SPI versus I2C protocols

From BitWizard WIKI

When buying a BitWizard expansion board, either I2C or SPI can be selected as the interface. This page will help you choose.

Contents

Introduction

I2C and SPI are both bus protocols that allow short-distance, serial data transfer. I2C originates from the Philips semiconductor devision, while SPI was created by Motorola. Both protocols are commonly used in electronic devices like smartphones, TV's and laptops to control peripherals like power management chips, input devices and DACs. For instance, Microsoft uses I2C for the detachable keyboard on their Surface tablet.

Technical Differences

General Differences

I2C allows multiple masters and slaves on the bus. On the other hand SPI can only work with one master device controlling multiple slaves. In both I2C and SPI the master device controls the clock for all slaves, but an I2C slave device can modify the main bus clock. We will talk about this feature later.

Full-duplex/Half-duplex

The I2C protocol is inherently half-duplex, while the SPI protocol is inherently full-duplex. So with SPI, every read is also a write. This could in theory double the speed of the bus, however, when implementing the SPI protocol we noticed that most of the time we didn't have data to send one direction. When you want to read from the device, you have to send "dummy bytes" to the device to trigger the clock signal that allows the slave to send data. Similarly, most of time the slave does not have anything to report back when you are actually sending data to the slave. With dummy packets, both protocols are essentially half-duplex.

I2C Clock stretching

I2C has a feature called 'clock stretching'. If the slave device cannot send the data fast enough, it can hold down the clock so data will be transfered slower. This way, the bitrate is decreased, but the slave can keep up with the processing of the data. This feature can be very useful, since the clock rate can be high, and the slave will delay only when it needs to. The only problem is that the SoC on the Raspberry Pi has a bug in this feature. If clock stretching is used at the wrong time, the received might be corrupt. All BitWizard I2C boards work around this issue by carefully timing all I2C messages. The performance penalty is small, but as a result, all BitWizard I2C boards only work at a bus speed of 100kHz.

Sending/Writing data

The addresses on the bus are 7 bits wide. The lower bit specifies if the transaction is to be a read or a write. Write transactions have the lower bit cleared (0), read transactions have the lower bit set (1).

Each transaction on the bus starts with the address of the board. All BitWizard boards will ignore any transactions on the bus that do not start with their own address. The protocols allow for changing the address in case you need more modules of the same type on one bus.

After the address a single byte indicates the "port" on the board that the data is written to. The software can thus define 256 ports on each board.

To write data using SPI or I2C send a number of bytes (usually 3 or 4).

```
<address> <register> <data byte1> <data byte2> ....
```

On SPI you activate (make it low!) the slave select line before the transaction. On I2C you cause a start condition before the transaction. After the transaction, you deactivate the slave select line (make it high) on SPI and cause a STOP condition on I2C.

To read data from the slaves, on I2C you first make a write transaction of just two bytes, the address and the register number. Next you read the data. On SPI this must be done in one transaction: write the address and register number, then write dummy bytes while reading the data.

To read on I2C:

```
<start> <address + WRITE> <register> <stop>
<start> <address + READ> <byte 1> <byte 2>
```

To read on SPI:

```
MOSI <address> <register> <dummy byte> <dummy byte>
MISO    xxx       xxx     <byte 1>     <byte 2>
```

Retrieved from "http://www.bitwizard.nl/wiki/index.php?title=SPI_versus_I2C_protocols&oldid=2860"

- This page was last modified on 18 July 2014, at 15:31.