Industry Standard Control Interfaces for inter IC communication

Moritz Nöltner
University of Heidelberg, ZITI

***Abstract*—This paper gives a short overview of the I^{2}C (Inter-Integrated Circuit) and SPI (Serial Peripheral Interface) bus systems and showcases a record of a typical I^{2}C transaction.**

***Index Terms*—System buses, SPI, I^{2}C, Two Wire Interface.**

I. Introduction

In computer systems, different parts of the circuit will need to exchange data and configuration information. To facilitate this, a vast number of data link systems have been used which are distinguishable in terms of complexity, data throughput and whether they are point-to-point connections or bus systems. Some notable examples of data link systems in roughly descending order of complexity are:

- HyperTransport
- PATA, SATA
- PCI, PCI-Express
- AGP
- ISA
- CAN
- UART, USART
- SPI
- I^{2}C, SMBus
- UNI/O
- 1-Wire

Along with USART (Universal [Asynchronous] Receiver Transmitter), SPI (Serial Peripheral Interface) and I^{2}C (Inter-Integrated Circuit) are probably the most commonly used data links for low speed, low complexity interfacing of integrated circuits (ICs). SPI and I^{2}C are special in that they are bus systems, which means that a possibly large number of ICs can share the same signal lines for communication.

II. Serial Peripheral Interface

The Serial Peripheral Interface was introduced by Motorola for the 6800 series of processors, though Texas Instruments introduced a similar system, Microwire[4]. It allows to communicate to external components by using a simple, yet versatile 4-wire bus system with dedicated chip select, clock and data lines. By now, SPI has become a widely accepted de-facto standard with microcontrollers, digital signal processors, FLASH memories, ADCs and DACs and many sensors offering an SPI interface.

A. Technical Description

The SPI connects one master and any number of slaves. The master is the device that controls the transactions and provides the clock signal. There are three signal lines common to all participants. These are `CLOCK`, `MOSI` and `MISO` and one line specific to each slave, $\overline{\texttt{SS}}$.

`MOSI` Master Out, Slave In
This line carries the serial data, the master sends for the slave to receive with the most significant bit first.

`MISO` Master in, Slave Out
This line carries the slave's serial data to the master, again with the most significant bit first.

`CLOCK`The synchronisation clock the master sends to synchronise with a slave. During each clock cycle, one bit of information is sent by master and slave each.

$\overline{\texttt{SS}}$ Slave Select
The master asserts this line for a slave prior to starting a transaction with it by tieing it to a low level. For all other slaves, this line has to be at a logical high level, therefore deasserted. A slave, whose slave select is active will configure it's `MOSI` pin as input and `MISO` pin as output. All other slaves must disconnect themselves from the bus. This means, these slaves must not effect any changes on the bus lines and not be affected by the ongoing transaction.

The basic part of an SPI device is a shift register with an additional parallel access tap to write and read data. The `MOSI` line is connected to the shift out of the masters

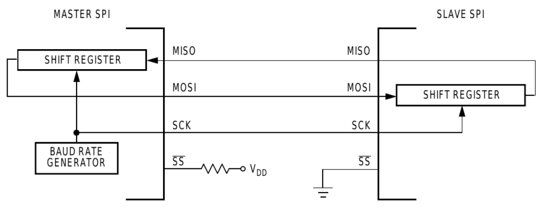


Fig. 1. A simple connection between a master and one slave.[3]

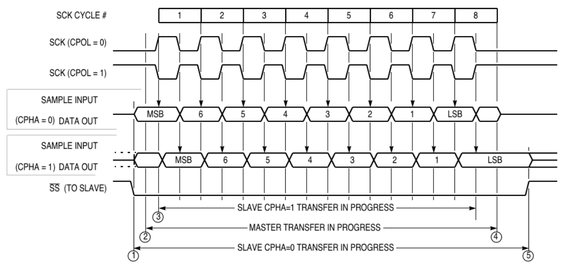


Fig. 2. An example timing diagram showcasing clock phase and polarity.[2]

shift register and to the shift in of the selected slaves shift register, and vice versa for the `MISO` line. The $\overline{\texttt{SS}}$ will determine which slaves connects it's shift register to the bus. `CLOCK` controls the shifting. Depending on the settings for clock polarity and clock phase, 4 modes are possible, which differ in when data is applied on the bus lines, and when it is sampled according to the clock cycle. See figure 1 for details on the connections and figure 2 for details on the clock signal.

B. Alternative Bus Topologies

There are a number of possible alternations of the SPI connection:

Star	All `MOSI` pins are tied together, all `CLOCK` pins are tied together and all `MISO` pins are tied together. The master controls the $\overline{\texttt{SS}}$ line of each slave. This is the most typical topology.
Serial	A master is connected to a number of slaves which are daisy chained. All $\overline{\texttt{SS}}$ pins can be tied together, or all the slaves $\overline{\texttt{SS}}$ pins can be set to a low level.

Star topology: To address a slave and start a transaction, the master only has to pull that slave's $\overline{\texttt{SS}}$ line low and start sending. After the frame is transmitted, master and slaves have exchanged the contents of their shift registers.

Serial topology: The master's `MOSI` pin is connected to the first slaves `MOSI` pin, the first slaves `MISO` pin is connected to the second slaves `MOSI` pin, whose `MISO` pin is connected to the third's `MOSI` pin and so on up to the last slave, whose `MISO` pin is connected to the master's `MISO` pin. The data sent over the bus will have to contain some address information and slaves which are not addressed by a data packet will have to pass it on through their `MISO` pin in the next frame. So, to address the n-th slave, the master will have to send a data packet for the n-th slave, and then n-1 more packets. During the transmission time of the n-1 packets, the data is passed on to the n-th slave. Then, to receive the response from the n-th slave, the master will have to continue sending until the response is passed through the remaining slaves. E.g. if there are a total of N slaves, the master will have to send N-n more packets.

Despite the obvious drawbacks, this topology provides two distinctive advantages over star topology:

1) The master does not need to control all the possibly quite many $\overline{\texttt{SS}}$ lines individually, therefore saving pins in i/o-pin restrained environments.
2) If there are only two devices whose $\overline{\texttt{SS}}$ pins are tied together, a multi-master communication becomes possible, because a master, whose $\overline{\texttt{SS}}$ pin is pulled low, must switch to slave mode if supported.

III. Inter-Integrated Circuit

The Inter-Integrated Circuit bus is a simple "snap-in" architecture developed by Philips, though some other companies use the name TWI for Two Wire Interface to avoid trademark conflicts pertaining to I^2C [9][10]. Devices can be added to or removed from the bus without the need to change anything else. The bus consists of just two wires and a pair of external pull-up resistors. The bus is designed for multi-master operation with collision detection being an integral part of the protocol. As with SPI, there are numerous devices of all kinds offering an I^2C interface. On top of this, a number of other bus systems are compatible with devices prepared to operate in either system. Such bus systems include the System Management Bus (SMBus)[7] found in modern personal computers, Power Management Bus (PMbus)[7] and CBUS as stated in the I^2C standard [6]. Figure 4 shows a (complex) setup of an I^2C system. For a simpler setup, consider only the rightmost part of the figure with the two F/S-mode devices, `SDA` and `SCL` lines, their pull-ups and V_{DD2}.

A. Technical Description

The I^2C bus connects a number of devices, of which any can become master to begin a transaction. Because

I^2C is a half-duplex bus, meaning only one device can send data at a time, there are not only the roles of master, which will iniate a transaction, and slave, which has to respond to it, but also transmitter and receiver. A transmitter is the device sending data, a receiver receives it. All four combinations of master/slave and transmitter/receiver are possible. The two signal lines are called `SDA` and `SCL`. Each device should have an open-drain connection to those lines, as these, together with their pull-up resistors perform a wired-AND. The function of those lines is as follows:

`SCL` Serial Clock Line
The master applies it's clock to this line to synchronise with a slave. The slave can pull this line low to stretch the clock.

`SDA` Serial Data Line
This line carries the data between master and slave and is used for handshaking.

A transaction involves a device becoming master, addressing a slave, setting the read/write bit, receiving or sending some data and ending the transaction. A device receiving data may use so called `CLOCK STRETCHING` to slow down the transmitter.

Becoming master of the bus: In order to become master, the bus has to be free, then a device can issue a `START` condition to the bus. This means creating a negative-going edge on the bus.

Sending an address or data: During a transaction, the transmitter will put it's data onto the bus during the low part of the clock cycle, the receiver will latch this data during the high phase of the clock cycle. After eight bits, the transmitter will leave the `SDA` line floating and the receiver has to pull it low to acknowledge the reception. If there was an acknowledge, the transmitter will sent the next byte of data, if the receiver fails to pull the data line low, this means there was a not-acknowledge (`NACK`) and the master will issue either a `STOP` condition, meaning a positive-going edge, become a slave again and free up the bus or issue a repeated `START` condition and start another transaction. See figure 3 for an example.

`CLOCK STRETCHING`: If a slave cannot handle the transmission speed, it may slow down the transmission by prolonging the low phase of the `SCL` line. The master will notice that the clock line did not rise to a high level when it turned of it's transistor pulling the line low, and will wait until the receiver releases the line before starting the next clock cycle and latching the next bit of data as receiver, or applying the next bit of data upon the bus as transmitter. Therefore `CLOCK STRETCHING` enables devices of different speed grades to communicate with each other at the highest speed supported by both devices, but also allows a receiver which is "distracted", e.g. because it is serving an interrupt for a short time, to suspend a transaction. However, if a receiver is not able to accept new data, because a buffer is full, or some time-consuming calculations have to be done, it should `NACK` the reception of a byte of data, to tell the transmitter to abort the transaction and free the bus.

B. Collision Detection and bus arbitration

Collision detection and arbitration is implemented using the wired-AND functionality of the bus. Devices operating as masters have to probe the `SDA` line while transmitting. If two devices try to become master of the bus, both will issue a start condition and then begin to drive the clock line and send the address of the slave they want to access followed by the read/write bit and possibly their own data in case of a write. While both masters send exactly the same bit sequence, both will be masters. However, if one master sends a "1" while the other sends a "0", the master sending the "1" will recognise that the data line is also driven by another master (because the data line is at a logic low due to the wired-AND) therefore lose the arbitration and silently become a slave for the rest of the transaction. Data integrity is ensured because the losing master became a slave before any race condition could occur.

C. 10-bit Addressing

The original specification of I^2C had 7 bits of address information and one bit specifying whether a read or write was to be performed. This was to be the first byte sent. With 16 addresses reserved for various reasons listed in figure I, this left a total of 112 distinguishable addresses. When I^2C gained popular acceptance, a need for further addresses arose, and thus 10-bit addressing was introduced. While there will seldom be that many devices on one bus, address clashes can occur with just a few devices, because the addresses are hardcoded, even though some devices allow programing one or two bits by tying pins to ground or V_{cc}. Figure 5 shows the pattern of 10-bit addressing.

To retain downward compatibility, 10-bit addressing uses a concatenation of 11110 + the first 2 bits of the 10-bit address + the $R/\overline{W}$-bit as the first transmitted byte, with the remaining 8 bits of the address being the second transmitted byte. 11110XX were reserved addresses, to which no device was allowed to respond. A device which utilises 10-bit addressing will acknowledge the reception of the first byte, if the first two bits of the match. If later on, the rest of the address matches, too, it will acknowledge the second byte as well and be addressed,

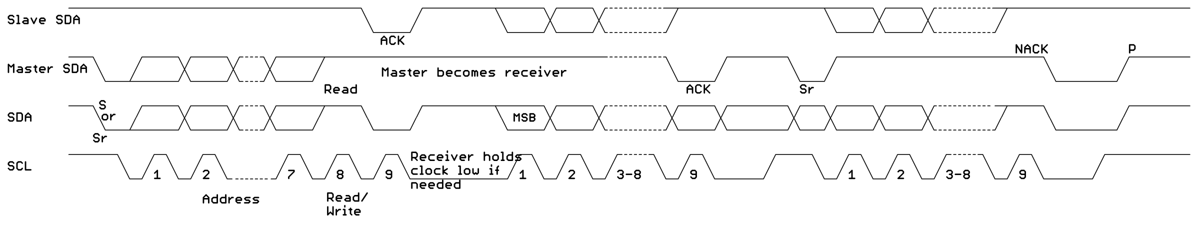
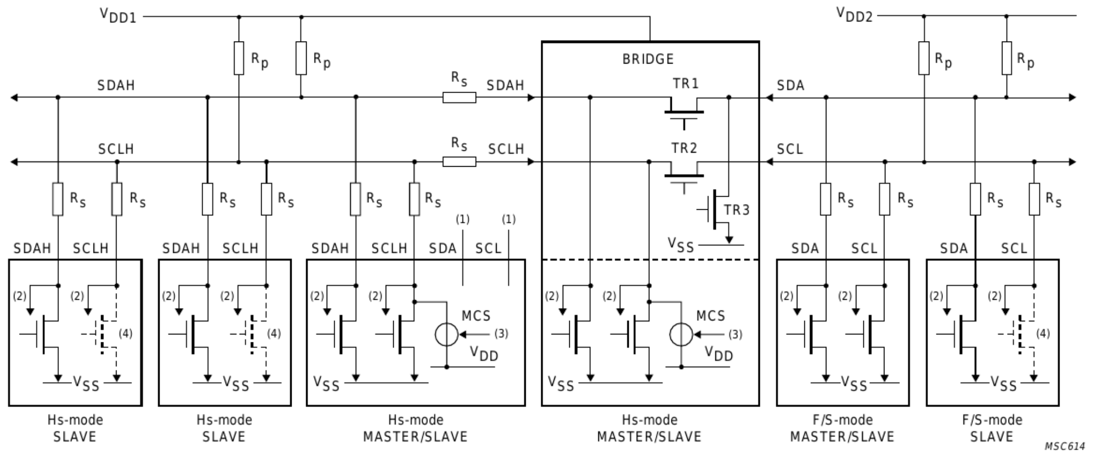


Fig. 3. An example of an I^{2}C read access.



(1) Bridge not used. SDA and SCL may have an alternative function.
(2) To input filter.
(3) Only the active master can enable its current-source pull-up circuit.
(4) Dotted transistors are optional open-drain outputs which can stretch the serial clock signal SCL or SCLH.

Fig. 4. Mixing High-Speed and Full-Speed/Standard-Speed I^{2}C devices.[5]

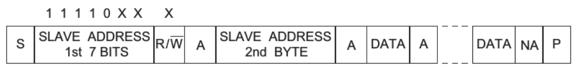


Fig. 5. Structure of a transaction with 10-bit addressing. Adapted from [6]

else it will not acknowledge the second byte and retreat from the transaction.

D. Higher Communication Speeds

In Standard-Mode, communication at up to 100 kbit/s is possible, Fast-Mode provides 400 kbit/s and Fast-Mode Plus allows up to 1 Mbit/s. To achieve this increase in speed, some timing restraints were loosened and for the Fast-Mode Plus, the drive strength of the devices was increases up to tenfold, but the protocol remained unchanged except for the increased speed.

The next speed step is at 3.4 Mbit/s and is called High-Speed Mode. This mode is not backwards compatible, the changes include:

TABLE I
RESERVED I^{2}C ADDRESSES.[6]

Address	R/$\overline{W}$-Bit	Description
0000 000	0	general call address
0000 000	1	START byte
0000 001	X	CBUS address
0000 010	X	reserved for different bus format
0000 011	X	reserved for future purposes
0000 1XX	X	Hs-mode master code
1111 1XX	1	device ID
1111 0XX	X	10-bit slave addressing

– A pull-up current source for the signal lines in each master to reduce the signal rise time.
– Spike suppression and schmitt triggers have been added to the inputs.
– Clock stretching may only occur after the `ACK` bit, during a transfer, the slaves may not interfere with the clock signal.

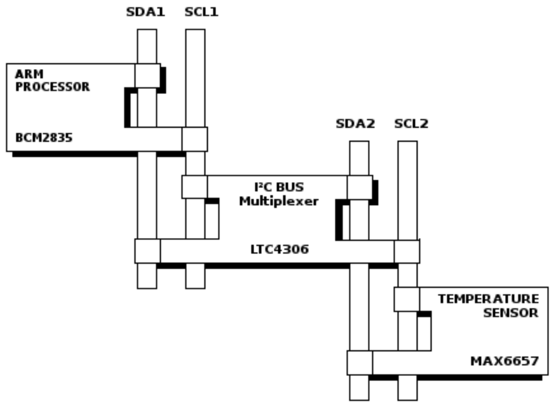


Fig. 6. Block diagram of the I^{2}C test setup.

– To initiate a High-Speed Mode transaction, the master will first have to transmit a master code while operating in Fast-Mode. This code, which is 0000 1XX, will tell the other devices that the master would like to start a High-Speed-Mode transfer.
– No bus arbitration is possible while in High-Speed-Mode. Arbitration happens exclusivly during the Fast-Mode-transmission of the master code, whose last three digits can be chosen by the system designer. This way, up to 7 high-speed masters can operate on the bus and complete arbitration within the transmission of the first byte.

To increase the speed even further, there is an Ultra-Fast-Mode with up to 5 Mbit/s. Although the bus protocol remains the same as with the other modes, only unidirectional communication is supported, and Ultra-Fast-Mode devices use push-pull output stages, making them incompatible with other I^{2}C-standards.

IV. Recording I^{2}C

As a practical example, an I^{2}C transaction was measured with an oscilloscope. For this, a test board with a temperature sensor was attached to a raspberry pi running a fresh install of the Raspbian Linux distribution. On this board, a bus multiplexer is attached between the raspberry pi and the sensor. See figure 6 for a connection scheme. To use the I^{2}C bus, the appropriate kernel modules (`i2c-bcm2708` and `i2c-dev`) need to be started and the package `i2c-tools` needs to be installed. Then, the following session was started, and the last transfer was recorded with an oscilloscope.

```
# Tell the multiplexer to connect the
# correct bus to the pi.
pi@raspberrypi ~ $ i2cset 1 73 3 64
WARNING! This program can confuse your
I2C bus, cause data loss and worse!
I will write to device file
/dev/i2c-1, chip address 0x49, data
address 0x03, data 0x40, mode byte.
Continue? [Y/n] y
# Read the temperature sensor.
pi@raspberrypi ~ $ i2cget -y 1 76 0
0x1b
# A thumb was placed on the sensor
# to warm it up a little
pi2@raspberrypi ~ $ i2cget -y 1 76 0
0x1d
```

Figure 7 shows the merged screenshots from the oscilloscope, annotated with the bus actions. The read transfer is broken down into two consecutive parts: First, the master writes the address it wants to read to the slave, which will then, after a repeated start and a read command, write the desired data on the bus.

V. Comparison of SPI and I^{2}C

Please refer to figure 8 for an illustration of the comparison. To compare things, categories of comparison are needed. There are the six categories considered in this comparison:

Speed: When it comes to speed, SPI is a clear winner with some devices communicating at up to 60 Mbit/s.

Simplicity of use: To communicate with a device attached to the I^{2}C bus, only it's address is needed. For SPI, the maximum clock speed, polarity and phase of the clock signal need to be known. So, in this category, I^{2}C is a winner.

Simplicity at chip-level: I^{2}C requires a comparatively complex logic setup for bus arbitration and clock stretching. On the other hand, to implement an SPI-device, not much more than a shift register is needed.

Simplicity at Board Level: I^{2}C and SPI will not need much space on a PCB, with I^{2}C using 2 wires and 2 resistors and SPI using 3 lines and one more for each slave. The difference is marginal, but I^{2}C wins in this category.

Versatility: Both SPI and I^{2}C are used in a great number of applications. I^{2}C allows easier attachment of components to the bus, while SPI is easier to comprehend and debug. Depending on the point of view, both buses could win, so this is a draw.

Robustness: Neither of the bus systems implement error checking or correction at protocol level. Further, with I^{2}C, the signal lines are only driven by a relatively weak pull-up resistor during the high phase, making them more susceptible to noise, while signal lines are always actively driven with SPI.

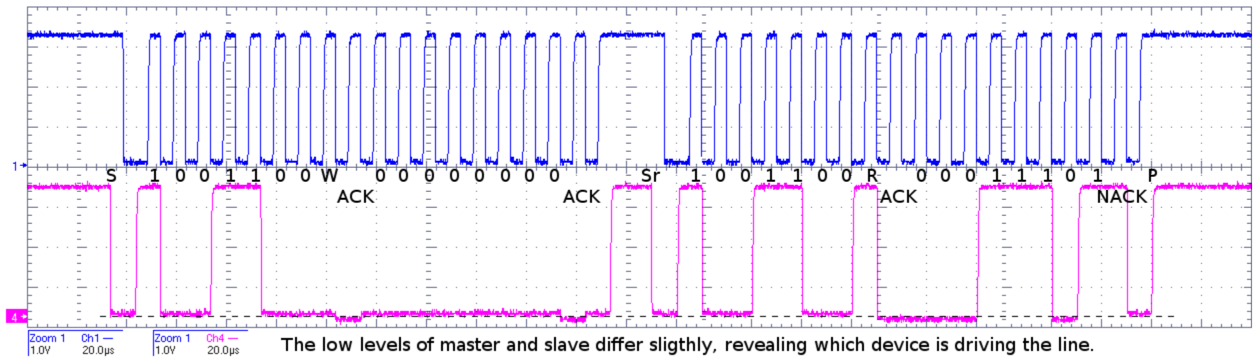


Fig. 7. An I^2C read access measured with an oscilloscope. The upper curve represents the `SCL` line, the lower curve shows the `SDA` line.

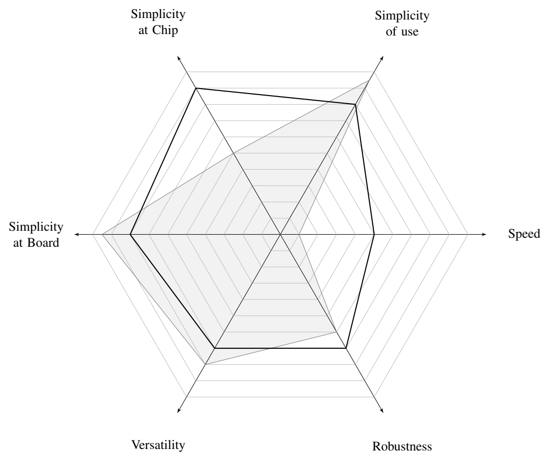


Fig. 8. Comparison of the buses. Filled: I^2C, Thick line: SPI.

As demonstrated, both bus systems have strengths and weaknesses. There can be no definite answer telling which bus is better as both buses were designed for different requirements. SPI is mostly used to transfer data to and from peripheral devices, I^2C is prevalently used for more configuration centric communication. Systems, which can tolerate the lower communication speeds profit by the slightly smaller complexity of I^2C, whereas systems requiring a higher data throughput or which rely on a guaranteed maximum time until a message has been sent, will probably prefer SPI.

VI. Conclusion

Both I^2C and SPI are very mature, while still up-to-date bus systems for low-speed, low-complexity interconnection of integrated circuits. With their wide acceptance and when used as intended, both I^2C and SPI are recommendable and reliable bus systems.

References

[1] Datasheet of the Motorola MC68HC11A8 microcontroller describing the SPI bus.
Last downloaded 2014-01-10
http://cache.freescale.com/files/microcontrollers/doc/data_sheet/MC68HC11A8.pdf

[2] Datasheet of the Motorola MC68HCP11A1VP microcontroller
Last downloaded 2014-01-10
http://pdf.datasheetcatalog.com/datasheet/motorola/MC68HCP11A1VP.pdf

[3] SPI Block Guide V03.06
Last downloaded 2014-01-10
http://www.ee.nmt.edu/~teare/ee308l/datasheets/S12SPIV3.pdf

[4] Datasheet detailing Microwire
Last downloaded 2014-01-11
http://www.ti.com/lit/an/snoa743/snoa743.pdf

[5] Phillips Semiconductors: The I^2C-Bus Specification, 2000
Last downloaded 2014-01-10
http://www.cs.unc.edu/Research/stc/FAQs/Interfaces/I2C-BusSpec-V2.1.pdf

[6] NXP Semiconductors: UM10204 I2C-bus specification and user manual
Last downloaded 2014-01-10
http://www.nxp.com/documents/user_manual/UM10204.pdf

[7] Homepage for PMBus, detailing it's connection to I^2C
Last visited 2014-01-11
http://pmbus.org/about/pmbusancestry

[8] Specification of SMBus
Last downloaded 2014-01-11
http://smbus.org/specs/smbus20.pdf

[9] Homepage about I^2C detailing TWI
Last visited 2014-01-11
http://www.i2c-bus.org/twi-bus

[10] Example of a datasheet using the term "Two Wire Interface" instead of I^2C
Last downloaded 2014-01-11
http://www.atmel.com/Images/2466S.pdf