General

Serial Peripheral Interface (SPI) & Inter-IC (IC2) (SPI_I2C)

Introduction

This Application note provide a general view on SPI and I^{2}C, comparison of the two communication standard is also detailed.

The H8/300L Super Low Power (SLP) series of 8-bit mircocontrollers has at least one Serial Communication Interface (SCI) channel. This communication interface channel can also support full standard synchronous communications. Serial Peripheral Interface (SPI) and Inter-IC (I^{2}C), both are synchronous communications provide good support for communication with slow peripheral devices that are accessed intermittently. The SPI and I2C can be emulated using SCI channel or I/O port.

Target Device

General

Contents

1. Serial Peripheral Interface (SPI™)

1.1 SPI Overview

SPI is a general-purpose synchronous serial interface. During an SPI transfer, transmit and receive data is simultaneously shifted out and in serially. A serial clock line synchronizes the shifting and sampling of the information on two serial data lines.

Motorola created the SPI port in the mid 1980's to use in their microcontroller product families. The SPI is mainly used to allow a microcontrollers to communicate with peripheral devices such as E^2PROMs.

SPI devices communicate using a master-slave relationship. Due to its lack of built-in device addressing, SPI requires more effort and more hardware resources than I^2C when more than one slave is involved. But SPI tends to be simpler and more efficient than I^2C in point-to-point (single master, single slave) applications for the very same reason; the lack of device addressing means less overhead.

1.2 SPI Detail

SPI is a serial bus standard established by Motorola and supported in silicon products from various manufacturers. SPI interfaces are available on popular communication processors and microcontrollers. It is a synchronous serial data link that operates in full duplex (signals carrying data go in both directions simultaneously).

Devices communicate using a master/slave relationship, in which the master initiates the data frame. When the master generates a clock and selects a slave device, data may be transferred in either or both directions simultaneously. In fact, as far as SPI is concerned, data are always transferred in both directions. It is up to the master and slave devices to know whether a received byte is meaningful or not.

So a device must discard the received byte in a "transmit only" frame or generate a dummy byte for a "receive only" frame.

SPI specifies four signals: clock (SCK_1); master data output, slave data input (SI_1); master data input, slave data output (SO_1); and chip select (CS). Figure 1 shows these signals in a single-slave configuration. SCK_1 is generated by the master and input to all slaves. SI_1 carries data from master to slave. SO_1 carries data from slave back to master. A slave device is selected when the master asserts its CS signal.

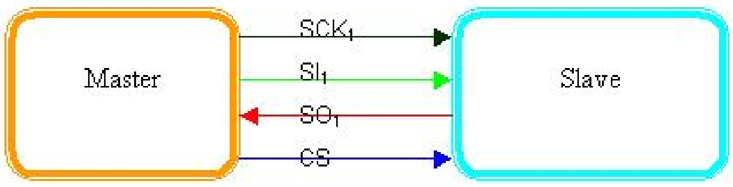


Figure 1: Single master, single slave SPI implementation

If multiple slave devices exist, the master generates a separate slave select signal for each slave. These relationships are illustrated in Figure 2.

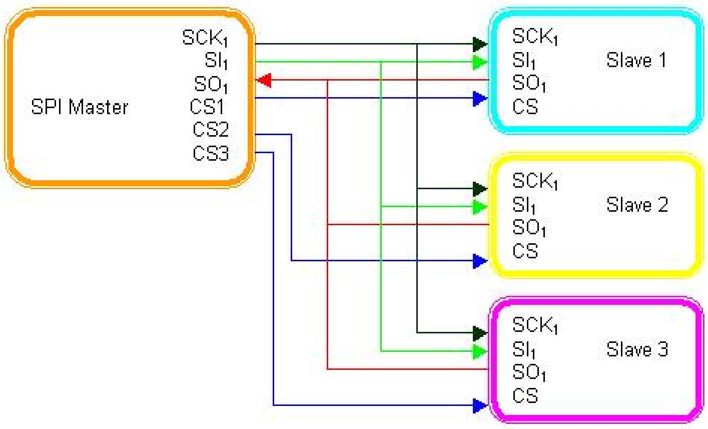


Figure 2: Single master, multiple slave SPI implementation

The master generates slave select signals using general-purpose discrete input/output pins or other logic. This consists of old-fashioned bit banging and can be pretty sensitive. You have to time it relative to the other signals and ensure, for example, that you don't toggle a select line in the middle of a frame.

While SPI doesn't describe a specific way to implement multi-master systems, some SPI devices support additional signals that make such implementations possible. However, it's complicated and usually unnecessary, so it's not often done.

A pair of parameters called clock polarity (CPOL) and clock phase (CPHA) determine the edges of the clock signal on which the data are driven and sampled. Each of the two parameters has two possible states, which allows for four possible combinations, all of which are incompatible with one another. So a master/slave pair must use the same parameter pair values to communicate. If multiple slaves are used that are fixed in different configurations, the master will have to reconfigure itself each time it needs to communicate with a different slave.

SPI does not have an acknowledgement mechanism to confirm receipt of data. In fact, without a communication protocol, the SPI master has no knowledge of whether a slave even exists. SPI also offers no flow control. If you need hardware flow control, you might need to do something outside of SPI.

Slaves can be thought of as input/output devices of the master. SPI does not specify a particular higher-level protocol for master-slave dialog. In some applications, a higher-level protocol is not needed and only raw data are exchanged. An example of this is an interface to a simple codec. In other applications, a higher-level protocol, such as a command-response protocol, may be necessary. Note that the master must initiate the frames for both its command and the slave's response.

1.3 Data and Control Lines of the SPI

The SPI requires two control lines (CS and SCK) and two data lines (SI and SO).

With CS (Chip-Select) the corresponding peripheral device is selected. This pin is mostly active-low. In the unselected state the SO lines are hi-Z and therefore inactive. The master decides with which peripheral device it wants to communicate. The clock line SCLK is brought to the device whether it is selected or not. The clock serves as synchronization of the data communication.

The majority of SPI devices provide these four lines. Sometimes it happens that SDI and SDO are multiplexed, for example in the temperature sensor LM74 from National Semiconductor, or that one of these lines is missing. A peripheral device which must or can not be configured, requires no input line, only a data output. As soon as it gets selected it starts sending data. In some ADCs therefore the SDI line is missing (e.g. MCCP3001 from Microchip).

There are also devices that have no data output. For example LCD controllers (e.g. COP472-3 from National Semiconductor), which can be configured, but cannot send data or status messages.

1.4 SPI Configuration

Because there is no official specification, what exactly SPI is and what not, it is necessary to consult the data sheets of the components one wants to use. Important are the permitted clock frequencies and the type of valid transitions.

There are no general rules for transitions where data should be latched. Although not specified by Motorola, in practice four modes are used. These four modes are the combinations of CPOL and CPHA. In table 1, the four modes are listed.

SPI-mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Table 1: SPI Modes

If the phase of the clock is zero, i.e. CPHA = 0, data is latched at the rising edge of the clock with CPOL = 0, and at the falling edge of the clock with CPOL = 1. If CPHA = 1, the polarities are reversed. CPOL = 0 means falling edge, CPOL = 1 rising edge.

The micro controllers from Motorola allow the polarity and the phase of the clock to be adjusted. A positive polarity results in latching data at the rising edge of the clock. However data is put on the data line already at the falling edge in order to stabilize. Most peripherals which can only be slaves, work with this configuration. If it should become necessary to use the other polarity, transitions are reversed.

2. Inter-IC (I^2C^{TM})

2.1 I^2C Overview

The I C-bus is developed by Philips to maximize hardware efficiency and circuit simplicity. The I^2C interface is a simple master/slave type interface. Simplicity of the I C system is primarily due to the bi-directional 2-wire design, a serial data line (SDA) and serial clock line (SCL), and to the protocol format.

I^2C is appropriate for interfacing to devices on a single board, and can be stretched across multiple boards inside a closed system, but not much further. An example is a host CPU on a main embedded board using I^2C to communicate with user interface devices located on a separate front panel board. A second example is SDRAM DIMMs, which can feature an I^2C EEPROM containing parameters needed to correctly configure a memory controller for that module.

2.2 I^2C Detail

I^2C is a two-wire serial bus, as shown in Figure 3. There's no need for chip select or arbitration logic, making it cheap and simple to implement in hardware.

The two I^2C signals are serial data (SDA) and serial clock (SCL). Together, these signals make it possible to support serial transmission of 8-bit bytes of data-7-bit device addresses plus control bits-over the two-wire serial bus. The device that initiates a transaction on the I^2C bus is termed the master.

The master normally controls the clock signal. A device being addressed by the master is called a slave.

In a bind, an I^2C slave can hold off the master in the middle of a transaction using what's called clock stretching (the slave keeps SCL pulled low until it's ready to continue). Most I^2C slave devices don't use this feature, but every master should support it.

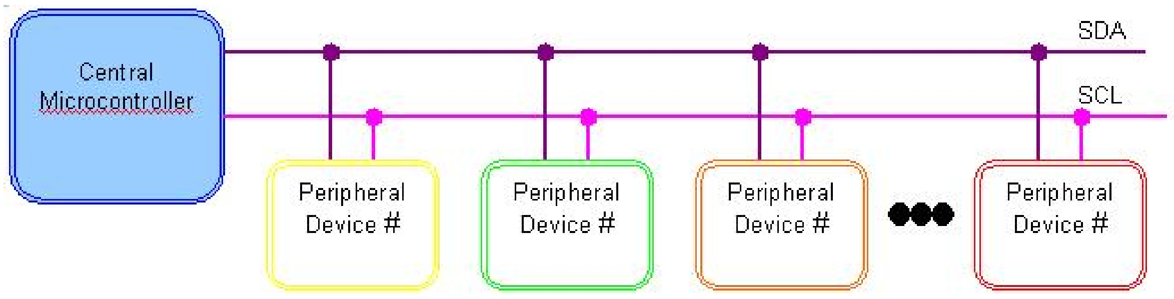


Figure 3: I^2C has two lines in total

The I^2C protocol supports multiple masters, but most system designs include only one. There may be one or more slaves on the bus. Both masters and slaves can receive and transmit data bytes.

Each I^2C-compatible hardware slave device comes with a predefined device address, the lower bits of which may be configurable at the board level. The master transmits the device address of the intended slave at the beginning of every transaction. Each slave is responsible for monitoring the bus and responding only to its own address. This addressing scheme limits the number of identical slave devices that can exist on an I^2C bus without contention, with the limit set by the number of user-configurable address bits (typically two bits, allowing up to four identical devices).

2.3 I^2C Protocol

The I^2C bus physically consists of 2 active wires and a ground connection. The active wires, called SDA and SCL, are both bi-directional. SDA is the **S**erial **DA**ta line, and SCL is the **S**erial **CL**ock line.

Every device hooked up to the bus has its own unique address, no matter whether it is an MCU, LCD driver, memory, or ASIC. Each of these chips can act as a receiver and/or transmitter, depending on the functionality. Obviously, an LCD driver is only a receiver, while a memory or I/O chip can be both transmitter and receiver.

The I^2C bus is a multi-master bus. This means that more than one IC capable of initiating a data transfer can be connected to it. The I^2C protocol specification states that the IC that initiates a data transfer on the bus is considered the Bus Master*,* which generally is a microcontrollers. Consequently, at that time, all the other ICs are regarded to be Bus Slaves.

First, the MCU will issue a *START* condition. This acts as an 'Attention' signal to all of the connected devices. All ICs on the bus will listen to the bus for incoming data.

Then the MCU sends the *ADDRESS* of the device it wants to access, along with an indication whether the access is a Read or Write operation (Write in our example). Having received the address, all IC's will compare it with their own address. If it doesn't match, they simply wait until the bus is released by the stop condition (see below). If the address matches, however, the chip will produce a response called the *ACKNOWLEDGEMENT* signal.

Once the MCU receives the acknowledgement, it can start transmitting or receiving *DATA*. In our case, the MCU will transmit data. When all is done, the MCU will issue the *STOP* condition. This is a signal that the bus has been released and that the connected ICs may expect another transmission to start any moment.

In general, the protocol for I^2C as illustrated in Figure 4:

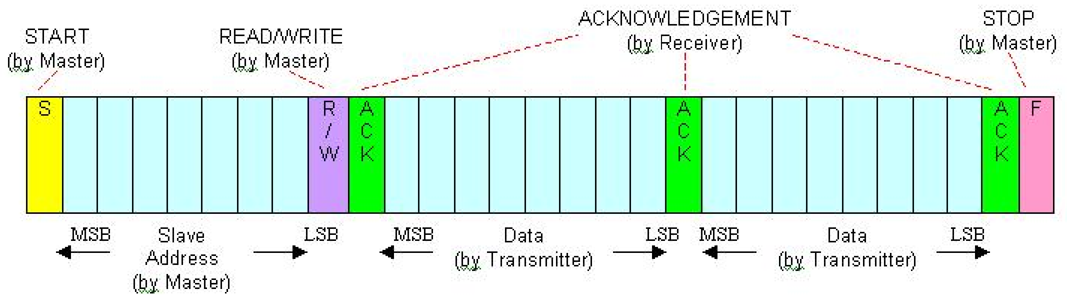
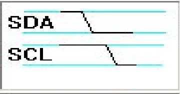
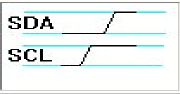


Figure 4: I2C Communication

** The protocol may have unique conditions depending on the bus. It's advisable for user to understand the physical structure and the hardware of the bus before implementing I^2C.*

2.4 I^{2}C Configuration

i. The Start and Stop Configuration

Start		The chip issuing the Start condition first pulls the SDA (data) line low, and next pulls the SCL (clock) line low.
Stop		The Bus Master first releases the SCL and then the SDA line.

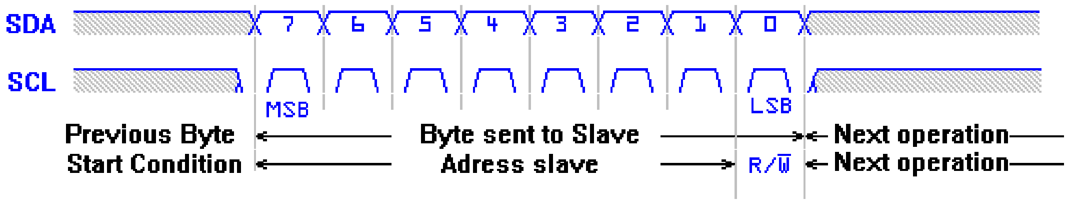
A few note about start and stop conditions:

- *A single message can contain multiple Start conditions. The use of this so-called "repeated start" is common in I^{2}C.*
- *A Stop condition ALWAYS denotes the END of a transmission. Even if it is issued in the middle of a transaction or in the middle of a byte. It is "good behaviour" for a chip that, in this case, it disregards the information sent and resumes the "listening state", waiting for a new start condition.*

ii) Transmitting a byte to a slave

Once the START condition has been sent, a byte can be transmitted by the MASTER to the SLAVE.

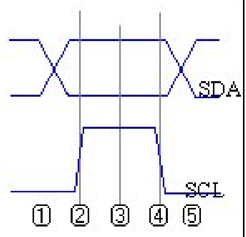
This first byte after a start condition will identify the slave on the bus (address) and will select the mode of operation. The meaning of all following bytes depends on the slave.



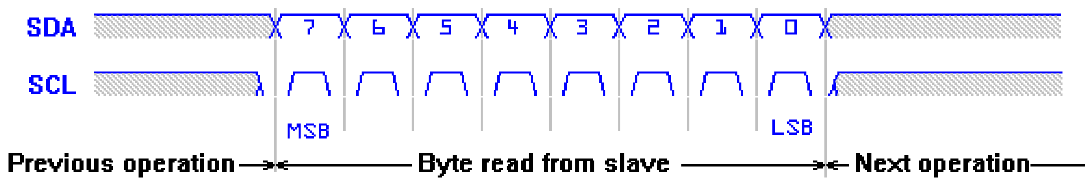
iii) Receiving a byte from a slave

Once the slave has been addressed and the slave has acknowledged this, a byte can be received from the slave if the R/W bit in the address was set to READ (set to '1').

The protocol syntax is the same as in transmitting a byte to a slave, except that now the master is not allowed to touch the SDA line. Prior to sending the 8 clock pulses needed to clock in a byte on the SCL line, the master releases the SDA line. The slave will now take control of this line. The line will then go high if it wants to transmit a '1' or, if the slave wants to send a '0', remain low.

	All the master has to do is generate a rising edge on the SCL line (2), read the level on SDA (3) and generate a falling edge on the SCL line (4). The slave will not change the data during the time that SCL is high. (Otherwise a Start or Stop condition might inadvertently be generated.) During (1) and (5), the slave may change the state of the SDA line.

In total, this sequence has to be performed 8 times to complete the data byte. Bytes are **always** transmitted **MSB first**.



The meaning of all bytes being read depends on the slave. There is no such thing as a "universal status register". You need to consult the data sheet of the slave being addressed to know the meaning of each bit in any byte transmitted.

iv) Getting Acknowledgement from a Slave

When an address or data byte has been transmitted onto the bus, then this must be ACKNOWLEDGED by the slave(s). In case of an address: If the address matches its own, then only that slave will respond to the address with an ACK. In case of a byte transmitted to an already addressed slave, then that slave will respond with an ACK as well.

The slave that is going to give an ACK pulls the SDA line low immediately after reception of the 8th bit transmitted, or, in case of an address byte, immediately after evaluation of its address. In practical applications this will not be noticeable

This means that as soon as the master pulls SCL low to complete the transmission of the bit (1), SDA will be pulled low by the slave (2).

SDA

The master now issues a clock pulse on the SCL line (3). the slave will release the SDA line upon completion of this clock pulse (4).

SCL

3. SPI vs. I^2C

Although both SPI and I^2C provide good support for communication with slow peripheral devices that are accessed intermittently, each of the way of communication have its own advantages towards each other.

SPI is better suited than I^2C for applications that are naturally thought of as data streams (as opposed to reading and writing addressed locations in a slave device). An example of a "stream" application is data communication between microprocessors or digital signal processors. Another is data transfer from analog-to-digital converters.

SPI can also achieve significantly higher data rates than I^2C which is limited to 400KHz in most cases. SPI-compatible interfaces often range into the tens of megahertz. SPI really gains efficiency in applications that take advantage of its duplex capability, such as the communication between a "codec" (coder-decoder) and a digital signal processor, which consists of simultaneously sending samples in and out.

Due to SPI lack of built-in device addressing, it requires more effort and more hardware resources than I^2C when more than one slave is involved. The disadvantage here lies that it is a three-wire interface and if you are having more than 1 device, then you have to provide each device with separate Chip Select pins (CS).

But SPI tends to be simpler and more efficient than I^2C in point-to-point (single master, single slave) applications for the very same reason; the lack of device addressing means less overhead.

On the other hand, I^2C requires only two wires to implement and has a unique address so that a master/slave relationship can be maintained compare to SPI which needed three wires to implement the addressing mode.

I^2C also offers better support for communication with on-board devices that are accessed on an occasional basis. I^2C's competitive advantage over other low-speed short-distance communication schemes is that its cost and complexity don't scale up with the number of devices on the bus because of the generic nature of the bus interface.

Besides, the complexity of the supporting I^2C software components can be significantly higher than that of several competing schemes such as SPI in a very simple configuration. With its built-in addressing scheme and straightforward means to transfer strings of bytes, I^2C is an elegant, minimalist solution for modest, "inside the box" communication needs.

I^2C is also a true multi-master bus because it has collision detection and arbitration to prevent data corruption if two or more masters simultaneously initiate data transfer. Furthermore, I^2C also preserve data integrity by filtering rejects spikes on the bus data line.

4. Summary

SPI	I^2C
1) Three bus lines are required; a data input line (SI_1), a data output line (SO_1) and a serial clock line (SCK_1) [plus 1 Chip Select (CS)]	1) Two bus lines are required; a serial data line (SDA) and a serial clock line (SCL)
2) No official specification (component dependent)	2) With official specification (I^2C protocol created by Philips)
3) Higher data rates (up to 10 MHz or more)	3) Support transfer speeds of around 100kHz (original standard, or 400kHz using the most recent standard)
4) More efficient in point-to-point (single master, single slave) applications	4) More efficient in multi-master, multi-slave applications
5) Lack of built-in device addressing	5) Built-in addressing scheme and straightforward
6) Does not have an acknowledgement mechanism to confirm receipt of data.	6) Have an acknowledgement mechanism to confirm receipt of data
7) Less overhead when handling point-to-point application	7) More overhead when handling point-to-point application
8) Suited better for applications that are naturally thought of as data streams	8) Suited better for communication with on-board devices that are accessed on an occasional basis.

5. Implementation Feasibility

Any SLP series can be supported by the I^2C bus and SPI, either by communicating using Serial Communication Interface (SCI) or I/O ports.

For implementing I2C with SCI, it requires extra I/O lines and components.

*Please refer to Application Note on "Interfacing with EEPROM with emulating SPI" for example of using **SCI to implement SPI**.*

*For implementing **I2C using I/O ports**, please refer to Application Note on "Interfacing With EEPROM with emulating I^2C".*

Reference

1. http://www.embedded.com/97/feat9711.htm
2. http://www.mct.net/faq/spi.html
3. http://www.epanorama.net/links/serialbus.html#spi
4. http://www.mcumaster.com/hc11/Block/SPI/spi.html#Interrupts
5. http://www.cdcentre.demon.co.uk/teletext/i2c.htm
6. The *I^2C-Bus Specification (Version 2.1)*, January 2000, Philips Semiconductor

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Sep.03	-	First edition issued