces?

   Reply

   - 
     quick2wire
     May 25, 2012 at 06:13

     I don’t know of one, but there are certainly chip families which offer similar functionality via SPI and I2C. What sort of memory are you looking for? EEPROM, RAM or FRAM?

And what sort of capacity and speed?

Reply

Scott Penrose
July 9, 2012 at 03:27

Yes… every SD card has SPI interface. Very handy.

Reply

ITCSW
June 11, 2012 at 14:17

Hi guys, I have a digital TV which has an i2c chip. My intention is to integrate my RasPi into the TV and have it run the TV entirely, creating a “smart TV” that’s capable of running all the usual TV stuff as well as offering internet and network connectivity to provide on-demand services as well as standard web browsing. I have a circuit diagram for the TV, can you foresee any problems that I might encounter. I’m a qualified sparky and have been working in the B2B IT market as an IT director for the past 14 years, the problem is I have no electronic experience so I’ll be relying heavily on the community for help.
Cheers, Justin.

Reply

jean
June 30, 2012 at 04:52

the bigest problem is to reprogram the tv firmware

Reply

Jeroen Kransen
July 14, 2012 at 11:11

Hi I have the Raspberry Pi and a temperature sensor (thermometer?) with a SPI interface, which I want to read out. The sensor comes as a ready-to-use chip, the Dallas DS1620. The SPI driver seems to be in place, and I can figure out the wiring, but now I want to actually read data from

code, preferable Python or even Java. Could you give me directions on how to do that?

Reply

4. 
JBeale
August 10, 2012 at 21:18

"… the RasPi and BeagleBone can both support standard and fast I2C." According to the manual[1], the R-Pi I2C clock is programmable right up to 150 MHz (probably the I/O pins limit around 100 MHz)? But the limit is certainly above 400 kHz.
[1]BCM2835 manual p. 34: Clock Divider: SCL = core clock / CDIV Where core_clk is nominally 150 MHz. http://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf

Reply

5. 
Bhavik
October 3, 2012 at 12:59

I am trying to get Gyro compatible with analog signal and should have 3 axis. Is there any body help me out to select right Gyro please?

Reply

6. 
Bhavik
October 3, 2012 at 13:00

I am trying to get Gyro compatible with analog signal and should have 3 axis. Is there any body help me out to select right Gyro please? fffffffffffff

Reply

   - 
Romilly Cocking
November 12, 2012 at 15:34

There's an excellent buying guide at http://www.sparkfun.com/pages/accel_gyro_guide

Reply

7. 

Carsten
November 12, 2012 at 15:06

I am having a hard time to WRITE something to an EEPROM (24LC32). I can read it fine, but can't figure out how to write?! Writing with i2cset works well.

Could you give me a hint where to look or some code example?

Here my code (very rough) :

```
#!/usr/bin/python3

from time import sleep
import quick2wire.i2c as i2c

eeprom = 0×50

bus = i2c.I2CMaster(1)

for i in range(0,10):
bus.transaction(i2c.writing_bytes(eeprom, 0×00, 0×00+i),i2c.writing_bytes(eeprom, 0×00+i))
# bus.transaction(i2c.writing_bytes(eeprom, i))

bus.transaction(i2c.writing_bytes(eeprom, 0×00, 0×00))

i=0
while (True):
temp = bus.transaction(i2c.reading(eeprom,1))
print(temp)
i=i+1

sleep(.1)
```

Reply

   - 

Romilly Cocking
November 12, 2012 at 15:25

What happens when you run your code?

Reply

8. 

Carsten
November 12, 2012 at 15:37

Hi @Romilly Cocking,

No error, after the supposed to write lines (10-12) are done, it reads all bytes starting from address 0×000 just fine (I put some values with i2cset into the first memory cells).

So reading works well. So far I was not able to write something to adresses with Python and quick2wire. Maybe writing to a register works, but I havn't tried that so far.

Thanks,
Carsten

Reply

- 
  Brian Adams
  November 12, 2012 at 21:35

  Can you post the i2cset command lines that you used? I'm not familiar with the 24LC32, so that might be the best starting point.

  Reply

9. 
carsten
November 12, 2012 at 22:47

set the memory pointer…
root@raspiheiz:/home/cw# i2cset -y 1 0×50 0×00 0×00
write some bytes …
root@raspiheiz:/home/cw# i2cset -y 1 0×50 0×00 0×00 0xde i
root@raspiheiz:/home/cw# i2cset -y 1 0×50 0×00 0×01 0xad i
root@raspiheiz:/home/cw# i2cset -y 1 0×50 0×00 0×03 0xbe i
root@raspiheiz:/home/cw# i2cset -y 1 0×50 0×00 0×04 0xef i
set pointer to start…
root@raspiheiz:/home/cw# i2cset -y 1 0×50 0×00 0×00
every read will increment the pointer
root@raspiheiz:/home/cw# i2cget -y 1 0×50
0xde
root@raspiheiz:/home/cw# i2cget -y 1 0×50
0xad
root@raspiheiz:/home/cw# i2cget -y 1 0×50
root@raspihei
root@raspiheiz:/home/cw# i2cget -y 1 0×50

0xbe
root@raspiheiz:/home/cw#

sorry got a bit chaotic, did it on my tablet remotely but i think you can see it worked (beside my typo.)

carsten

Reply

Brian Adams
November 13, 2012 at 00:13

Can you try something like the following?

```
bus.transaction(i2c.writing_bytes(eeprom, 0×00, 0x00, 0xde));
bus.transaction(i2c.writing_bytes(eeprom, 0×00, 0x01, 0xad));
bus.transaction(i2c.writing_bytes(eeprom, 0×00, 0x02, 0xbe));
bus.transaction(i2c.writing_bytes(eeprom, 0×00, 0x03, 0xef));
```

(Disclaimer: I don't have my Pi with me, so forgive any typos!)

A call to 'writing_bytes' represents a new transaction segment, so will begin by signaling a START condition on the I2c bus. For your device, you should need only one START per memory write.

Reply

Carsten
November 13, 2012 at 09:46

cw@raspiheiz ~/firstpy $ sudo python3 ReadEEPROM.py
Traceback (most recent call last):
File "ReadEEPROM.py", line 15, in
bus.transaction(i2c.writing_bytes(eeprom, 0×00, 0×00, 0xde))
File "/usr/local/lib/python3.2/dist-packages/quick2wire/i2c.py", line 72, in transaction
ioctl(self.fd, I2C_RDWR, addressof(ioctl_arg))
IOError: [Errno 5] Input/output error
cw@raspiheiz ~/firstpy $

Maybe a problem with the 16bit Adresses of the EEPROM? I don't understand the quick2wire code by the first look 😉 So please take with a grain of salt. Looks to me that it would work with the usual byte wide addressing of registers in i2c devices.

Thanks for your support,

Carsten

Reply

Romilly Cocking
November 13, 2012 at 15:31

Carsten, the 24LC32 needs some time to perform the write. From memory I think it takes about 1.5ms

Try adding
from time import sleep
to the head of your program, and add
sleep(0.01)
after the line that writes the bytes. That will pause for 10 ms, which will be slow but at least we can see if it is working.

Reply

Carsten
November 13, 2012 at 16:12

Woh!

Works now.

I wonder if there is a way to check if the data was written? I mean I could put an try: around but maybe this should better be done in the module?

Thanks again,
Carsten

Romilly Cocking
November 13, 2012 at 16:49

I'm glad you got it working.

We will take a look at the best way to handle this type of device in the library.

Best wishes, Romilly

10. 

Joao Leitao
December 11, 2012 at 17:55

Hi,

i am having a problem with this…

```
data=[]
data.append('RUN(CKEY);')
data[0]=data[0]+chr(13)
address = 0x3e
iodir_register = 0×01
gpio_register = 0×01

with i2c.I2CMaster(1) as bus:
bus.transaction(i2c.writing(address, data[0]))
while True:
read_results = bus.transaction(i2c.writing_bytes(address, gpio_register),i2c.reading(address, 1))
print read_results
sleep(0.1)
```

This code works, although in the reading i always loose the some bytes…
instead of KEYED i get: KYD.
if i change to :

```
data=[]
data.append('RUN(CKEY);')
data[0]=data[0]+chr(13)

address = 0x3e
iodir_register = 0×01
gpio_register = 0×01

with i2c.I2CMaster(1) as bus:
bus.transaction(i2c.writing(address, data[0]))
while True:
read_results = bus.transaction(i2c.writing_bytes(address, gpio_register),i2c.reading(address, 10))
print read_results
sleep(0.1)
```

to read 10 bytes he reads them without any problem.

the question is that i never know how much bytes i will be getting on a read. this is for an Itron SMART TFT…

Any help?

thanks .

Reply

11. 

WaltR
June 24, 2013 at 07:27

I am working on spi to interface RF module(AWM24xxL) with PIC32mx360f512l. here, i can able to read but can't write the value in a specified address. Here is my code:

```
SPI1CON=0×0060; //Enabling Master mode and ckp=1;
SPI1BRG=320; // Setting the Baud rate
SPI1CON|=0×08000; //Enabling SPI module

while(1)
{
RF_Write_data(0×01,0×03);
cc=RF_Read_data(0×01);
if(cc==0×03)
mPORTFSetBits(BIT_3);
for(v1=0;v1++<10;);
mPORTFClearBits(BIT_3);
}

void RF_Write_data(unsigned char address,unsigned char data)
{
unsigned char k;
mPORTGClearBits(BIT_2);// slave_select =0
address &= 0x3F;
address |= ( REG_WRITE );
Spi_WriteByte( address );
Spi_WriteByte( data );
mPORTGSetBits(BIT_2); //Slave_select =1
}

unsigned char RF_Read_data(unsigned char address)
{
unsigned char var1;
mPORTGClearBits(BIT_2);
address &= 0x3F;
Spi_WriteByte(address);
var1=Spi_ReadByte(0×00);
mPORTGSetBits(BIT_2);
return var1;
}

unsigned char Spi_WriteByte(unsigned char data)
{
short temp;
SPI1BUF=data;
```

```
for(temp=0;temp++<1e3;);
temp=SPI1BUF;
return temp;
}

unsigned char Spi_ReadByte(unsigned char data)
{
short temp;
SPI1BUF=data;
for(temp=0;temp++<1e3;);
temp=SPI1BUF;
return temp;
}
```

Reply

12. Peter
August 13, 2013 at 10:37

Hi!

I've got a question concerning the spi chip-select.
Can anyone tell me where I have to change the quick2wire-code to use normal GPIO-pins for chip-select? (for Raspberry Pi)

Thanks a lot!

Reply

13. ps
August 19, 2013 at 11:10

What all things limits the speed of data transfer in SPI and I2C?

Reply

14. reza
September 15, 2013 at 06:44

Hi
I want use a AD7732!

this IC can connect to microcontroller with interface a-2wire b-spi!
interface 2 wire Better or interface SPI better?

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Comment

You may use these HTML tags and attributes: `<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <strike> <strong>`

Post Comment

Follow Us

- @Quick2Wire

Aritcles

- Testing the I2C Analogue Board – part 1
- Getting Started With Quick2Wire Boards
- Getting Started with the Beta kit
- How to Add Quick2Wire as a Raspbian Software Source
- Physical Python – Part 1
- Working safely with your Pi
- A gentle guide to Git and GitHub
- I2C and SPI

Recent Posts

- C3Pi and the Raspberry Pi learn another language
- Website update
- Raspberry Pi and Arduino linked via I2C

- The Babelboard is coming
- Thumbs up for QCon London

Archives

- April 2013
- March 2013
- February 2013
- January 2013
- December 2012
- November 2012
- August 2012
- July 2012
- June 2012
- May 2012
- April 2012
- March 2012

Categories

- API
- Arduino
- GPIO
- Hardware
- I2C
- News
- Programming
- Python
- Raspberry Pi
- SPI

Search:

Search

Design by picomol.